ETSI GR NFV-REL 013 V5.1.1 (2023-02)

GROUP REPORT

Network Functions Virtualisation (NFV) Release 5; Reliability; Report on cognitive use of operations data for reliability

Disclaimer

The present document has been produced and approved by the Network Functions Virtualisation (NFV) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference

DGR/NFV-REL013

Keywords

availability, NFV

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Important notice

The present document can be downloaded from:
http://www.etsi.org/standards-search

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the prevailing version of an ETSI deliverable is the one made publicly available in PDF format at www.etsi.org/deliver.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx

If you find errors in the present document, please send your comment to one of the following services:
https://portal.etsi.org/People/CommiteeSupportStaff.aspx

If you find a security vulnerability in the present document, please report it through our Coordinated Vulnerability Disclosure Program:
https://www.etsi.org/standards/coordinated-vulnerability-disclosure

Notice of disclaimer & limitation of liability

The information provided in the present deliverable is directed solely to professionals who have the appropriate degree of experience to understand and interpret its content in accordance with generally accepted engineering or other professional standard and applicable regulations.
No recommendation as to products and services or vendors is made or should be implied.
No representation or warranty is made that this deliverable is technically accurate or sufficient or conforms to any law and/or governmental rule and/or regulation and further, no representation or warranty is made of merchantability or fitness for any particular purpose or against infringement of intellectual property rights.
In no event shall ETSI be held liable for loss of profits or any other incidental or consequential damages.

Any software contained in this deliverable is provided "AS IS" with no warranties, express or implied, including but not limited to, the warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property rights and ETSI shall not be held liable in any event for any damages whatsoever (including, without limitation, damages for loss of profits, business interruption, loss of information, or any other pecuniary loss) arising out of or related to the use of or inability to use the software.

Copyright Notification

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

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 aims at studying how operations data (KPIs, metrics, alarm notifications, event logs, debug information) can be exploited to ensure the availability and reliability of NFV-MANO and of the network services it manages using data analysis/data driven techniques. This includes, among others, the use of machine learning to find patterns for cases where detailed semantics information is unavailable (e.g. due to confidentiality) or the amount of data is overwhelming. Use cases are created describing how the information can be used offline (for example for root cause analysis and predictive maintenance resulting in, e.g. creation and/or changes of deployment flavours) and online (to identify appropriate LCM operations and policy changes in order to achieve the intended service availability objectives).

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] ETSI GS NFV-IFA 005 (V3.4.1): "Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Or-Vi reference point - Interface and Information Model Specification".

[i.2] ETSI GS NFV-IFA 006 (V3.4.1): "Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Vi-Vnfm reference point - Interface and Information Model Specification".

[i.3] ETSI GS NFV-IFA 007 (V3.4.1): "Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Or-Vnfm reference point - Interface and Information Model Specification".

[i.4] ETSI GS NFV-IFA 008 (V3.4.1): "Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Ve-Vnfm reference point - Interface and Information Model Specification".

[i.5] ETSI GS NFV-IFA 013 (V3.4.1): "Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Os-Ma-Nfvo reference point - Interface and Information Model Specification".

[i.6] ETSI GS NFV-IFA 030 (V3.4.1): "Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Multiple Administrative Domain Aspect Interfaces Specification".

[i.7] ETSI GS NFV-IFA 031 (V3.4.1): "Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Requirements and interfaces specification for management of NFV-MANO".

[i.8] ETSI GS NFV-IFA 032 (V3.4.1): "Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Interface and Information Model Specification for Multi-Site Connectivity Services".

[i.9] ETSI GR NFV-IFA 015 (V3.4.1): "Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Report on NFV Information Model".

[i.10] ETSI GR NFV-REL 010 (V3.1.1): "Network Functions Virtualisation (NFV) Release 3; Reliability; Report on NFV Resiliency for the Support of Network Slicing".

[i.11] Ivar Thokle Hovden: "Optimizing Artificial Neural Network Hyperparameters and Architecture", University of Oslo, 2019.

NOTE: Available at www.mn.uio.no/fysikk/english/people/aca/ivarth/works/in9400_nn_hpo_nas_hovden_r2.pdf.

[i.12] ETSI GS NFV-REL 005 (V1.1.1): "Network Functions Virtualisation (NFV); Accountability; Report on Quality Accountability Framework".

[i.13] ETSI GR NFV 003 (V1.6.1): "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV".

[i.14] S. Azadiabad et al.: "Availability and Service Disruption of Network Services: from High-level Requirements to Low-level Configuration Constraints", Elsevier Computer Standards and Interfaces, Vol 80, March 2022.

NOTE: Available at doi.org/10.1016/j.csi.2021.103565.

[i.15] ETSI GS NFV-SOL 009 (V3.5.1): "Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models: RESTful protocols specification for the management of NFV-MANO".

[i.16] Z. Li et al.: "Predictive Analysis in Network Function Virtualization", IMC'18, October 31-November 2, Boston, MA, USA.

NOTE: Available at dl.acm.org/doi/10.1145/3278532.3278547.

[i.17] I. Hadj-Kacem et al.: "Anomaly prediction in mobile networks: a data driven approach for machine learning algorithm selection", NOMS 2020, April 20-24, Budapest, Hungary.

NOTE: Available at ieeexplore.ieee.org/document/9110429.

[i.18] A.L. Dennis: "Cognitive Computing Demystified: The What, Why, and How", DATAVERSITY, February 2017.

NOTE: Available at www.dataversity.net/cognitive-computing-the-what-why-and-how/.

[i.19] B. Marr: "What Everyone Should Know About Cognitive Computing", Forbes, March 2016.

NOTE: Available at www.forbes.com/sites/bernardmarr/2016/03/23/what-everyone-should-know-about-cognitive-computing/.

[i.20] "Supervised Learning", IBM Cloud Education, August 2020.

NOTE: Available at www.ibm.com/cloud/learn/supervised-learning.

[i.21] "Unsupervised Learning", IBM Cloud Education, September 2020.

NOTE: Available at www.ibm.com/cloud/learn/unsupervised-learning.

[i.22] M. Tim Jones: "Train a software agent to behave rationally with reinforcement learning", October 2017.

NOTE: Available at developer.ibm.com/articles/cc-reinforcement-learning-train-software-agent/.

[i.23] "Machine learning", Wikipedia.

NOTE: Available at en.wikipedia.org/wiki/Machine_learning.

[i.24] R.S. Sutton and A.G. Barto: "Reinforcement Learning: An Introduction", Second Edition MIT Press, 2018.

NOTE: Available at www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf.

[i.25] T. Hastie et al.: "The elements of Statistical Learning - Data Mining, Inference, and Prediction", Second Edition, Springer, January 2017.

NOTE: Available at https://hastie.su.domains/Papers/ESLII.pdf.

[i.26] J. Ahmed et al.: "Automated diagnostic of virtualized service performance degradation", NOMS 2018, April 23-27, Taipei, Taiwan.

NOTE: Available at ieeexplore.ieee.org/document/8406234.

[i.27] G. Jung et al.: "Detecting bottleneck in n-tier IT applications through analysis", International Workshop on Distributed Systems: Operations and Management (DSOM 2006), October 23-25, Dublin, Ireland.

[i.28] D.J. Dean et al.: "UBL: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems", 9th International Conference on Autonomic Computing (ICAC'12), September 18-20, San Jose, CA, USA.

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.13] apply.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the abbreviations given in ETSI GR NFV 003 [i.13] and the following apply:

AAF	Availability Assurance Function
ACK	Acknowledgement
ADAM	Adaptive Moment Estimation
AFR	Average Failure Rate
ANN	Artificial Neural Network
APF	Anomaly Prediction Function
ASDD	Acceptable Service Data Disruption
ASDT	Acceptable Service Disruption Time
BA	Balanced Accuracy
BMU	Best Matching Unit
BW	Bandwidth
c/n/s	compute/network/storage
CA	Classification Accuracy
CoM	Composite Model
CpI	Checkpointing Interval
CSCF	Call Session Control Function
DLR	Distributed-Log Regression
DS	dissimilarity vector
FDLF	Fault Detection and Localization Function
FE	Functional Entity
FLM	Fault Localization Model

FN	False Negative
FP	False Positive
FPCA	Functional Principal Component Analysis
GRE	Generic Routing Encapsulation
HI	Health-Check Interval
ICMP	Internet Control Message Protocol
IPV4	Internet Protocol Version 4
IPV6	Internet Protocol Version 6
IPVLAN	Internet Protocol Virtual Local Area Network
LB	Load Balancing
LiReg	Linear Regression
LoReg	Logistic Regression
LSTM	Long Short-Term Memory
LTE	Long Term Evolution
MP	MultiPoint
MSCS	Multi-Site Connectivity Services
MTTR	Mean Time to Repair/Restore
NL	Latency
NN	Neural Network
NsDf	NS Deployment Flavour
NsQoS	Network service Quality of Service
ODU2	Optical Data Unit 2
PCA	Principal Components Analysis
PTP	Precision Time Protocol
RA	Required Availability
RAID	Redundant Array of Independent Disks
RCA	Root Cause Analysis
ReLU	Rectified Linear Unit
RF	Random Forest
SARSA	State-Action-Reward-State-Action
SB	Standby Capacity
SL2	Scale Level 2
SL3	Scale Level 3
SLAAC	Stateless Address Autoconfiguration
SLO	Service Level Objectives
SOM	Self-Organizing Map
SVM	Support Vector Machine
Tanh	Hyperbolic Tangent
TN	True Negative
TP	True Positive
UDP	User Datagram Protocol
vPE	Virtualised Provider Edge

4 Overview

Operations of communication networks produce huge amounts of data related to aspects such as characteristics, lifecycle, behavior or performance/fault monitoring.

In the context of the multi-vendor, multi-layer NFV architecture, the exploitation of such massive data with the use of cognitive approaches would ease the networks management, and could provide, in particular, the assurance of resilient runtime operations of these networks.

The present document studies how machine learning could be applied to NFV operations data for reliability and availability purposes. Clause 5 details the NFV architecture interfaces through which the field data may be collected, together with the operations which create these data.

Three families of data-driven techniques are described in clause 6: supervised, unsupervised and reinforcement learning. Used for operations control and management, they may help to build zero-touch control loops and pave the way to autonomous networking.

Three selected use cases for the use of operations data for reliability and availability are described in clause 7:

- Service availability assurance for meeting and maintaining a given network service availability.
- Root cause analysis, i.e. real-time fault localization for mitigation of incidents which underly detected anomalies.
- Anomaly prediction for anticipation of failures which could lead to outages.

A list of recommendations, some of which call for requirements specification, is finally provided in clause 8 for each use case.

5 Operations data

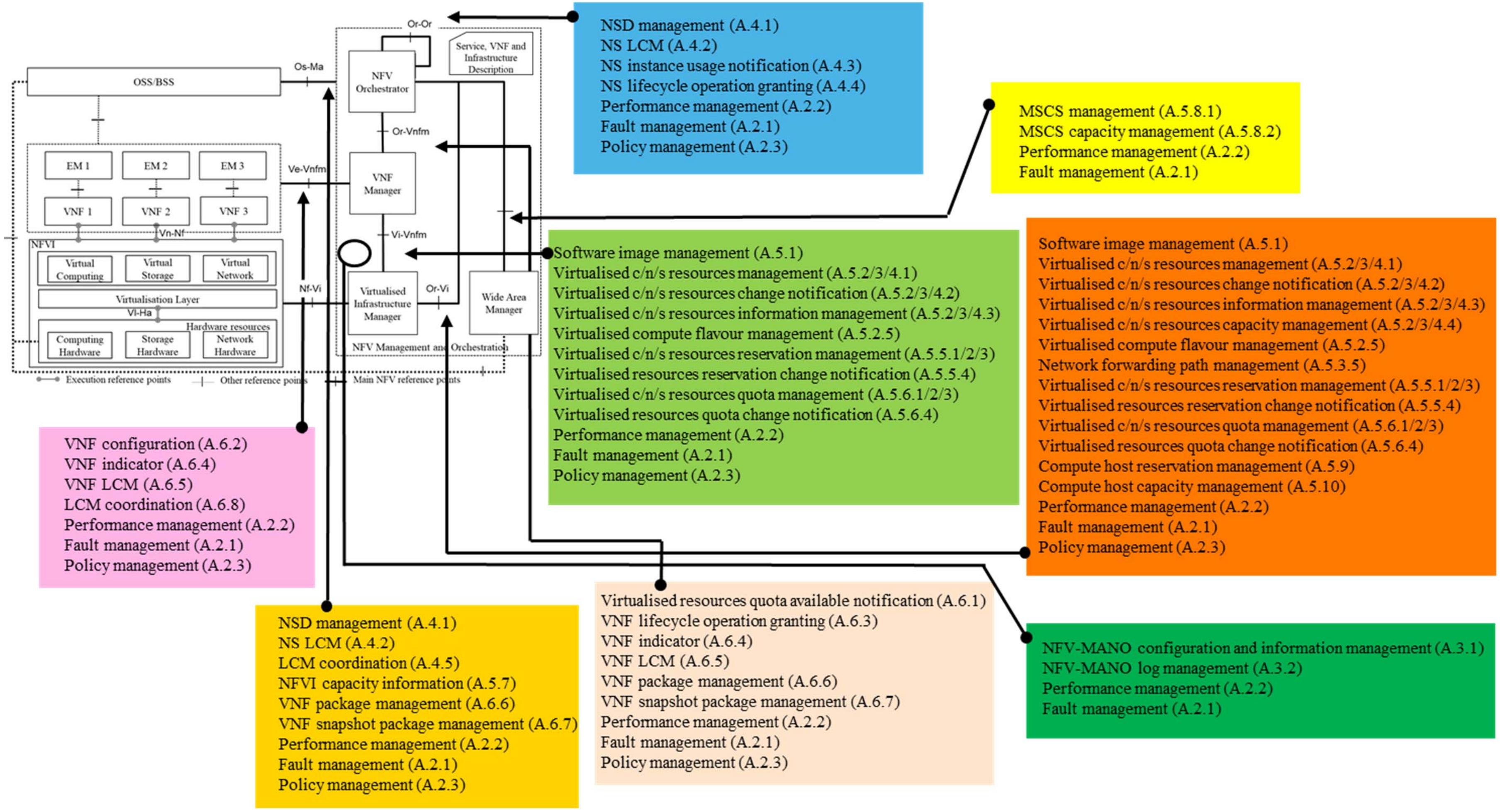
5.1 Nature of the data

The main input needed for cognitive approaches such as machine learning is data. Operations data of the NFV ecosystem arise from operations launched at the numerous NFV-MANO interfaces. Figure 5.1-1 shows all the interfaces through which operations data can be collected. In this figure, the interfaces are grouped, when it is possible, according to the reference points of the NFV architecture. The number following each interface refers to the clause number in Annex A of the present document elaborating on the interface. Two interfaces are common to all reference points: performance management and fault management. In addition, the policy management interface is also common with two exceptions currently. A number of interfaces exists in similar form for different resources. To improve readability, similar interfaces related to compute, network and storage are grouped through the symbol "c/n/s". Their references are also combined (e.g. A.5.2/3/4.1 meaning respectively A.5.2.1, A.5.3.1 and A.5.4.1).

The operations executed at these interfaces are of different nature:

- Lifecycle management: create, activate, associate, upload, fetch, instantiate, add, allocate, attach, build, transfer, extract/delete, stop, deactivate, disassociate, terminate, detach.
- Modification: modify, change, update, scale, migrate.
- Inquiry: query, get (alarm list, operation status, indicator value).
- Incidents: escalate severity, ACK alarms, heal, revert-to snapshot.
- Other LCM operations: grant (NS/VNF lifecycycle), coordinate, operate, set (configuration).
- Subscription: initialization, termination, information query.
- Reporting: notification.

All operations found at the different interfaces are listed in Annex B.



NSD management (A.4.1)
NS LCM (A.4.2)
NS instance usage notification (A.4.3)
NS lifecycle operation granting (A.4.4)
Performance management (A.2.2)
Fault management (A.2.1)
Policy management (A.2.3)

MSCS management (A.5.8.1)
MSCS capacity management (A.5.8.2)
Performance management (A.2.2)
Fault management (A.2.1)

Software image management (A.5.1)
Virtualised c/n/s resources management (A.5.2/3/4.1)
Virtualised c/n/s resources change notification (A.5.2/3/4.2)
Virtualised c/n/s resources information management (A.5.2/3/4.3)
Virtualised compute flavour management (A.5.2.5)
Virtualised c/n/s resources reservation management (A.5.5.1/2/3)
Virtualised resources reservation change notification (A.5.5.4)
Virtualised c/n/s resources quota management (A.5.6.1/2/3)
Virtualised resources quota change notification (A.5.6.4)
Performance management (A.2.2)
Fault management (A.2.1)
Policy management (A.2.3)

Software image management (A.5.1)
Virtualised c/n/s resources management (A.5.2/3/4.1)
Virtualised c/n/s resources change notification (A.5.2/3/4.2)
Virtualised c/n/s resources information management (A.5.2/3/4.3)
Virtualised c/n/s resources capacity management (A.5.2/3/4.4)
Virtualised compute flavour management (A.5.2.5)
Network forwarding path management (A.5.3.5)
Virtualised c/n/s resources reservation management (A.5.5.1/2/3)
Virtualised resources reservation change notification (A.5.5.4)
Virtualised c/n/s resources quota management (A.5.6.1/2/3)
Virtualised resources quota change notification (A.5.6.4)
Compute host reservation management (A.5.9)
Compute host capacity management (A.5.10)
Performance management (A.2.2)
Fault management (A.2.1)
Policy management (A.2.3)

VNF configuration (A.6.2)
VNF indicator (A.6.4)
VNF LCM (A.6.5)
LCM coordination (A.6.8)
Performance management (A.2.2)
Fault management (A.2.1)
Policy management (A.2.3)

NSD management (A.4.1)
NS LCM (A.4.2)
LCM coordination (A.4.5)
NFVI capacity information (A.5.7)
VNF package management (A.6.6)
VNF snapshot package management (A.6.7)
Performance management (A.2.2)
Fault management (A.2.1)
Policy management (A.2.3)

Virtualised resources quota available notification (A.6.1)
VNF lifecycle operation granting (A.6.3)
VNF indicator (A.6.4)
VNF LCM (A.6.5)
VNF package management (A.6.6)
VNF snapshot package management (A.6.7)
Performance management (A.2.2)
Fault management (A.2.1)
Policy management (A.2.3)

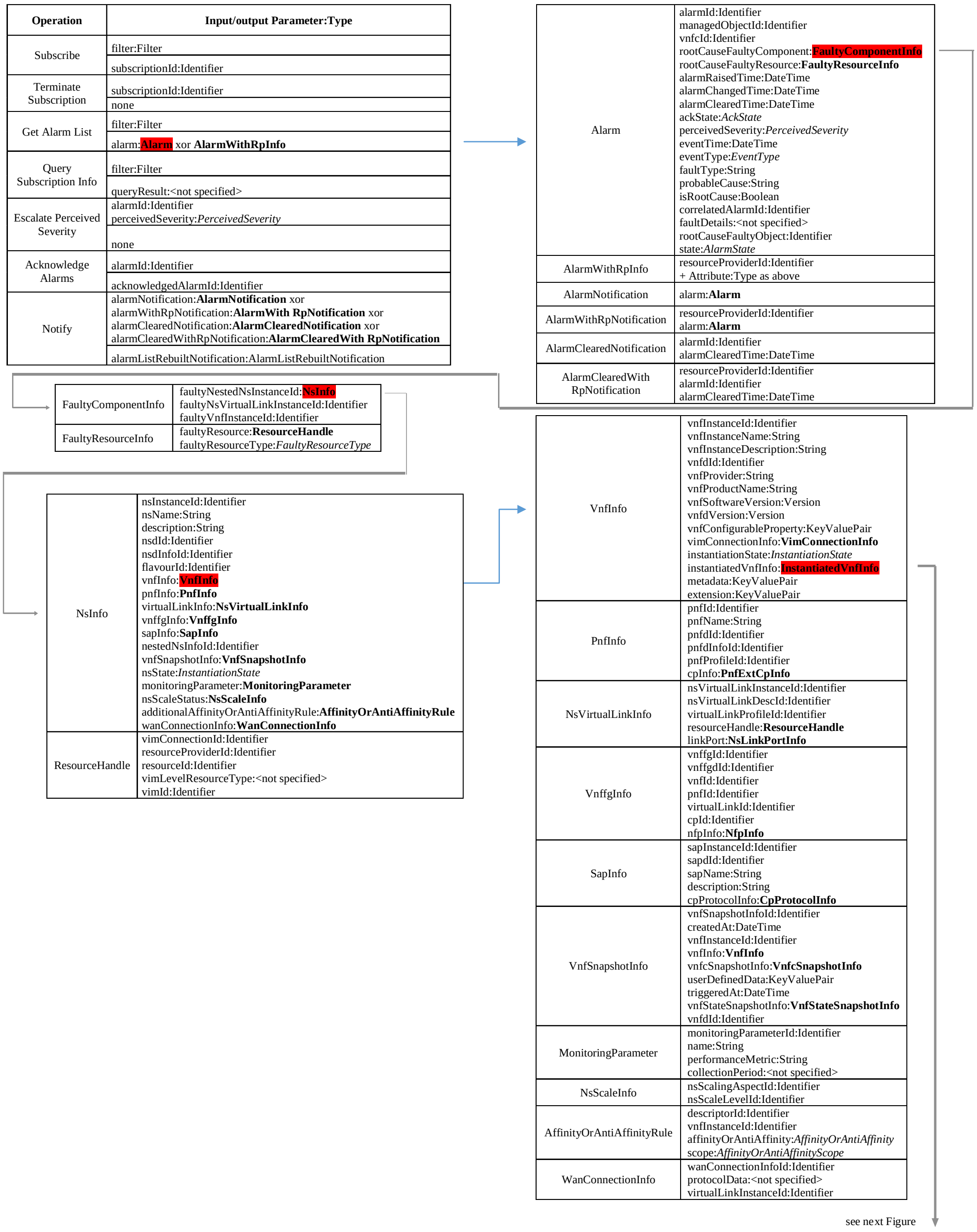
NFV-MANO configuration and information management (A.3.1)
NFV-MANO log management (A.3.2)
Performance management (A.2.2)
Fault management (A.2.1)

Figure 5.1-1: Interfaces of the NFV architecture

5.2 Data contents

Operations data arise from the operations shown in Annex B as the contents of their input/output parameters. These contents are composed of encapsulated information elements associated with attributes and types. Annex A shows these information. For illustration purpose, the encapsulation of fault management operations data is used below. Figure 5.2-1 lists the operations related to the fault management interface which is defined for all NFV reference points. The input/output parameters, followed by their type, are also provided for the operations. Figures 5.2-1 and 5.2-2 develop the encapsulation of the information elements in use, associated with their attributes and types. The notation applied in these figures is the following:

- 'xor' means that only one parameter:type or attribute:type can be used at a time from the list;
- regular typeset means a primitive/predefined type;
- bold typeset means a type/information element whose data structure is given in the present clause - for illustration purposes, the arrows show the encapsulation of the data structure for some particular types/information elements of an alarm;
- italic typeset means Enum whose values are listed in Table 5.2-1;
- bold+italic typeset means abstract type in place of which an appropriate specialization/child type can be used.



Operation	Input/output Parameter:Type
Subscribe	filter:Filter subscriptionId:Identifier
Terminate Subscription	subscriptionId:Identifier none
Get Alarm List	filter:Filter alarm:Alarm xor **AlarmWithRpInfo**
Query Subscription Info	filter:Filter queryResult:<not specified>
Escalate Perceived Severity	alarmId:Identifier perceivedSeverity:*PerceivedSeverity* none
Acknowledge Alarms	alarmId:Identifier acknowledgedAlarmId:Identifier
Notify	alarmNotification:**AlarmNotification** xor alarmWithRpNotification:**AlarmWith RpNotification** xor alarmClearedNotification:**AlarmClearedNotification** xor alarmClearedWithRpNotification:**AlarmClearedWith RpNotification** alarmListRebuiltNotification:AlarmListRebuiltNotification

Alarm	alarmId:Identifier managedObjectId:Identifier vnfcId:Identifier rootCauseFaultyComponent:**FaultyComponentInfo** rootCauseFaultyResource:**FaultyResourceInfo** alarmRaisedTime:DateTime alarmChangedTime:DateTime alarmClearedTime:DateTime ackState:*AckState* perceivedSeverity:*PerceivedSeverity* eventTime:DateTime eventType:*EventType* faultType:String probableCause:String isRootCause:Boolean correlatedAlarmId:Identifier faultDetails:<not specified> rootCauseFaultyObject:Identifier state:*AlarmState*
AlarmWithRpInfo	resourceProviderId:Identifier + Attribute:Type as above
AlarmNotification	alarm:**Alarm**
AlarmWithRpNotification	resourceProviderId:Identifier alarm:**Alarm**
AlarmClearedNotification	alarmId:Identifier alarmClearedTime:DateTime
AlarmClearedWith RpNotification	resourceProviderId:Identifier alarmId:Identifier alarmClearedTime:DateTime

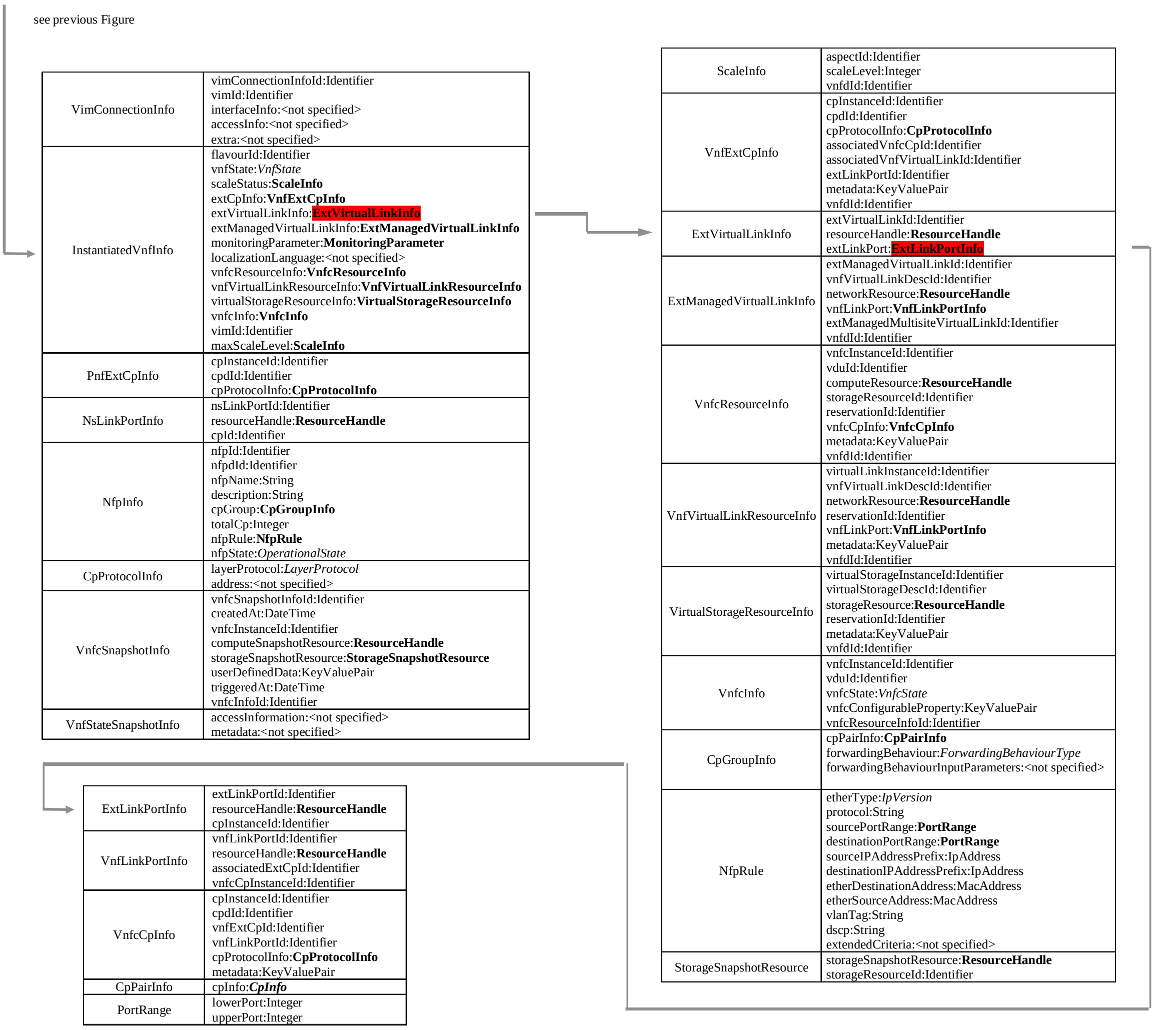
FaultyComponentInfo	faultyNestedNsInstanceId:NsInfo faultyNsVirtualLinkInstanceId:Identifier faultyVnfInstanceId:Identifier
FaultyResourceInfo	faultyResource:**ResourceHandle** faultyResourceType:*FaultyResourceType*

NsInfo	nsInstanceId:Identifier nsName:String description:String nsdId:Identifier nsdInfoId:Identifier flavourId:Identifier vnfInfo:VnfInfo pnfInfo:**PnfInfo** virtualLinkInfo:**NsVirtualLinkInfo** vnffgInfo:**VnffgInfo** sapInfo:**SapInfo** nestedNsInfoId:Identifier vnfSnapshotInfo:**VnfSnapshotInfo** nsState:*InstantiationState* monitoringParameter:**MonitoringParameter** nsScaleStatus:**NsScaleInfo** additionalAffinityOrAntiAffinityRule:**AffinityOrAntiAffinityRule** wanConnectionInfo:**WanConnectionInfo**
ResourceHandle	vimConnectionId:Identifier resourceProviderId:Identifier resourceId:Identifier vimLevelResourceType:<not specified> vimId:Identifier

VnfInfo	vnfInstanceId:Identifier vnfInstanceName:String vnfInstanceDescription:String vnfdId:Identifier vnfProvider:String vnfProductName:String vnfSoftwareVersion:Version vnfdVersion:Version vnfConfigurableProperty:KeyValuePair vimConnectionInfo:**VimConnectionInfo** instantiationState:*InstantiationState* instantiatedVnfInfo:InstantiatedVnfInfo metadata:KeyValuePair extension:KeyValuePair
PnfInfo	pnfId:Identifier pnfName:String pnfdId:Identifier pnfdInfoId:Identifier pnfProfileId:Identifier cpInfo:**PnfExtCpInfo**
NsVirtualLinkInfo	nsVirtualLinkInstanceId:Identifier nsVirtualLinkDescId:Identifier virtualLinkProfileId:Identifier resourceHandle:**ResourceHandle** linkPort:**NsLinkPortInfo**
VnffgInfo	vnffgId:Identifier vnffgdId:Identifier vnfId:Identifier pnfId:Identifier virtualLinkId:Identifier cpId:Identifier nfpInfo:**NfpInfo**
SapInfo	sapInstanceId:Identifier sapdId:Identifier sapName:String description:String cpProtocolInfo:**CpProtocolInfo**
VnfSnapshotInfo	vnfSnapshotInfoId:Identifier createdAt:DateTime vnfInstanceId:Identifier vnfInfo:**VnfInfo** vnfcSnapshotInfo:**VnfcSnapshotInfo** userDefinedData:KeyValuePair triggeredAt:DateTime vnfStateSnapshotInfo:**VnfStateSnapshotInfo** vnfdId:Identifier
MonitoringParameter	monitoringParameterId:Identifier name:String performanceMetric:String collectionPeriod:<not specified>
NsScaleInfo	nsScalingAspectId:Identifier nsScaleLevelId:Identifier
AffinityOrAntiAffinityRule	descriptorId:Identifier vnfInstanceId:Identifier affinityOrAntiAffinity:*AffinityOrAntiAffinity* scope:*AffinityOrAntiAffinityScope*
WanConnectionInfo	wanConnectionInfoId:Identifier protocolData:<not specified> virtualLinkInstanceId:Identifier

see next Figure

Figure 5.2-1: Operations data at the fault management interface



VimConnectionInfo	vimConnectionInfoId:Identifier vimId:Identifier interfaceInfo:<not specified> accessInfo:<not specified> extra:<not specified>
InstantiatedVnfInfo	flavourId:Identifier vnfState:*VnfState* scaleStatus:**ScaleInfo** extCpInfo:**VnfExtCpInfo** extVirtualLinkInfo:**ExtVirtualLinkInfo** extManagedVirtualLinkInfo:**ExtManagedVirtualLinkInfo** monitoringParameter:**MonitoringParameter** localizationLanguage:<not specified> vnfcResourceInfo:**VnfcResourceInfo** vnfVirtualLinkResourceInfo:**VnfVirtualLinkResourceInfo** virtualStorageResourceInfo:**VirtualStorageResourceInfo** vnfcInfo:**VnfcInfo** vimId:Identifier maxScaleLevel:**ScaleInfo**
PnfExtCpInfo	cpInstanceId:Identifier cpdId:Identifier cpProtocolInfo:**CpProtocolInfo**
NsLinkPortInfo	nsLinkPortId:Identifier resourceHandle:**ResourceHandle** cpId:Identifier
NfpInfo	nfpId:Identifier nfpdId:Identifier nfpName:String description:String cpGroup:**CpGroupInfo** totalCp:Integer nfpRule:**NfpRule** nfpState:*OperationalState*
CpProtocolInfo	layerProtocol:*LayerProtocol* address:<not specified>
VnfcSnapshotInfo	vnfcSnapshotInfoId:Identifier createdAt:DateTime vnfcInstanceId:Identifier computeSnapshotResource:**ResourceHandle** storageSnapshotResource:**StorageSnapshotResource** userDefinedData:KeyValuePair triggeredAt:DateTime vnfcInfoId:Identifier
VnfStateSnapshotInfo	accessInformation:<not specified> metadata:<not specified>

ScaleInfo	aspectId:Identifier scaleLevel:Integer vnfdId:Identifier
VnfExtCpInfo	cpInstanceId:Identifier cpdId:Identifier cpProtocolInfo:**CpProtocolInfo** associatedVnfcCpId:Identifier associatedVnfVirtualLinkId:Identifier extLinkPortId:Identifier metadata:KeyValuePair vnfdId:Identifier
ExtVirtualLinkInfo	extVirtualLinkId:Identifier resourceHandle:**ResourceHandle** extLinkPort:**ExtLinkPortInfo**
ExtManagedVirtualLinkInfo	extManagedVirtualLinkId:Identifier vnfVirtualLinkDescId:Identifier networkResource:**ResourceHandle** vnfLinkPort:**VnfLinkPortInfo** extManagedMultisiteVirtualLinkId:Identifier vnfdId:Identifier
VnfcResourceInfo	vnfcInstanceId:Identifier vduId:Identifier computeResource:**ResourceHandle** storageResourceId:Identifier reservationId:Identifier vnfcCpInfo:**VnfcCpInfo** metadata:KeyValuePair vnfdId:Identifier
VnfVirtualLinkResourceInfo	virtualLinkInstanceId:Identifier vnfVirtualLinkDescId:Identifier networkResource:**ResourceHandle** reservationId:Identifier vnfLinkPort:**VnfLinkPortInfo** metadata:KeyValuePair vnfdId:Identifier
VirtualStorageResourceInfo	virtualStorageInstanceId:Identifier virtualStorageDescId:Identifier storageResource:**ResourceHandle** reservationId:Identifier metadata:KeyValuePair vnfdId:Identifier
VnfcInfo	vnfcInstanceId:Identifier vduId:Identifier vnfcState:*VnfcState* vnfcConfigurableProperty:KeyValuePair vnfcResourceInfoId:Identifier
CpGroupInfo	cpPairInfo:**CpPairInfo** forwardingBehaviour:*ForwardingBehaviourType* forwardingBehaviourInputParameters:<not specified>
NfpRule	etherType:*IpVersion* protocol:String sourcePortRange:**PortRange** destinationPortRange:**PortRange** sourceIPAddressPrefix:IpAddress destinationIPAddressPrefix:IpAddress etherDestinationAddress:MacAddress etherSourceAddress:MacAddress vlanTag:String dscp:String extendedCriteria:<not specified>
StorageSnapshotResource	storageSnapshotResource:**ResourceHandle** storageResourceId:Identifier

ExtLinkPortInfo	extLinkPortId:Identifier resourceHandle:**ResourceHandle** cpInstanceId:Identifier
VnfLinkPortInfo	vnfLinkPortId:Identifier resourceHandle:**ResourceHandle** associatedExtCpId:Identifier vnfcCpInstanceId:Identifier
VnfcCpInfo	cpInstanceId:Identifier cpdId:Identifier vnfExtCpId:Identifier vnfLinkPortId:Identifier cpProtocolInfo:**CpProtocolInfo** metadata:KeyValuePair
CpPairInfo	cpInfo:***CpInfo***
PortRange	lowerPort:Integer upperPort:Integer

Figure 5.2-2: Operations data at the fault management interface (cont.)

Table 5.2-1 shows the Enum values for use in Figures 5.2-1 and 5.2-2. If applicable, the column "Expandable" indicates that additional values can be defined for the related Enum.

Table 5.2-1: Enum values

Enum	Values	Expandable
AckState	ACKNOWLEDGED, UNACKNOWLEDGED	
AffinityOrAntiAffinity	AFFINITY, ANTI_AFFINITY	
AffinityOrAntiAffinityScope	NFVI_NODE, NFVI-PoP, Zone, ZoneGroup, NFVI-node, network-link-and-node, container-namespace, NIC, VIRTUAL_SWITCH_OR_ROUTER, PHYSICAL_NIC, PHYSICAL_NETWORK	X
AlarmState	FIRED, UPDATED, CLEARED	
EventType	COMMUNICATION_ALARM, PROCESSING_ALARM, ENVIRONMENT_ALARM, QOS_ALARM, EQUIPMENT_ALARM	
FaultyResourceType	COMPUTE, STORAGE, NETWORK	
ForwardingBehaviourType	ALL, LB	X
InstantiationState	NOT_INSTANTIATED, INSTANTIATED	
IpVersion	IPV4, IPV6	
LayerProtocol	Ethernet, MPLS, ODU2, IPV4, IPV6, Pseudo-Wire	X
OperationalState	ENABLED, DISABLED	
PerceivedSeverity	CRITICAL, MAJOR, MINOR, WARNING, INDETERMINATE, CLEARED	
VnfcState	STARTED, STOPPED	
VnfState	STARTED, STOPPED	

6 Cognitive analysis of operations data

6.1 Data driven techniques

6.1.1 Cognitive computing

Data analytics has begun with exercises in the description and diagnostics of systems. Over time, it has become more sophisticated tackling predictions and the proposal of corrective or preventative actions. This shift was enabled by advances in data modeling techniques resulting in more accurate future forecasts and actionable intelligence. Cognitive computing, a subfield of artificial intelligence, simulates the human thought process using self-learning algorithms through data mining, pattern recognition, and natural language processing.

These may rely on different machine learning techniques, such as deep learning algorithms and neural networks that process information by comparing them to some model constructed from a data set provided as reference. The result of such comparison can be then used by a function of the system for different purposes, including operations control and management, thus, allowing to build zero-touch control loops and paving the way to autonomous networking.

The reference data set can be obtained in different ways: it can be a synthetic set of data generated by a simulation model of the targeted system, it can be collected from the operations of a substantially similar system (e.g. a pre-deployment staging of the system), or from the previous or current operations of the targeted system. The machine learning techniques used are traditionally divided into three broad categories.

Depending on the "feedback" available to the learning system these are supervised, unsupervised and reinforcement learning, which are next described briefly. More details can be found in [i.18], [i.19], [i.20], [i.21], [i.22] and [i.23], which in turn also provide further references.

6.1.2 Supervised learning

6.1.2.1 Introduction

Supervised learning uses example inputs and their desired outputs - the training data set - to learn a general rule that maps inputs to outputs. The supervision is achieved with labels applied to the training data set that guide the learning process.

6.1.2.2 Techniques used for supervised learning

Various algorithms and computation techniques have been used in supervised machine learning:

- *Artificial Neural Networks* are primarily leveraged for deep learning algorithms. They mimic the interconnectivity of the human brain through layers of nodes. Each node is made up of inputs, weights, a bias (or threshold), and an output. If the output value calculated from the weighted inputs exceeds a given threshold, it activates the node, passing data to the next layer in the network.

- *Naive Bayes* is a classification approach that adopts the principle of class conditional independence from the Bayes Theorem. That is, the presence of one feature does not impact the presence of another in the probability of a given outcome, and each predictor has an equal effect on that result.

- *Linear Regression* is used to identify the relationship between a dependent variable and one or more independent variables. It is typically used to make predictions about future outcomes. It seeks to plot a line of best fit, which is calculated through the method of least squares.

- *Logistic Regression* is used when the dependent variable is categorical, meaning it has binary outputs (e.g. true or false). Therefore, logistic regression is suited to solve binary classification problems.

- *Support Vector Machine* is preferred for data classification by constructing a hyperplane where the distance between two classes of data points is at its maximum. This hyperplane is known as the decision boundary, separating the classes of data points (e.g. events of normal operations vs. events of anomalous operations) on either side of the plane.

- *K-Nearest Neighbour* is a non-parametric algorithm that classifies data points based on their proximity and association to other available data. It makes the assumption that similar data points can be found near each other. Hence, it calculates the distance between data points, usually through Euclidean distance, and assigns a category based on the most frequent category or average.

- *Random Forest* is used for both classification and regression purposes. The "forest" in the naming references a collection of uncorrelated decision trees, which are then merged together to reduce variance and create more accurate data predictions.

6.1.2.3 Challenges

Although supervised learning offers several advantages such as deep data insights, improved automation, there are some challenges that need to be overcome in building supervised learning models. Some of these challenges are:

- Supervised learning cannot cluster or classify data on its own.

- Supervised learning models require accurate structuring and labelling of the training data, which requires field expertise.

- Due to this human input, there is a higher likelihood of human errors or bias resulting, potentially, in incorrect learning.

- The training process of supervised learning models can be very time intensive.

6.1.3 Unsupervised learning

6.1.3.1 Introduction

Unsupervised learning implies that no guidance (in the form of labels) is given to the learning algorithm, leaving it to its own devices to find a structure in the provided input. This task (finding a structure) can be a goal in itself, or it can be a means towards an end such as "feature learning", which can replace "feature engineering" (i.e. identifying characteristics, properties, attributes in general referred to as "features") often performed by a human expert for supervised learning. Unsupervised learning models are utilized for three main tasks: clustering, association, and dimensionality reduction.

6.1.3.2 Clustering

Clustering is a data mining technique which groups unlabelled (i.e. raw) typically large amount of data based on their similarities or differences into groups represented by structures or patterns in the information. Different techniques exist to measure similarity based on different methods of distance calculation, e.g. Euclidean distance.

The categories of clustering algorithms are as follows:

- Exclusive clustering is a form of grouping that stipulates a data point can exist only in one cluster. The K-means clustering algorithm is an example of this method where data points are assigned into K groups, where K is an input and represents the number of clusters to be formed based on the distance from each group's centroid.
- Overlapping clustering differs from exclusive clustering in that it allows data points to belong to multiple clusters with separate degrees of membership. "Soft" or fuzzy k-means clustering is an example of overlapping clustering.
- Hierarchical clustering (or hierarchical cluster analysis) algorithms can be agglomerative or divisive:
    - Agglomerative clustering is a "bottoms-up approach", which starts from the isolated data points as groupings which then are merged iteratively into higher-level groupings based on their similarity until a single top-level cluster is formed.
    - Divisive clustering is used less frequently, and it takes a "top-down" approach. That is, all data points are considered as a single cluster, which then is divided based on the differences between data points.
- Probabilistic clustering groups data points based on the likelihood that they belong to a particular distribution. The most used probabilistic clustering method is the gaussian mixture model.

6.1.3.3 Association

Association rule learning is an unsupervised learning method for discovering relationships between data points in large datasets. It is one of the rule-based machine learning approaches that identify, learn, and evolve "rules" to store, manipulate, and/or apply knowledge. The identified set of relational rules collectively represents the knowledge captured by the system.

Association rule learning is frequently used for market basket analysis, allowing companies to better understand relationships between different products. There are a few different algorithms used to generate association rules, such as Apriori, Eclat, and FP-Growth, the Apriori algorithm being the most widely used currently.

6.1.3.4 Dimensionality reduction

While more data generally yields more accurate results, it can impact the performance of machine learning algorithms and make it difficult to visualize datasets. Dimensionality reduction is used to reduce the number of features (i.e. dimensions or input) when their number is too high in a dataset. The goal is to have a manageable size input while preserving the integrity of the dataset as much as possible. It is commonly used as part of the data pre-processing stage.

Methods that can be used for dimensionality reduction are:

- Principal component analysis is used to reduce redundancies and to compress datasets through feature extraction. Linear transformation is used to create a new data representation, yielding a set of "principal components." The first principal component is the direction which maximizes the variance of the dataset. Each subsequent principal component also finds the maximum variance in the data, which is completely uncorrelated to the previous principal component(s), thus, yielding a direction that is orthogonal to that/those component(s).

- Singular value decomposition factorizes a matrix, A, into three, low-rank matrices. It is denoted by the formula, $A = UDV^T$, where U and V are orthogonal matrices, while the matrix D is diagonal with positive real entries. (Note that T denotes the transpose of the matrix.)

- Autoencoders leverage neural networks to compress data and then recreate a new representation of the original data's input. The stage from the input layer to the hidden layer is referred to as "encoding" while the stage from the hidden layer to the output layer is known as "decoding."

6.1.3.5 Challenges

Unsupervised learning also comes with some challenges that need to be considered. These challenges can include:

- Computational complexity due to the high volume of training data needed to produce the intended output resulting in longer training times.

- Higher risk of inaccurate results (due to the lack of supervision) combined with the lack of transparency into the basis on which those results were generated (e.g. data was clustered).

- Need for human intervention to validate the output.

6.1.4 Reinforcement learning

6.1.4.1 Introduction

Reinforcement learning [i.24] is about learning interactively how to behave in order to achieve some goal. Thus, a learning agent is introduced, which interacts with its environment over a sequence of discrete time steps. In this process, the agent observes the state of its environment and takes actions that affect the state of the environment.

Reinforcement learning always encompasses three elements: a policy, a reward signal and a value function. In addition, it may also include a model of the environment. The *policy* (sometimes called action selection) defines the behaviour of the learning agent. That is, it determines the action the agent takes in each state of the environment. The policy may be a lookup table, a simple function (including a random function), or may involve extensive computations.

The *reward signal* is generated by the environment towards the agent at each time step and indicates the perceived immediate quality of the action taken by the agent. The agent accumulates the rewards, and its sole objective is to maximize its total rewards. Thus, the reward can be used to reinforce or adjust the policy. As opposed to the reward signal, the *value function* specifies what is good in the long run; therefore, it reflects the goal of a reinforcement learning problem. The value of a state is the total amount of rewards an agent can expect to accumulate over future actions, starting from that state. Accordingly, a state might yield a low reward but still have a high value because it is followed by states that yield high rewards.

Reinforcement learning may use a model of the environment allowing for planning, that is, deciding on a course of action by considering possible future situations before they are actually experienced. A reinforcement learning agent may use a combination of model-based and model-free methods.

A key difference between reinforcement learning and other learning methods is that the agent can operate in exploration or exploitation mode. In exploration mode, it learns by trial and error the uncharted territory of the environment through its actions and the received rewards. In the exploitation mode, the agent applies the learnt knowledge/the model to navigate the environment.

Different reinforcement learning methods have been developed exploring different aspects of these elements, e.g. model-based vs model-free, on-policy vs off-policy methods. Some of the most popular methods are temporal difference learning methods and, in particular, SARSA and Q-learning. Reinforcement learning approaches were also extended to multiple agents.

The two main areas to which reinforcement learning has been applied successfully are prediction and control problems (e.g. prediction of total driving time, mobile robot control, hypothesis of dopamine neuron activity, Go game playing, autonomous gliders, etc.).

6.1.4.2 Temporal difference learning

Temporal difference learning is a class of model-free reinforcement learning methods which learn by bootstrapping the current estimate of the value function. Meaning that predictions can be learned from observations of the environment, and they can be adjusted as more observations become available before the final outcome is known.

In temporal difference learning, the value function is called *Q*-value function and is based on the Bellman equations. At each time step, the agent updates the *Q*-value using the weighted average of the old value and the new information. The *Q*-value function incorporates two factors that tailor the operation:

- *Learning rate*, which defines to what extent a new Q-value will override the old one. 0 means no update, i.e. the agent will not learn anything, while 1 means that newly discovered information completely overrides the old.
- *Discount factor*, which defines the importance of future rewards. 0 means that only short-term rewards are considered, while 1 puts more importance on long-term rewards.

In case of SARSA (State-Action-Reward-State-Action), when the Q-value is updated for a state-action pair, the next state and action have already been selected based on the same policy (i.e. on-policy) simplifying the temporal difference learning algorithm significantly.

Q-learning is an off-policy temporal difference learning because it takes the maximum of values of future state action-pairs rather than considering only the state-action pair that would be selected by the policy. It is also model-free, hence, there needs to be a balance between exploration and exploitation.

6.1.4.3 Multi-agent reinforcement learning

In multi-agent reinforcement learning, the learning is performed by a population of reinforcement learning agents. In this context, different scenarios can be considered based on whether there are conflicts of interest among the agents.

In a cooperative scenario, all agents try to maximize a common reward signal that they all receive simultaneously. This is a team problem and there are no conflicts of interest among the agents.

The scenario becomes a competitive problem if different agents receive different reward signals. In this case, deciding what the best collective action should be is a non-trivial aspect of game theory.

6.1.4.4 Challenges

Reinforcement learning also comes with challenges, some of which are:

- Trade-off between exploration and exploitation: to obtain a lot of rewards, a reinforcement learning agent needs to prefer actions that it has tried in the past and found to be effective in producing rewards.
- Design of the reward signal so that the agent learns and eventually achieves, what the application's designer actually desires, i.e. the goals of the agent and the designer are distinct. Reinforcement learning agents can discover unexpected ways to make their environments deliver rewards, some of which might be undesirable, or even dangerous.
- Design of representing and storing the value functions and/or policies. I.e. a large or even infinite state set does not allow exhaustive representation. A successful application of reinforcement learning requires human knowledge and intuition about the specific problem.
- Off-policy learning is relatively new and unsettled, that is, the best way achieving it is still a mystery.

The key feature of reinforcement learning is considering the problem of a goal-directed agent interacting with an uncertain environment as a whole. Therefore, these challenges need to be addressed in combination. For example, in temporal difference learning it is a further challenge to combine a powerful value function approximation, off-policy learning, and the efficiency and flexibility of bootstrapping without introducing the potential for instability.

6.2 Application to the NFV environment for reliability

As presented in clause 5, a multitude of information is available and can be collected on the different interfaces in an NFV system. This gives the opportunity to apply data driven techniques discussed in clause 6.1 to different problems related to NFV systems. These include those related to reliability and availability - the focus of the present document.

From the perspective of reliability and availability, an NFV system needs to provide any deployed Network Service (NS) in accordance with the requested non-functional requirements such as the intended NS availability. The fulfilment of the NS level requirements depends on the fulfilment of derivate requirements applicable to the elements composing the NS and the supporting resources. The decomposition of higher-level requirements into lower-level requirements, and vice versa, the aggregation of lower-level fulfilments into higher level fulfilments might not be a straightforward task in one or the other direction, for example, while aggregating the lower-level delays into a total higher-level delay is a simple summation, the decomposition of a delay requirement can be performed in many different ways and would benefit from learnt knowledge. Even more so in case of availability, for which neither the decomposition nor the composition might be straightforward in an NFV system considering the complexity and dynamism of the system, and the difficulty of mapping instantaneous measurements to long-period characteristics.

For example, the requested availability of an NS can be fulfilled only if the VNFs and VLs composing the NS fulfil their respective availability requirements as derived from the NS availability requirement. In turn, the VNF and VL availability requirements put availability and reliability requirements on the supporting resources. The fulfilment of these requirements needs to be monitored potentially at all levels and adjustments need to be made whenever deviations are detected.

The first use case (see clause 7.1) presents how data driven techniques can be utilized for this purpose. Namely, how the analytical models used to configure NSs to fulfil their availability requirements efficiently can be used to generate training and validation data sets to create artificial neural network models using a supervised machine learning technique. These models then can be utilized to compare the data collected from the system monitoring the fulfilment of the requirements with those expected by the models to guarantee efficient fulfilment. Whenever there is a deviation of the monitored parameters, the models can also be used to identify the configuration adjustments needed to achieve the target requirements with optimal resource usage. Thus, artificial neural network models trained through supervised learning can serve as a reference, and guide the system in correct and efficient operation under different circumstances to fulfil the NS requirements.

The deviations detected in the system can have different causes. For example, if there were no failures detected for a significant amount of time then the actual failure rate of some or all resources of an NS might be lower than considered previously. In this case, the model(s) is used to determine if it is possible to release some of the resources to improve efficiency. In the opposite case, however, when the actual failure rate is higher than anticipated, it is essential not only to adjust the configuration to the current situation as suggested by the model(s), but to understand where and why the problems (errors and failures) occur in the system.

A failure at one level can result in multiple errors and/or failures at another level. Each error/failure may trigger their appropriate signalling mechanisms, and possibly local remediations (e.g. restart of a failed entity or failing over its services to a spare/standby if available). Such a cascade of events in a complex heterogeneous system makes it non-trivial to pinpoint the root cause(s) of a problem. Finding the root cause of a problem is essential for fixing it, and any delay may result in, e.g. severe penalties. In such cases, the data received through the fault management system need to be analysed and correlated to find the root cause.

However, fault management data (e.g. data collected at the time of a failure) by itself might be insufficient to resolve all issues, availability of additional data could be beneficial to determine the whole context of the problem. Thus, continuous monitoring and data collection is necessary from all potentially relevant data sources, which, considering the amount, can only be processed using data driven techniques. In such cases, even the identification of relevant data sources may not be trivial. Therefore, a possible approach is to exploit unsupervised learning techniques capable of discovering hidden patterns in the collected data, thus, in the system behaviour. The second use case presented in clause 7.2 elaborates on such an approach.

Namely, clause 7.2 presents a use case of fault detection and localization technique achieved by combining unsupervised and supervised machine learning techniques. For this purpose, a 2-layered Self-Organizing Map (SOM) is introduced akin to artificial neural networks. The first layer of this SOM is trained using unclassified operational data (e.g. resource performance metrics) according to unsupervised clustering techniques to determine behavioural patterns of resources. The second layer of the SOM is trained with service level operational data labelled according to the fulfilment status of service level target(s). That is, a supervised machine learning technique is used. The combination of these SOM layers allows the projection of service level qualification of data to the behavioural patterns of resources, which in turn allows the qualification of resource level operational data at runtime into healthy and unhealthy behaviour. If a sample collected at runtime reflects unhealthy behaviour, a K-nearest neighbour technique is used to identify the fault type and location most likely causing this anomaly at its root.

Ideally, however, the monitoring and collected information could help to prevent problems even before the need for root cause analysis arises. Hence the idea and possibilities of using operations data to predict anomalies are presented in the third use case, in clause 7.3. As it was mentioned already, models trained through supervised learning can provide a reference for the correct behaviour of the system. Thus, detecting deviations from this can indicate that there is good chance that a problem is about to manifest in the system. At the simplest case, this prediction is just an indication that such a possibility exists without actually identifying or diagnosing the problem itself. Clause 7.3 discusses a number of techniques ranging from linear regression to random forest that can be utilized for this purpose.

Identifying/diagnosing the problem requires additional capabilities of pattern recognition and classification that are characteristic to unsupervised learning, as discussed with respect to the 2-layered SOM proposed for root cause analysis.

Alternatively, reinforcement learning offers the possibility of combining different machine learning techniques. There have been attempts to use reinforcement learning in the context of telecom. For example, it is possible to orchestrate the scaling of VNFs and NSs by an agent that uses reinforcement learning. However, this approach did not show a significant improvement over analytical/statistical methods while it required significantly more system resources. With respect to fault detection, diagnostics, and predictions, another issue that one needs to face is that the state space of faulty behaviours is infinite and exploring it might not be feasible upfront.

This could be a daunting task even for correct behaviour. Thus, the learning agent needs to include some arbitration mechanism which can qualify any new behaviour deviating from the currently known patterns, what category it belongs before it can use it to update its model. Solutions addressing these and other challenges of reinforcement learning require further investigations.

7 Use cases

7.1 Service Availability Assurance

7.1.1 Introduction

An NFV system needs to provide a Network Service (NS) in accordance to its requested service availability requirement. This is a complex task, which starts with the design of the NS and continues throughout its lifecycle. The present Service Availability Assurance use case demonstrates how machine learning models can support this task by encapsulating at design time the information necessary for managing the NS at runtime according to the service availability requirement. That is, machine learning models allow fast processing of operational data reflecting the runtime conditions to produce configuration changes if needed for the NS to meet the requirements.

To achieve this task, it needs to be ensured that all the composing elements of the NS, i.e. the VNFs and VLs, meet certain service availability requirements derived from the service availability requirement of the NS. Furthermore, the composing VLs are supported by virtualised network resources, which in turn need to meet the service availability requirements of the VLs they are supporting respectively. In case of the composing VNFs, each VNF itself is a composite entity, which is supported by a set of virtualised compute, network and storage resources. Thus, these virtualised resources need to meet the applicable service availability requirements derived from the service availability requirement of the VNF.

As described in ETSI GR NFV-REL 010 [i.10], the availability of an NS is calculated based on the availability of the composing VNFs and VLs, which in turn is calculated using the availability of the supporting resources. These calculations take into account the redundancy used at each VNF and NS levels. That is, within an NS a VNF can be deployed redundantly, similarly within a VNF a VNFC can be deployed redundantly, and at both levels VLs can be deployed redundantly. However, the NFV system, and in particularly the NFV-MANO functional entities are not aware of these redundancies as they are handled at the VNF application level. In other words, the NFV-MANO functional entities cannot distinguish between the use of resources for handling the workload from their use to increase service availability. This means, that to calculate the service availability of an NS and its composing VNFs, this application level component needs to be considered together with the service availability of the virtualised infrastructure.

The NFV-MANO functional entities are only aware and can manage the service availability of the virtualised resources of the NFVI, each of which can be composed of a set of physical resources with certain reliability characteristics, which might also utilize different redundancy schemas (e.g. RAID for storage, redundant physical links, etc.). The NFV-MANO functional entities can monitor these reliability characteristics over time and based on them predict/estimate the service availability of the virtualised resources hosted on the physical resources. In turn these estimates can be used at the VNF and NS levels to estimate the service availability of the respective composite entity to determine if the service availability requirements are met or actions need to be taken to meet them.

Over time the reliability characteristics of physical resources might change, for example, they can age, or fail after which they can be replaced with resources of different characteristics, or resources can be reallocated to accommodate workloads of higher priority. These changes also need to trigger the re-evaluation of the service availability that can be delivered at the different levels (i.e. NFVI, VNF and NS levels) and can trigger further actions at the different levels as the changes percolate up to the NS level.

As mentioned with respect to the physical resources, redundancy is another factor impacting service availability. With the changes in the workload the number of redundant instances at all levels changes in order to add/remove service capacity, and to maintain the appropriate level of protection for the service. However, the link between the service capacity and service protection is not straightforward. For one redundancy model, a standby instance might be necessary for each active instance. In another redundancy model, a single spare instance might be enough to protect any number of active instances. For the NFV-MANO functional entities, the redundancy used for either purpose is reflected in the scaling levels, but potentially of different deployment flavours. That is, each deployment flavour specifies the scaling of the deployment according to the capacity and the associated redundancy model. The selection of the deployment flavour and, therefore, the redundancy model is influenced by the reliability/availability of the underlying resources and the time required for the recovery from a failure at the given level. Thus, changes in the reliability/availability of the underlying resources might require changing deployment flavours, unless these changes can be compensated by improving the recovery time, for example, by adjusting related configuration parameters such as heartbeat and/or checkpointing rates, etc.

In any case, meeting and maintaining a given NS service availability is a complex task which can greatly benefit from cognitive techniques based on, or utilizing, operational data. For example, an NS design method used to determine the different deployment flavours with their scaling levels, which is based on a composite model (CoM) of different analytical models, can be used to generate training and validation data for deep neural network models for the NS, which in turn can replace some of the analytical models to be used to determine at runtime the required adjustments to the deployed NS instance and its composing elements to compensate for the changes in the availability characteristics of the supporting resources monitored as shown in Figure 7.1.1-1.

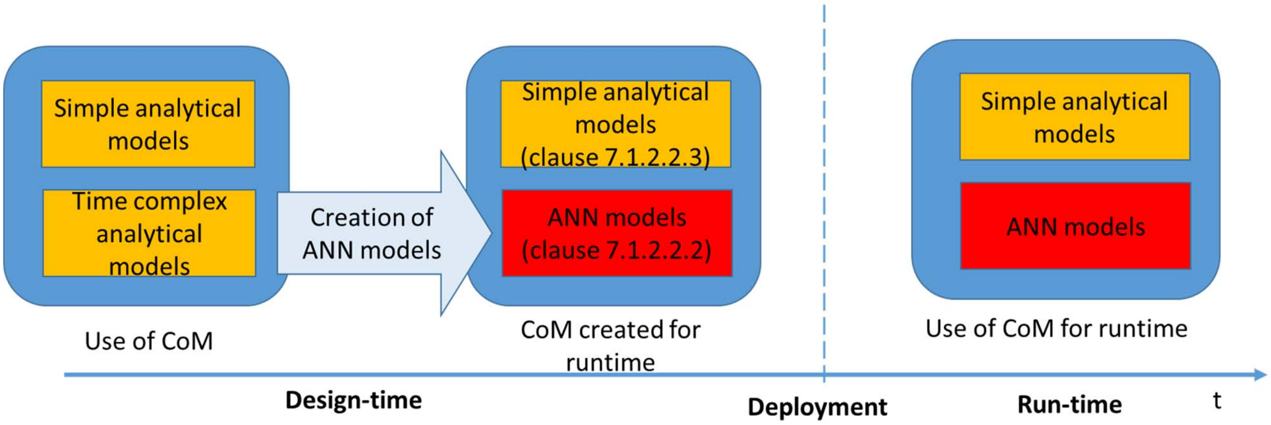


Figure 7.1.1-1: Process of replacing analytical models with artificial neural network models for runtime use

The rest of this clause will explore various concepts and different use case scenarios in this context.

7.1.2 Design-time model creation

7.1.2.1 Design-time process of creating ANN models

To design an NS so that it meets a certain availability requirement, analytical models similar to those discussed in ETSI GR NFV-REL 010 [i.10] can be used. That is, the analytical models for VNFs and VLs would be combined according to the composition of the NS resulting in a CoM. For this purpose, information is needed about the different possible deployment and configuration options of the infrastructure and the VNF implementations to determine the availability the NS can provide considering the different deployment and configuration options. This CoM can be used to determine the deployment and configuration parameters that meet best the availability requirement of the NS. This process is shown in Figure 7.1.2.1-1.

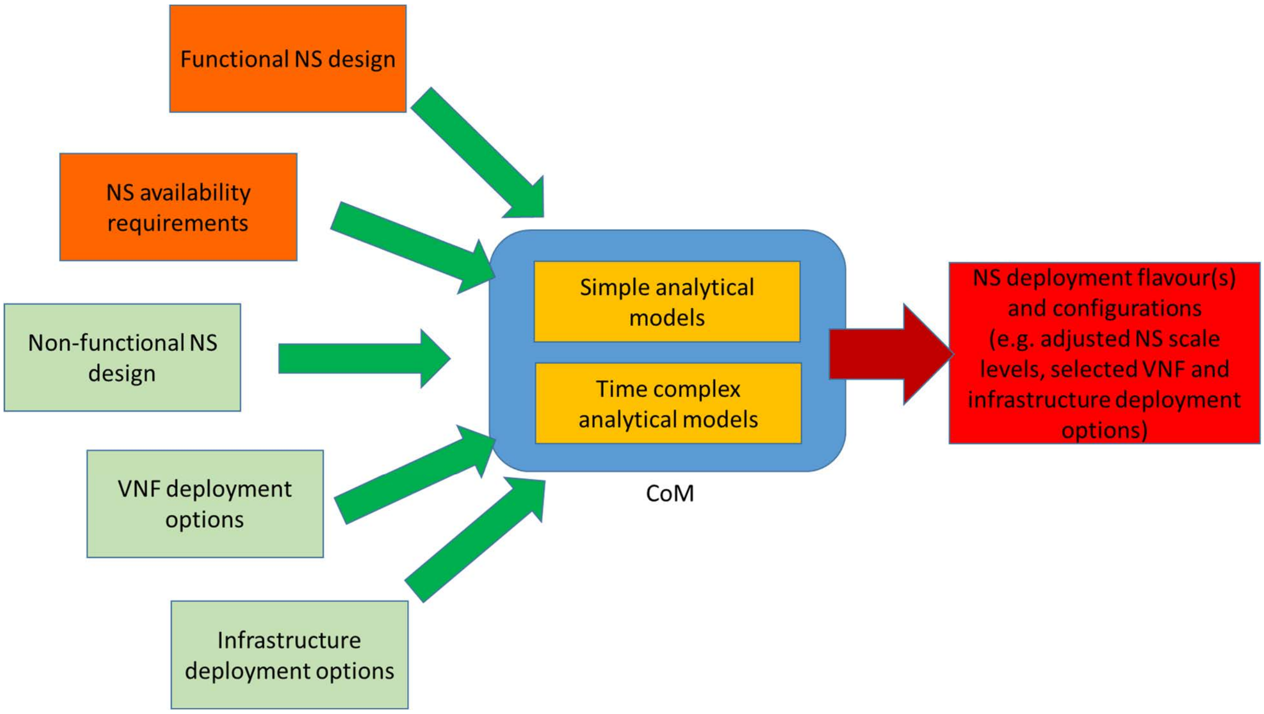

Functional NS design
NS availability requirements
Non-functional NS design
VNF deployment options
Infrastructure deployment options
Simple analytical models
Time complex analytical models
CoM
NS deployment flavour(s) and configurations (e.g. adjusted NS scale levels, selected VNF and infrastructure deployment options)

Figure 7.1.2.1-1: Using a composite model to design an NS flavour to meet certain requirements

The CoM constructed for the NS could also be used at runtime to recalculate the NS availability considering the actual values for the deployment if any changes occur. These recalculations result in any necessary adjustment to parameters configurable at runtime to maintain the requested availability. However, the CoM might be complex requiring significant computational power and time for any recalculation. In such cases, Artificial Neural Network (ANN) - such as feed-forward deep neural network - models can be used to approximate the CoM or parts of it. To do so, the CoM can be used to generate data to train and validate the ANN models as shown in Figure 7.1.2.1-2. That is, the CoM can be used to generate different solutions for potential changes in its input parameters. Then these sets of input and generated output parameters - so called labels - can be used to train and validate one or more ANN model(s) for the NS to imitate the recalculation at runtime.

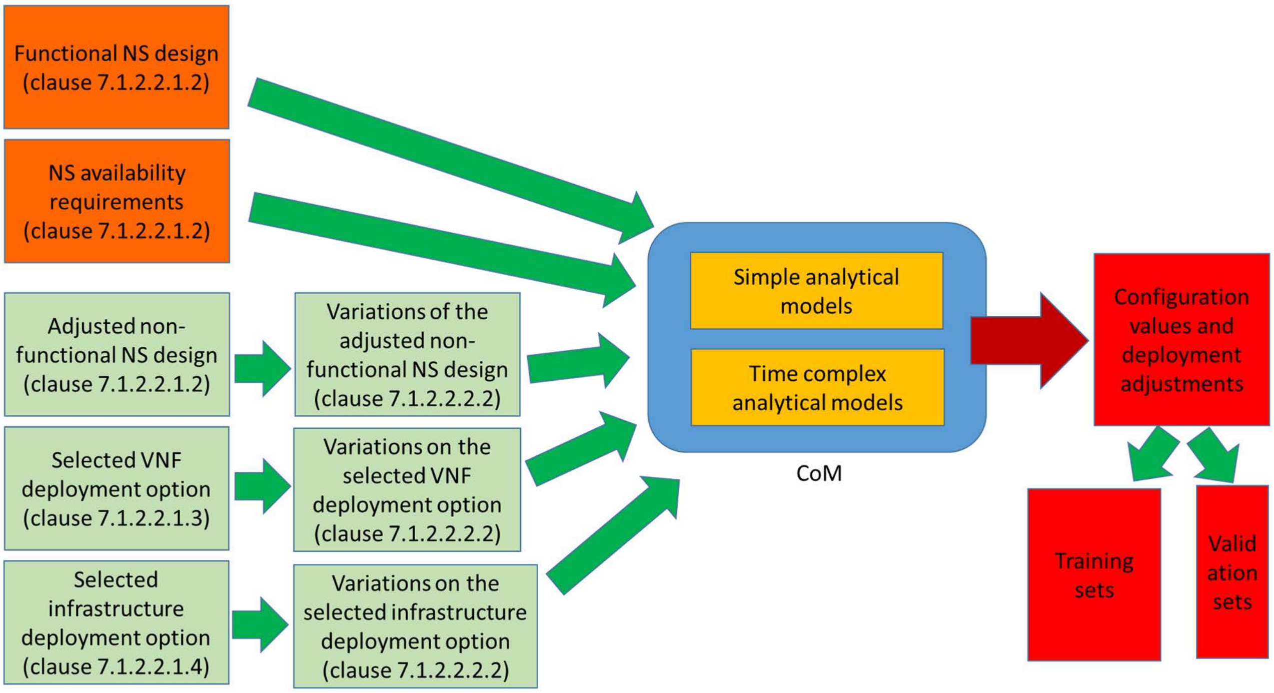

Functional NS design (clause 7.1.2.2.1.2)
NS availability requirements (clause 7.1.2.2.1.2)
Adjusted non-functional NS design (clause 7.1.2.2.1.2)
Selected VNF deployment option (clause 7.1.2.2.1.3)
Selected infrastructure deployment option (clause 7.1.2.2.1.4)
Variations of the adjusted non-functional NS design (clause 7.1.2.2.2.2)
Variations on the selected VNF deployment option (clause 7.1.2.2.2.2)
Variations on the selected infrastructure deployment option (clause 7.1.2.2.2.2)
Simple analytical models
Time complex analytical models
CoM
Configuration values and deployment adjustments
Training sets
Valid ation sets

Figure 7.1.2.1-2: Process of generating synthetic datasets for model training and validation

Thus, the following steps need to be performed to obtain the ANN models:

1) Identify the parts of the CoM that need to be replaced with ANN models for runtime. Analytical models of the CoM that are simple enough, e.g. do not depend on the size of the deployment (e.g. number of VL instances), can be used at runtime without change.

2) Identify the input parameters of the CoM that can change at runtime together with the range within which they can change. For example, a host type is selected at design time because of its associated availability and failure characteristics. If it is determined through their monitoring that the hosts of a chosen type fail more often, the availability calculations that selected this host type might not apply any more, thus, the NS would not meet its availability requirement with this selection. Accordingly, the availability and the failure rates of the host type need to be considered as changeable for which the ranges need to be identified. At runtime, they are monitored and changes to these parameters are detected/derived from operational data.

3) Determine the number of ANN models needed to replace the selected analytical models of the CoM. The number of models can depend on the relationship of the analytical models in the CoM and their parameters. For example, considering an ANN model and its output parameters, if according to the CoM there is a dependency between these output parameters, chaining multiple ANN models according to the dependencies might be beneficial. Thus, determine the input and output parameters of the ANN models, i.e. their label structure.

4) Generate sets of input parameters with values randomly changing within their range for the ANN models to be trained.

5) Determine the ANN model(s) architecture [i.11]. This includes the number of hidden layers and the number of their nodes in the model. It might be a trial-and-error process of the following steps to find the best values for these parameters.

6) Determine the *hyper parameters* of the ANN model(s): the activation function, loss function, learning rate, regularization rate, batch size, and the number of epochs for training [i.11]. Similar to the ANN architecture parameters, these may be tuned in a trial-and-error process of the following steps.

7) Apply the sets of input parameter values generated in step 4 to their respective analytical models of the CoM to generate the corresponding output parameters of these analytical models. In other words, generate the labels needed for the training of each ANN model.

8) Pre-process the sets of generated labels (e.g. scaling, normalization, etc.) and eliminate any duplicate among them. Then, split each set into two: training labels (usually 90 %) and validation labels (remaining 10 %).

9) Apply the training sets of labels to the respective ANN models to be trained.

10) If the model converges very slowly or diverges during training, meaning that the learning loss during training decreases very slowly or increases, the model architecture and hyper parameters in steps 5 and 6 need to be adjusted.

11) Apply the validation sets of labels to the respective trained ANN models to validate the models. If the validation of a trained ANN model is unsuccessful, that is, the output of the trained model diverges unacceptably from the output in the validation set, the ANN model needs to be changed starting with changing its parameter determined in steps 5 and 6.

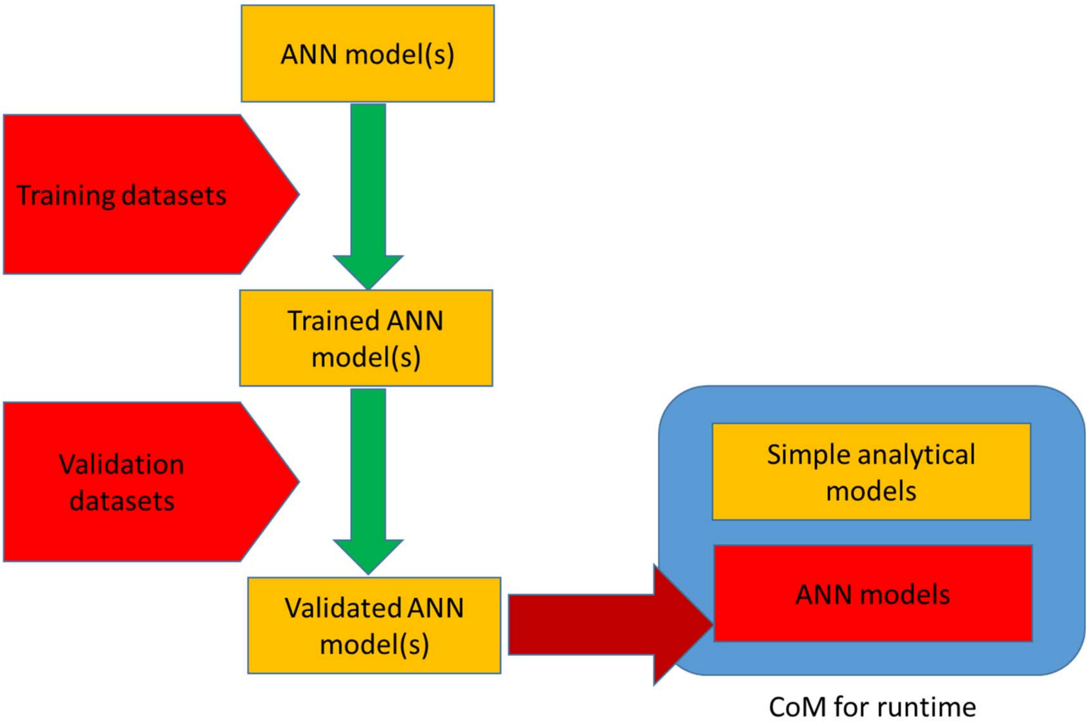


Figure 7.1.2.1-3: Process of model training and validation

Successfully validated models replace the respective analytical model(s) in the CoM as shown in Figure 7.1.2.1-3, which itself becomes part of the artifacts accompanying the NS to be used at runtime. Namely, at runtime the CoM can be used to determine new configuration parameter values for the NS as shown in Figure 7.1.2.1-4, so it can maintain the requested availability despite the changes occurring in the parameters that were identified in step 2. Different scenarios of using the CoM including the ANN models at runtime are presented in clause 7.1.3.

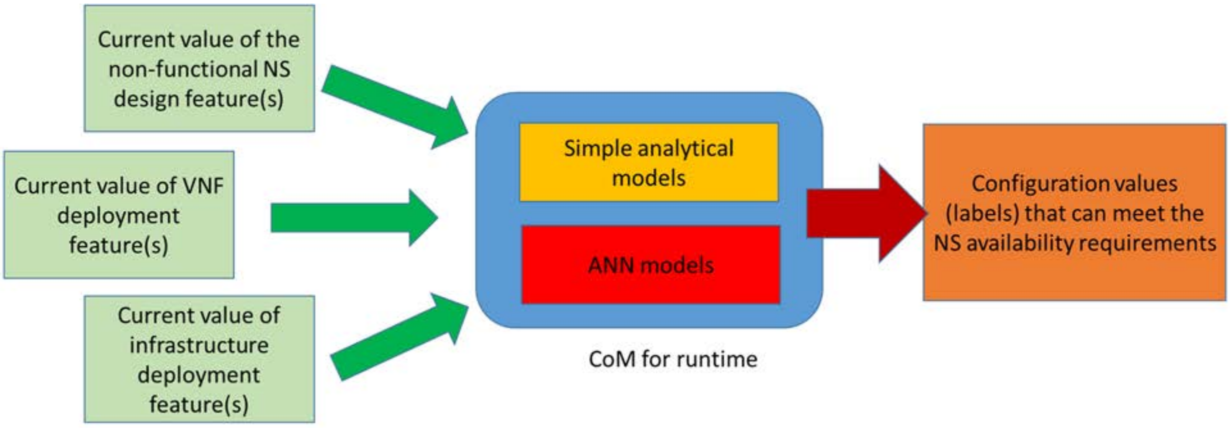

Current value of the non-functional NS design feature(s)
Current value of VNF deployment feature(s)
Current value of infrastructure deployment feature(s)
Simple analytical models
ANN models
CoM for runtime
Configuration values (labels) that can meet the NS availability requirements

Figure 7.1.2.1-4: Runtime use of the CoM including the trained and validated ANN models

7.1.2.2 Example of creating ANN models for runtime adjustments

7.1.2.2.1 Input and output information of the NS design

7.1.2.2.1.1 Overview

To present the process of creating ANN models, which can replace some analytical models in the CoM, an example NS shown in Figure 7.1.2.2.1.1-1 is considered.

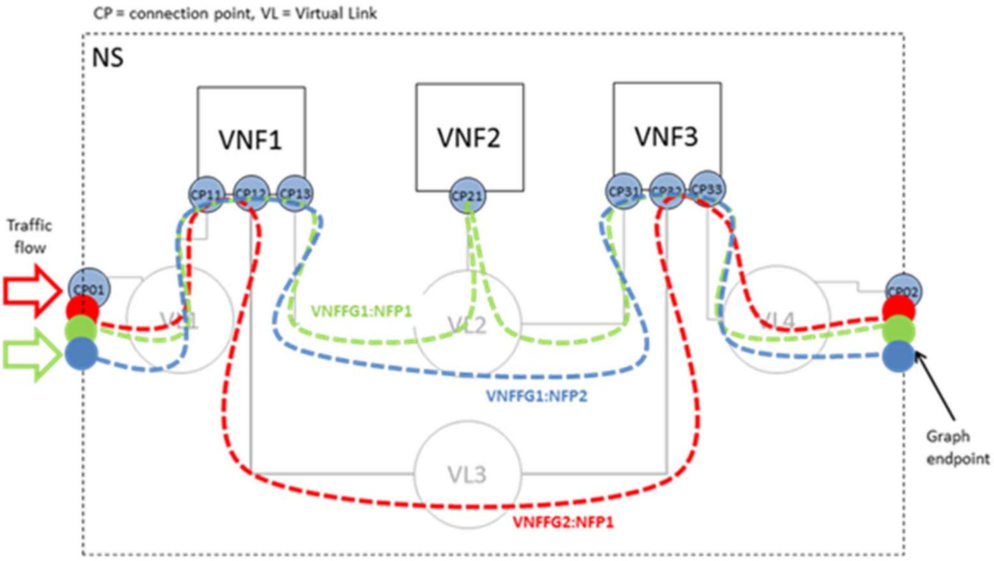


Figure 7.1.2.2.1.1-1: Example NS

The analytical models composing the CoM follow the principles of the models described in ETSI GR NFV-REL 010 [i.10], which have been further elaborated for the VNFs in [i.14].

First, the input used by the CoM to customize an NS to meet the requested NS availability is discussed together with its output parameters to determine the nature of the different parameters at runtime. I.e. whether they are constant, they can change on their own (i.e. to-be-monitored parameters), or they can change through re-configuration (i.e. configurable parameters/properties).

The input information for the CoM can be divided into four categories:

1) The functional design of the NS:
That is, the VNFs to be used to deliver the NS functionalities and their interconnections described by the VNF forwarding graphs. Each VNFFG also describes the network forwarding paths through which the requested NS functionalities will be delivered. For each of these NS functionalities, different availability requirements can be requested.

2) The non-functional design of the NS with respect to the workload:
   That is, at least one deployment flavour is provided for the NS design, which specifies the NS scale levels in terms of the number of VNF and VL instances to be used to provide the service capacity necessary for different volumes of workload and the requested availability characteristics for each NS functionality.

3) The deployment options of the VNFs of the NS, and their availability related characteristics at the VNF application level:
   These deployment options and characteristics can be selected and configured for the deployment. A subset of configuration options can also be reconfigured at runtime, while the selected characteristics need to be maintained by the deployment at runtime. The deployment options and characteristics can include the estimated availability of VNF instances deployed according to the different VNF deployment flavours, the application-level failure rate of these VNFs and their health monitoring options, as well as for stateful VNFs, their checkpointing features, etc.

4) The different options of infrastructure resources:
   These can be selected to deploy the VNFs and VLs of the NS based on their availability characteristics including the availability and failure rate of different host types as well as the different networking options.

Based on the above input, the output calculated for the NS design includes the most appropriate deployment options of the VNFs at the application level and for the hosting infrastructure resources as well as for the networking resources at the NS and VNF levels.

With respect to runtime, these input and output parameters are considered all together to determine:

- which ones remain constant;
- which can change (e.g. due to failures) and this change cannot be controlled; and
- which can be controlled, that is, re-configured at runtime if necessary, to compensate for any changes.

Accordingly, the parameters that remain constant and that can change on their own become the input parameters for the runtime adjustments, while the configurable parameters become the output. More details are presented using the example NS.

7.1.2.2.1.2 Functional and non-functional design information and requirements for the example NS

Figure 7.1.2.2.1.1-1 shows the functional design of the example NS. It has three VNFs and four VLs. They compose three Network Forwarding Paths (NFP), each providing a different NS functionality. VNF1 and VNF3 as well as VL1 and VL4 are shared by all NFPs. While VNF2 is part of NFP1 (in Figure 7.1.2.2.1.1-1 labelled as VNFFG1:NFP1) only. VL2 is shared by NFP1 and NFP2 (in Figure 7.1.2.2.1.1-1 labelled as VNFFG1:NFP2), and VL3 is only used by NFP3 (in Figure 7.1.2.2.1.1-1 labelled as VNFFG2:NFP1). NFP1 provides Func1, NFP2 Func2, and NFP3 Func3.

Using the CoM discussed in [i.14], the NS is designed to provide service capacity with given reliability characteristics for the different functionalities. The former can be expressed as minimum and maximum service data rate, while the latter as required availability, acceptable service disruption time and/or acceptable service data disruption.

In this case, the maximum and minimum service data rates are used to calculate the highest and lowest NS scale levels for the deployment flavour to provide the required service capacity.

The different availability requirements could be defined as follows:

- Required Availability (RA) is the percentage of time the service needs to be accessible.
- Acceptable Service Disruption Time (ASDT) is defined as the maximum amount of time in a year for which the service state can be lost. The service state is considered lost at each failure for the period from the last time the service state was checkpointed till this checkpoint is restored as shown in Figure 7.1.2.2.1.2-1 (the restoration is done by activating a standby instance).
- Acceptable Service Data Disruption (ASDD) is defined as the maximum tolerable amount of data that can be lost for a failure. Restoring an earlier service state typically results in the retransmission of service data (e.g. in a video streaming) starting with the restored state. Thus, this retransmission might also be undesirable and, therefore, might need to be limited.

All the above information is considered as constant at runtime since they describe the requirements that a deployed NS instance needs to meet and maintain.

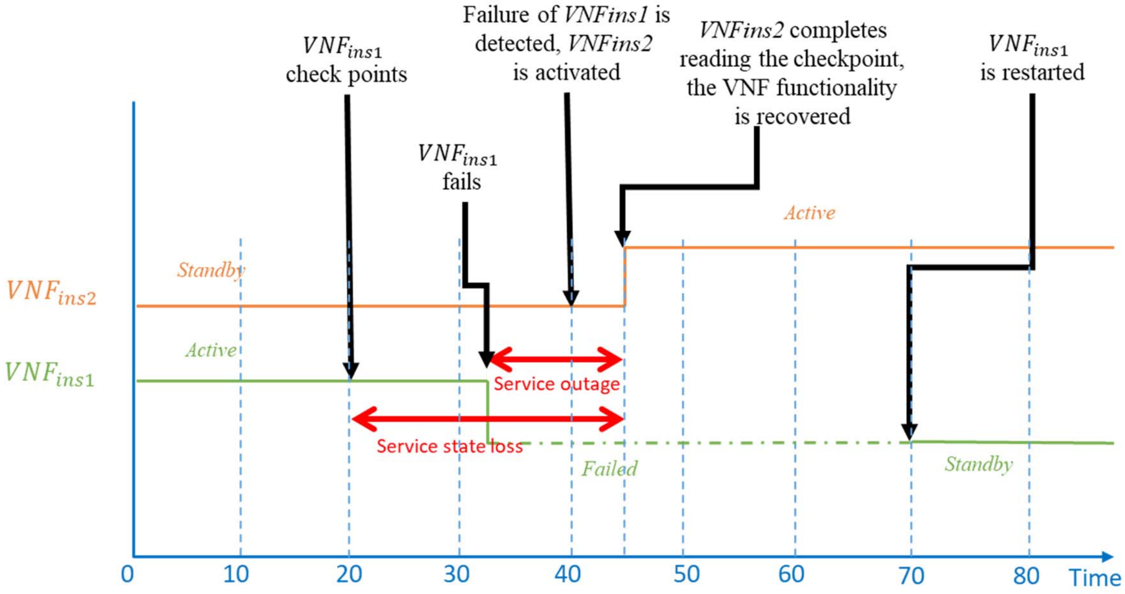


Figure 7.1.2.2.1.2-1: Example of service disruption concepts

To satisfy these capacity and availability requirements, the different NS scale levels are designed (as described in [i.14]) in terms of number of VNF instances (total number of instances = instances for active capacity + instances for standby capacity). The numbers of active VNF instances reflect the number of instances needed to handle the volume of the workload. The numbers of standby VNF instances ensure that the NS availability requirements (RA, ASDT and/or ASDD) are met.

The number of running VNF instances is considered re-configurable at runtime by NS scaling or changing the NS deployment flavour. Note however that the NFV-MANO is only aware of the total number of VNF instances. At the application level, the VNFs could be aware of the split in roles and/or this number might be configurable through VNF configurable properties defined in the VNFD.

7.1.2.2.1.3 Deployment options and characteristics of the VNFs of the example NS

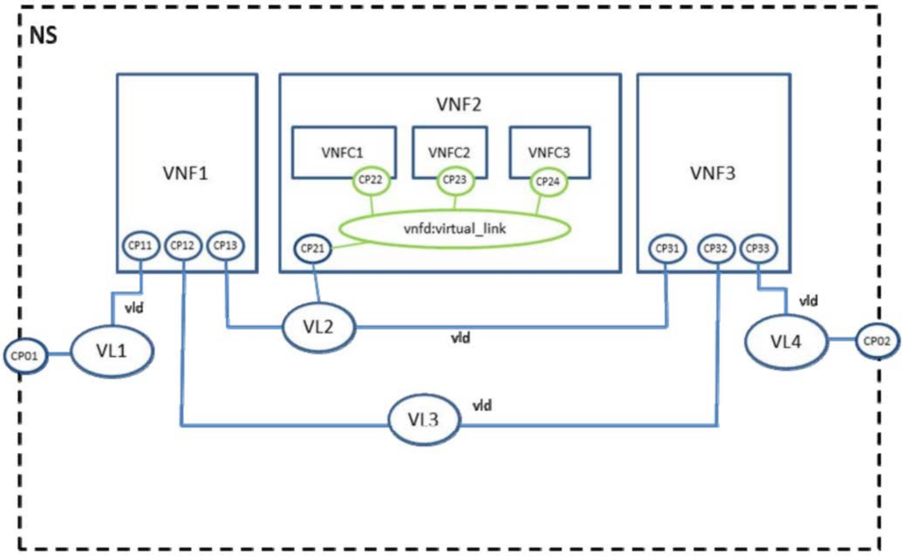


Figure 7.1.2.2.1.3-1: VNF components and internal VLs

Figure 7.1.2.2.1.3-1 shows the internal structure of the VNFs, i.e. the VNFCs and internal VLs for the NS of Figure 7.1.2.2.1.1-1. VNF1 and VNF3 each consists of only one VNFC. VNF2 has three VNFCs and an internal VL.

The VNF deployment flavours provide information about the VNFCs, internal VLs, and VNF scaling levels of each VNF.

At the application level, the VNF vendors could further characterize their VNFs by the availability and average failure rate (i.e. the average number of failures per year) of the VNF components expected to maintain and information facilitating the setting of their configurable properties.

Such information can provide the minimum health-check interval, the health-check interval increment, the restart time, the takeover time, if applicable, the checkpointing method, the checkpoint size, the checkpoint preparation time, the checkpoint commitment time, the minimum checkpointing interval, and the checkpointing interval increment.

Based on such information, at design time using the method described in [i.14], the networking option most appropriate for checkpointing is selected , as well as the health-check interval and the checkpointing interval are configured. At runtime however, only the last two parameters can be considered re-configurable and, therefore, would be considered as output of the CoM. The others are considered as input if they can change.

7.1.2.2.1.4 Options and characteristics of the infrastructure resources for the deployment of the example NS

To characterize the infrastructure available for the deployment, the different hosting options are described by their availability, average failure rate and (relative) cost. The different networking options are characterized by their latency, maximum bandwidth, maximum availability and failure rate of VLs that can be requested from the infrastructure.

With respect to runtime, the assumption is that due to failure or other reasons, the VNFC instances can be moved to hosts of a different type, e.g. even though host type 3 was selected for deployment, a VNFC is moved to host type 2 - with higher failure rate - due to shortage in the selected resources. It is also possible that the hosts of selected option underperform, thus, the actual value may be different from the value considered at design-time. Similar changes can be considered with respect to VLs and the networking options. As a result, all these parameters with their possible changes are considered as input parameters at runtime at their own level or as they impact input parameters of a higher level (e.g. the failure rate of a hosted VNF).

7.1.2.2.2 Creating the ANN model for the VNFs

7.1.2.2.2.1 Determining the label structure

According to the analytical models of the CoM discussed in [i.14], the availability characteristics of the NS depend on the availability characteristics of the VNFs and VLs composing the NS. In turn, the availability characteristics of the VNFs and VLs depend on the characteristics of the infrastructure. For example, a change in the failure rate of some hosts can change the failure rate of the hosted VNFs. Changes in the bandwidth and/or the latency of the network, due to some congestion, impact the checkpointing feature of the VNFs, therefore, also impacting the recovery of those VNFs and their availability characteristics. The models described in [i.14] show also that such changes can be compensated by changing the checkpointing interval, the health-check interval, and/or the number of standbys of the VNFs (e.g. used to prevent outage if the recovery lasts too long). The time complexity of the calculations using the analytical models is exponential, i.e. $O(2^x)$ see [i.14], where x is the number of VNFs. This can be improved by using heuristics, however, not necessarily to the extent desired for runtime use. Thus, depending on the size of the NS, the analytical models of [i.14] could be replaced with ANN models. This is possible as, with heuristics especially, they are suitable for design-time generation of synthetic data (i.e. label generation) for training such ANN models.

Considering the input/output parameters of the analytical models discussed in clause 7.1.2.2.1, at runtime:

- the size of the deployment changes according to the traffic to increase/decrease the active capacity;
- the characteristics of the infrastructure can change due to failures, network congestions, etc.

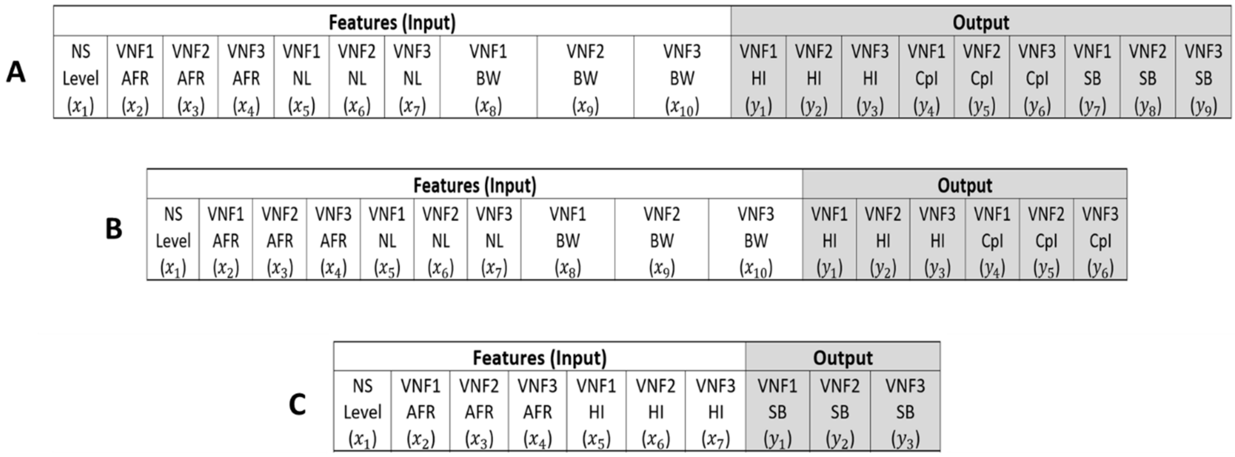
According to [i.14], these are reflected by the following parameters of the CoM of the NS: the NS scale level (NS Level), the Average Failure Rate (AFR) of each VNF, the latency (NL) and the Bandwidth (BW) of the network used for checkpointing by each VNF. The changes in these parameters (referred to as features in machine learning) are uncontrolled and need to be considered as input parameters for runtime. They will also need to be monitored in the deployment for the changes.

The configurable parameters that could be used to compensate for these changes are the health-check interval (HI), the checkpointing interval (CpI), and the standby capacity (SB). Thus, they need to be the output of the ANN models used at runtime. Hence, the label structure A shown in Figure 7.1.2.2.2.1-1 can be considered to determine them for the example NS, so that its required availability characteristics can be maintained in spite of the changes.

All other parameters of the analytical model can be considered constant and set to the values that were determined for the NS deployment. There is no need to not reflect them in the ANN models.

However, to determine the number of standby instances of a VNF, first, the health-check interval HI of the VNF is determined, then, the VNF outage time is calculated using the VNFs failure rate AFR and the health-check interval HI (for details see [i.14]). Finally, the number of standby instances SB is determined using the VNF outage time. In short, the number of standby instances of a VNF depends on the VNFs failure rate and health-check interval, which means that in the label structure A of Figure 7.1.2.2.2.1-1, both the health-check interval HI and the number of standby instances SB are output parameters at the same time. Therefore, their dependency might not be learnt properly by an ANN model trained on such data.

This logic of the analytical models could be better reflected by constructing two ANN models: one to determine the HI and CpI parameters using the label structure B of Figure 7.1.2.2.2.1-1. Then a second ANN, to determine the SB for each VNF using the label structure C shown in Figure 7.1.2.2.2.1-1, where the input features are the AFRs of the VNFs together with the HI values determined by the first ANN model. At runtime, these two ANN models will then be chained through the HI values produced by the first model, which are used as input feature by the second model.



A

Features (Input)										Output								
NS	VNF1	VNF2	VNF3	VNF1	VNF2	VNF3	VNF1	VNF2	VNF3	VNF1	VNF2	VNF3	VNF1	VNF2	VNF3	VNF1	VNF2	VNF3
Level	AFR	AFR	AFR	NL	NL	NL	BW	BW	BW	HI	HI	HI	CpI	CpI	CpI	SB	SB	SB
(x_1)	(x_2)	(x_3)	(x_4)	(x_5)	(x_6)	(x_7)	(x_8)	(x_9)	(x_{10})	(y_1)	(y_2)	(y_3)	(y_4)	(y_5)	(y_6)	(y_7)	(y_8)	(y_9)

B

Features (Input)										Output					
NS	VNF1	VNF2	VNF3	VNF1	VNF2	VNF3	VNF1	VNF2	VNF3	VNF1	VNF2	VNF3	VNF1	VNF2	VNF3
Level	AFR	AFR	AFR	NL	NL	NL	BW	BW	BW	HI	HI	HI	CpI	CpI	CpI
(x_1)	(x_2)	(x_3)	(x_4)	(x_5)	(x_6)	(x_7)	(x_8)	(x_9)	(x_{10})	(y_1)	(y_2)	(y_3)	(y_4)	(y_5)	(y_6)

C

Features (Input)							Output		
NS	VNF1	VNF2	VNF3	VNF1	VNF2	VNF3	VNF1	VNF2	VNF3
Level	AFR	AFR	AFR	HI	HI	HI	SB	SB	SB
(x_1)	(x_2)	(x_3)	(x_4)	(x_5)	(x_6)	(x_7)	(y_1)	(y_2)	(y_3)

Figure 7.1.2.2.2.1-1: Example label structures

Thus, multiple ANN models with different label structures are possible, and the choice can determine the precision of the generated values potentially at the price of a higher training time.

7.1.2.2.2.2 Label generation

To obtain the ANN model(s) at design time, the analytical models of the CoM can be used to generate synthetic training datasets. This can be achieved by solving the CoM for different variations of the input parameters. For the example NS, the following input feature changes can be considered to simulate possible changes in the infrastructure:

- Network delay and bandwidth:
    - To simulate failovers, single VNFs (as in case of a single network interface failover) or all VNFs using a given option (as in case of a router failover) can be switched to a different network option.
    - To simulate link congestions and router overloads, the delay and bandwidth values can be changed randomly in a given (e.g. 30 %) range of the original value.
- Host availability and AFR:
    - To simulate failover/migration of single VNFCs or all VNFCs using the same host type, their host type can be switched to another host type.

  - To simulate changes in the characteristics of single VNFCs or all VNFCs using the same host type, the AFR and availability (changes within the range of the last digit) values can be changed randomly.

- VL availability and AFR:
  - To simulate changes in the characteristics of VLs (VNF internal as well as VLs of the NS), the AFR and availability (changes within the range of the last digit) values can be changed randomly.

For each label to be generated, a random number of input features can be selected first and then these can be changed randomly within their applicable value ranges. Then with these input features the corresponding output values can be generated solving the CoM.

Once the desired number of labels have been generated, they would be pre-processed according to the general methodology [i.11], which includes encoding categorical data, data scaling and normalization. Then, all label duplicates would need to be removed to ensure that there will be no overlap between the training and the validation sets. Finally, part (e.g. 10 %) of the generated labels is set aside for the model validation, while the rest composes the training dataset.

7.1.2.2.2.3 ANN model construction

The ANN model construction can be performed according to the standard methodology [i.11]. This consists of selecting and tuning of the hyperparameters of the ANN model.

First, the number of hidden layers is determined, then the number of nodes for the hidden and the output layers. These hyperparameters determine the learning capacity of the ANN model. More complex problems require higher capacity, i.e. more hidden layer with more nodes.

Considering the sample NS and the label structures discussed in clause 7.1.2.2.2.1, the number of nodes in the output layer depends on the output parameters of the label structure, and, accordingly, on the number of VNFs in the NS. In case of a single-ANN model using label structure A of Figure 7.1.2.2.2.1-1, the number of nodes in the output layer is three times the number of VNFs in the NS. In case of chained ANN models using label structure B and C of Figure 7.1.2.2.2.1-1, it is twice the number of VNFs of the NS for the first ANN, and it is the number of VNFs for the second ANN.

It is also necessary to select an activation function for the different layers of the ANN, as well as a loss function and an optimization algorithm.

The activation function defines how the weighted sum of the input is transformed into an output from the nodes of a layer. The most often used activation functions are the Rectified Linear Unit (ReLU), the Logistic (Sigmoid) and the Hyperbolic Tangent (Tanh) functions. Considering the example NS, the Rectified Linear Unit (ReLU) function can be used as the activation function for the hidden layers for all ANNs, while the output layer can use a linear function since the problem is a regression.

The loss function is used to estimate the loss of the model, which needs to be reduced through repeated evaluations. The choice of loss function is specific to the modeling problem, such as classification or regression. For the ANN(s) of the example NS, the mean squared error loss function can be used.

The optimization algorithm can impact significantly the time needed to achieve good results. The optimization algorithms generally either use differentiable objective functions or non-differentiable objective functions. The derivative of a differentiable objective function characterizes the change in the function and optimization algorithms that can use this feature are fast and efficient. This is the case with the example NS, for which the ADAM (adaptive moment estimation) algorithm can be used.

Once the hyperparameters of the ANN model have been selected, the model can be trained using the training datasets. If the training is successful, the ANN model needs to be validated as discussed next. If the training does not progress well, the hyperparameters need to be tuned further.

7.1.2.2.2.4 ANN model validation

A successfully trained ANN model is validated using the validation datasets set aside in the step of label generation (see clause 7.1.2.2.2.2). If the ANN model cannot be validated, two reasons need to be considered.

In the simpler case, the hyperparameters could be tuned further, for example, if it turns out that the problem is more complex than the current learning capacity of the model. In this case, the previous ANN model construction step needs to be revisited (see clause 7.1.2.2.2.3).

In a more complicated case, the label structure does not reflect properly the input/output parameters of the problem. For example, for the sample NS the standard deviation for the numbers of standbys of VNF1 and of VNF2 are relatively high if a single ANN model is used with label structure A of Figure 7.1.2.2.2.1-1. If this is unacceptable, then two ANN models need to be constructed as in label structures B and C of Figure 7.1.2.2.2.1-1, which means that the steps starting with determining the label structure need to be repeated at least partially to come up with new label structures and the resulting ANN models.

Note that the ANN model(s) might need to be created for each NS instance individually as they are created considering not only the NS design, but also the available deployment infrastructure and the availability requirements the NS instance needs to meet.

One might also consider the input and output parameters at different levels. As an example, NS level ANN models have been discussed in these clauses; however, the applicable analytical models can be created and grouped separately for the infrastructure and for the VNF levels. Accordingly, ANN models can be created for each of these levels separately for runtime use following the methodology described in clause 7.1.2.

7.1.2.2.3 Model for the VL redundancy

The analytical model used at design-time to determine the number of VL instances that meet the availability requirements of the NS can be summarized as follows.

A functionality provided by an NFP as shown in Figure 7.1.2.2.3-1 is available if all its VNFs and VLs are available. A VNF or a VL is available if at least one of their instances is available, although the overall service performance may be degraded.

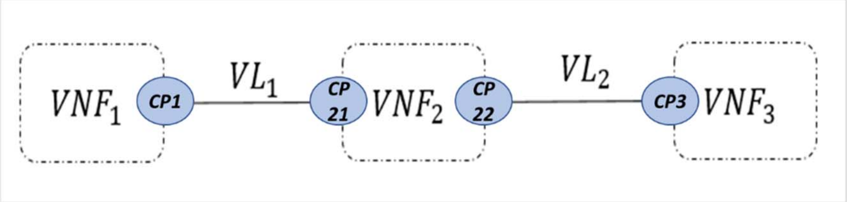


Figure 7.1.2.2.3-1: An NFP with three VNF and two VL profiles

The RA of the functionality (RA_{Func}) provided by the NFP can be met as long as the product of the availability of the VNFs and VLs is greater or equal than the RA:

$$(VNFs\ availability) * (VLs\ availability) \geq RA_{Func} \tag{1}$$

The *VNFs availability* has been addressed in clause 7.1.2.2.2. There, it was determined that ANN models could be created for runtime. Here, the focus is on the *VLs availability*, whether ANN models are needed. Since VLs are considered to be stateless, only their availability needs to be taken into account. Based on (1), for optimal solution, the availability of the VNFs and VLs should satisfy equation (2):

$$VNFs\ availability = VLs\ availability \geq \sqrt{RA_{Func}} \tag{2}$$

The VLs availability is the product of the availability of each VL (RA_{VL}). This means that for this example (Figure 7.1.2.2.3-1), each VL is required to have an availability as shown in equation (3):

$$RA_{VL} \geq \sqrt[4]{RA_{Func}} \tag{3}$$

Comparing the RA_{VL} with the maximum availability of VL instances (A_{vl-max}) the infrastructure can provide, which is one of the input parameters for the infrastructure, it can be determined whether redundancy is needed for a VL instance. If the maximum availability the infrastructure can provide for VL instances (i.e. A_{vl-max}) is greater than the RA_{VL} required from a VL availability, then one instance is enough for each VL. Otherwise, the VLs require redundancy.

NOTE: The reasoning behind considering the maximum availability the infrastructure can provide for VL instances is that for the instantiation of VLs connecting VNFs the SAL (service availability level) attribute is an input parameter. That is, the VIM can be asked to provide a VL instance with certain availability.

In case redundancy is needed, for each VL the minimum number of instances (n) needs to be determined so that it keeps the availability of the redundant VLs (A_{VL}) greater than or equal to the RA_{VL}.

Based on the principles outlined in ETSI GR NFV-REL 010 [i.10], the availability of n redundant VLs is calculated using equation (4):

$$A_{VL} = 1 - (1 - A_{vl-max})^{n} \qquad (4)$$

Therefore, inequation (5) can be used to determine the number of VL instances needed to meet the RA_{Func}:

$$n \geq \ log_{(1-A_{vl-max})}(1 - RA_{VL}) \qquad (5)$$

Inequation (5) can also be used for the runtime adjustment of VL redundancy. This is true even if the VL availability provided by the infrastructure changes. In this case, A_{vl-max} can be replaced in inequation (5) by the current availability of the VLs $A_{vl-current}$. This means that the calculation of VL redundancy is simple and can be used at runtime the same way as at design time. That is, no replacement with an ANN model is required. Otherwise the process of ANN model creation described for the VNFs in clause 7.1.2.2.2 can be followed.

7.1.3 Runtime use of ANN models

7.1.3.1 Overview of the model-based runtime adjustment

Clause 7.1.2 presented the design-time process of creating a CoM for an NS, which needs to satisfy certain availability requirements. Then, at runtime, this CoM can be used to evaluate the NS and determine if configuration adjustments are necessary for the NS instance to meet and maintain the requested availability characteristics. The CoM can include analytical models parameterized for and/or ANN model(s) trained for the requested availability characteristics for the NS instance.

For this purpose, for runtime, the to-be-monitored parameters of the CoM are defined as monitored parameters (or indicators from which they can be derived) of the NS. The evaluation of the NS instance using the CoM is triggered whenever some changes are detected in these monitored parameters.

When there is a change in a monitored parameter, the CoM is evaluated using the current values of the input parameters including the changed one. The generated output provides an answer about what configuration adjustments are necessary, if any, to the NS instance to maintain the requested availability characteristics. Such evaluation, if any, has been using analytical models in the past, which can be used the same way today within the mentioned time complexity limitations (see clause 7.1.2.2.2.1). Therefore, the focus of this clause is on the ANN models.

In case of ANN models, they are trained with certain training datasets, which determine the scope (see clause 7.1.2.2.2) within which the trained models are applicable. Namely, the input features (or input parameters) used in a training dataset determine the characteristics (or performance indicators) to be monitored for changes. The output parameters of the training dataset determine the configuration parameters that are considered for adjustments by the ANN model. Therefore, these parameters are expected to be adjustable at runtime, e.g. they are configurable properties of the VNFs.

Accordingly, at runtime the following steps are looped through repeatedly:

1) Detecting changes in any monitored parameter (or performance indicator) corresponding to an input feature/parameter of the CoM.

2) If their relation is not 1:1, mapping the detected change in the monitored parameter to a corresponding change in the input feature/parameter of the CoM, and complementing it with the current values of all other input features/parameters to obtain a complete input dataset.

3) Applying the complete input dataset to the CoM to determine the output parameters.

4) Mapping the output parameters to configuration adjustments required to the NS instance (e.g. configurable properties, scale level, etc.) and determining the operations needed to apply them to the NS instance and its constituents.

5) Executing the identified operations and storing the new values for the configurable parameters to be used in subsequent iterations if necessary.

7.1.3.2 Examples of model-based configuration adjustments at runtime

7.1.3.2.1 Introduction and goal

Let consider a deployed instance of the sample NS presented in clause 7.1.2.2. For this NS a CoM was developed including two chained ANN models for the VNFs as described in clause 7.1.2.2.2. These chained ANN models are to be used at runtime to determine if configuration adjustments are needed to the NS instance and its VNFs, so that the requested availability characteristics can be maintained. The evaluation of the ANN models is performed whenever a change is detected or reported for any of the monitored parameters related to the input features (or parameters) of the ANN models (i.e. for this example as shown in Table 7.1.2.2.2.1-2, NS scale level, average failure rate of the VNFs, latency and bandwidth of the links used by the VNFs for checkpointing and/or health-check). That is, these parameters are the monitored parameters for the deployed NS instance.

NOTE 1: It might not be possible to monitor directly a to-be-monitored parameter used as an input feature. In such a case, the parameters to be monitored are those from which the to-be-monitored parameter can be derived.

For the purpose of the discussion, an Availability Assurance Function (AAF) is assumed to perform the task of evaluating the ANN models with the changed parameters and initiating any configuration adjustments identified by the ANN models. No assumption is made which entity or entities can play this role.

For example, the VNF failure rate is one of the monitored parameters since it is an input feature for the ANN models. Therefore, its change might indicate a need for adjustments. The EM managing one or more instances of the VNF might collect this information and detect that due to more frequent failures than anticipated, the actual yearly average failure rate is higher than the one currently considered for the VNF. Accordingly, the EM notifies the VNFM indicating the change in the parameter, which in turn reports to the NFVO the received value. The NFVO, on the other hand, needs to check with the AAF to determine if configuration adjustments are necessary. Once the AAF receives the information, it evaluates the NS instance configuration using the chained ANN models (as part of the CoM) and determines the adjustments necessary to the Health-check Interval (HI), Checkpointing Interval (CpI), and the number of StandBy instances (SB) of the VNF. If the number of standby instances needs to be increased, the AAF selects the appropriate scale level (and possibly deployment flavour) and initiates the NS scaling operation with the NFVO (and possibly updating the NS as well). In addition, the AAF might also initiate the modification of the VNF info of the affected VNF instance(s) to update the related configurable properties. In turn, the VNFM will communicate these configuration changes to the VNF instance(s).

NOTE 2: It is also possible that the EM does not report VNF failure rates. In this case, the information can be derived from the VNF failures detected and reported by VNFM. The VNFM itself might derive the yearly average failure rate by potentially using its own model and report any changes. Alternatively, the VNFM might just notify the NFVO about the VNF failures, which will then derive this value.

Subsequently, after some configuration changes were made, it is possible that the NS instance is scaled to accommodate some increased traffic volume. Since configuration adjustments have been made, the NS scale levels included in the NS deployment flavour (NsDf) might not be applicable the same way anymore as they were designed for a given VNF failure rate, which has changed. Thus, the other NS scale levels and deployment flavours need to be checked as well. Any time the NS needs scaling, its configuration needs to be re-evaluated, if additional configuration adjustments are necessary. Scaling the NS can be triggered by the NFVO itself or the NFVO can be asked to scale the NS. In either case, the NFVO needs to check with the AAF if other configuration adjustments are needed, which are then applied together with the scaling as described above.

With respect to the monitored parameters for the virtual links interconnecting the VNF instances for health-check and/or checkpointing, the latency and bandwidth are usually parameters requested by the NFVO from the VIM at the time of the instantiation of the NS and its virtual links. Accordingly, the VIM is expected to notify the NFVO of any discrepancy. However, it is possible that if the VIM detects an issue, it might take corrective actions at its level, e.g. add more redundancy to the network implementing the virtual link(s). Thus, only if the actions at the resource level were not sufficient, further configuration adjustments will be needed at the NS level. For the resource level adjustments, the VIM might use a model similar to the one discussed in clause 7.1.2.2.3.

The following clauses present such scenarios in more details.

7.1.3.2.2 Actors and roles

Table 7.1.3.2.2-1 describes the use case actors and roles.

Table 7.1.3.2.2-1: Model-based runtime configuration adjustment actors and roles

#	Role	Description
1	NFVO	NFV Orchestrator managing the NS instance
2	VNFM	VNF Manager managing one or more VNF instances of the NS instance
3	VIM	VIM managing the virtualised resources of the NS instance
4	EM	Element manager responsible for managing one or more VNF instances
5	AAF	Availability Assurance Function, which on request evaluates changes in the NS with respect to the requested availability characteristics and determines if configuration adjustments are necessary to maintain these characteristics. The function might or might not be part of NFV-MANO. When it is part of NFV-MANO, it might be part of the NFVO only or distributed to different NFV-MANO functional entities.

7.1.3.2.3 Pre-conditions

Table 7.1.3.2.3-1 describes the use case pre-conditions.

Table 7.1.3.2.3-1: Model-based runtime configuration adjustment pre-conditions

#	Pre-condition	Additional description
1	The NS has been onboarded with a CoM and it is available for the AAF	The CoM includes the ANN models for the NS to be used by the AAF for runtime configuration adjustments of the NS to maintain its requested availability characteristics based on changes in the to-be-monitored parameters
2	The output parameters of the ANN models are configurable parameters	The HI and CpI parameters produced by the ANN models as output are declared as configurable properties in the respective VNFDs
3	The NFVO, VNFM, VIM, EM and the AAF are operating correctly	
4	The NS has been instantiated according to the selected NsDf	The CoM model provided as part of pre-condition #1 has been built for the NsDf selected for instantiation
5	The to-be-monitored parameters are being monitored	The monitored parameters include the NS scale level, the failure rate of VNFs, and/or the network latency and bandwidth used by the VNFs, or monitored parameters from which these parameters can be derived
6	The NFV-MANO entities have subscribed with each other and with their respective managed entities to receive notifications	The notifications can - among others - indicate changes in the values of monitored parameters, or alarm conditions for managed entities as well as clearing such alarm conditions
7	The NS instance provides its services according to the requested availability characteristics	The CoM has been built to identify changes in order to maintain these requested availability characteristics

7.1.3.2.4 Post-conditions

Table 7.1.3.2.4-1 describes the use case post-conditions.

Table 7.1.3.2.4-1: Model-based runtime configuration adjustment post-conditions

#	Post-condition	Additional description
1	The NS instance continues to provide its services according to the requested availability characteristics	The AAF has made any adjustment if necessary to maintain the availability characteristics

7.1.3.2.5 Flow description of NS scaling with no other configuration adjustment

Table 7.1.3.2.5-1 describes the use case flow for the scenario when in response to a change in a monitored parameter the NS is scaled to a different NS scale level. This new NS scale level and the current values of the other monitored parameters can satisfy the requested NS availability without any further configuration adjustment.

Table 7.1.3.2.5-1: Flow description of NS scaling with no other configuration adjustment

#	Actor/Role	Action/Description
Begins when	NFVO	Based on some indicators the NFVO determines that the NS instance needs to be scaled to a given NS Scale Level (SL2).
Step 1	NFVO -> AAF	The NFVO informs the AAF that the new NS scale level should be SL2.
Step 2	AAF	The AAF collects the current values for all other monitored parameters and runs the ANN models to check if additional adjustments are needed. (see note.) It determines that the HI and CpI values output by the model for SL2 are the same as the current values of HI and CpI. While the output SB value is the same as the SB value associated with SL2.
Step 3	AAF -> NFVO	The AAF informs the NFVO that SL2 can be deployed without any further configuration adjustment needed.
Ends when	NFVO	The NFVO proceeds with the scaling operation as usual.
NOTE: The AAF might have saved the values used for the different input features at the last evaluation and uses them for the input features for which no change is reported. Alternatively, the AAF might initiate the operations necessary to pull this information from the system.		

7.1.3.2.6 Flow description of configuration adjustments due to monitored parameter change

Table 7.1.3.2.6-1 describes the use case flow for the scenario when configuration adjustments are needed in response to a change in the average failure rate monitored parameter.

Table 7.1.3.2.6-1: Flow description of configuration adjustments due to monitored parameter change

#	Actor/Role	Action/Description
Begins when	EM	The EM detects that the average failure rate for the VNF, which is a monitored parameter, has changed compared to the value currently assumed for the VNF.
Step 1	EM -> VNFM	The EM notifies the VNFM indicating the change in the average failure rate monitored parameter.
Step 2	VNFM -> NFVO	The VNFM notifies the NFVO indicating the change in the average failure rate monitored parameter.
Step 3	NFVO -> AAF	The NFVO informs the AAF about the new value of the average failure rate of the VNF.
Step 4	AAF	The AAF collects the current values for all other monitored parameters and runs the ANN models to check if configuration adjustments are needed. (see note.) It determines that the HI and CpI configuration parameters of the VNF instances need to be changed. The values output by the ANN models differ from those currently set in the VnfInfo of the VNF instances.
Step 5	AAF -> NFVO	The AAF provides the NFVO with the configurable properties of the affected VNF instances with the new values of the HI and CpI parameters.
Step 6	NFVO -> VNFM	The NFVO informs the VNFM about the changes by invoking the *Modify VNF Information* operation
Ends when	VNFM -> VNF	The VNFM informs the VNF about the changes in the configurable properties by invoking the *Set Configuration* operation.
NOTE: The AAF might have saved the values used for the different input features at the last evaluation and uses them for the input features for which no change is reported. Alternatively, the AAF might initiate the operations necessary to pull this information from the system.		

7.1.3.2.7 Flow description of NS scaling with additional configuration adjustments

Table 7.1.3.2.7-1 describes the use case flow for the scenario when in response to a change in a monitored parameter reported as an indicator, the NS needs to be scaled to a new NS scale level. However, this new NS scale level with the current values of the other monitored parameters cannot satisfy the requested NS availability according to the ANN. Therefore, a different NS scale level and further configuration adjustments are identified.

Table 7.1.3.2.7-1: Flow description of NS scaling with additional configuration adjustments

#	Actor/Role	Action/Description
Begins when	NFVO	Based on some indicators, the NFVO determines that the NS instance needs to be scaled to a given NS scale level (SL2).
Step 1	NFVO -> AAF	The NFVO informs the AAF that the new NS scale level should be SL2.
Step 2	AAF	The AAF collects the current values for all other monitored parameters and runs the ANN models to check if adjustments are needed. (see note.) It determines that the HI and CpI values output by the model for SL2 differ from the current values of HI and CpI. Also, the output SB value differs from the SB value associated with SL2, hence another appropriate NS scale level (SL3) is identified.
Step 3	AAF -> NFVO	The AAF informs the NFVO that SL3 needs to be deployed and further configuration adjustments are needed. The AAF provides the NFVO with the new values of the appropriate configurable properties.
Step 4	NFVO -> VNFM	The NFVO proceeds with the scaling operation as usual and also updates the configurable properties of the affected VNF instances by invoking the *Modify VNF Information* operation.
Step 5	VNFM -> VIM	The VNFM obtains the resources required for scaling the NS.
Ends when	VNFM -> VNF	The VNFM informs the VNF about the changes in the configurable properties by invoking the *Set Configuration* operation.
NOTE: The AAF might have saved the values used for the different input features at the last evaluation and uses them for the input features for which no change is reported. Alternatively, the AAF might initiate the operations necessary to pull this information from the system.		

7.1.3.2.8 Flow description for configuration adjustments due to change in VL characteristics

Table 7.1.3.2.8-1 describes the use case flow for the case when the VIM detects that the virtualised network characteristics supporting a VL do not satisfy anymore the requested QoS. Some of this might be compensated by measures taken by the VIM. The QoS parameters that cannot be satisfied are reported to the NFVO and trigger the AAF to check whether additional configuration adjustments are needed to VNF(s).

Table 7.1.3.2.8-1: Model-based runtime configuration adjustment flow description

#	Actor/Role	Action/Description
Begins when	VIM	The VIM detects that (e.g. due to congestion) some VLs requested with certain QoS parameters cannot be satisfied by their supporting virtualised network resource. The VIM also determines that it cannot achieve the requested QoS parameters (see note 1).
Step 1	VIM -> NFVO	The VIM notifies the NFVO indicating the changed latency and bandwidth characteristics of the affected VLs based on the approximation.
Step 2	NFVO -> AAF	The NFVO determines the NS is impacted by the changes and informs the AAF about the changed characteristics.
Step 3	AAF	The AAF collects the current values for all other monitored parameters and runs the ANN models to check if adjustments are needed to the NS (see note 2). It determines that some of the HI and CpI values output by the model differ from the current values of those configurable properties.
Step 4	AAF -> NFVO	The AAF informs the NFVO that configuration adjustments are needed for some VNFs and provides the changed values of the appropriate configurable properties.
Step 5	NFVO -> VNFM	The NFVO updates the affected VNF instances by invoking the *Modify VNF Information* operation.
Ends when	VNFM -> VNF	The VNFM informs the VNF about the changes in the configurable properties by invoking the *Set Configuration* operation.
NOTE 1: It is assumed, that the VIM can take measures internally to provide virtualised resources with the requested QoS parameters and maintain them over time. However, these internal measures might not always be sufficient to achieve this goal, in which case the VIM will report any discrepancy. NOTE 2: The AAF might have saved the values used for the different input features at the last evaluation and use them for the input features for which no change is reported. Alternatively, the AAF might initiate the operations necessary to pull this information from the system.		

7.1.3.2.9 Flow description of handling changes in the characteristics of virtualised resources used by a VNF

Table 7.1.3.2.9-1 describes the flow for the case when the VIM detects that the characteristics of some virtualised resources hosting a VNF do not satisfy the characteristics requested for them. The virtualised resources could be virtualised compute and/or networking resources. The VIM notifies the VNFM of the new characteristics for the virtualised resources. The VNFM, in turn, reports the change to the EM of the affected VNF. The EM evaluates the impact on the VNF characteristics and, if it cannot maintain its requested characteristics, the EM reports the new VNF characteristics to the VNFM to trigger the flow described in Table 7.1.3.2.6-1.

Table 7.1.3.2.9-1: Model-based runtime configuration adjustment flow description

#	Actor/Role	Action/Description
Begins when	VIM	The VIM detects that some virtualised resources do not satisfy any more their requested QoS. The changes go beyond those that the VIM can handle to maintain the requested QoS (e.g. physical resources are not available).
Step 1	VIM -> VNFM	The VIM notifies the VNFM indicating the new QoS characteristics of the impacted virtualised resources.
Step 2	VNFM -> EM	The VNFM identifies the VNF affected by the changes and notifies its EM about the new QoS values.
Step 3	EM	The EM evaluates the impact of the changes and takes any action necessary to mitigate them.
Ends when	EM	The EM evaluates the new VNF characteristics whether they meet the requested QoS. If the QoS requested for the VNF is met, no further action takes place. If the QoS cannot be met, the flow continues as described in clause 7.1.3.2.6 with reporting changes not (only) in the failure rate, but in any impacted monitored parameters.

7.2 Root cause analysis

7.2.1 Introduction

The behaviour of a large system, such as the NFV system, may deviate from its intended behaviour. Such a situation needs to be detected and resolved in a timely manner for reliable operations. There are a number of mechanisms in place for detection: at an interface the sign of anomaly could be the reception of wrong results or no result at all. Results may also be received later than expected or the system may consume an unexpected amount of resources. In such cases, the fault which is the root cause for such anomalies and/or failures needs to be found and removed.

Knowing the root cause may allow for faster recovery and the removal of the cause. If the failure has already happened, it is important to recover the system as soon as possible. If the root cause cannot be removed right away (e.g. software bug), understanding it allows to take precautions so that the failure does not happen again, or it is less severe. When the recovery depends on root cause analysis (RCA) results, it is preferred to perform the analysis real-time. Otherwise, it can be done offline.

Faults may manifest as errors or failures. An error does not manifest in the delivered service and may not even be detectable by the fault management system. For a successful RCA, the right data needs to be collected. The data required to perform successful RCA may not be limited to data available at the interfaces and may include also data internal to a part of an NFV system.

The amount of data that is created during operations may be very high. Therefore, collecting and storing them could reduce the system performance while not all data might be necessary. Hence, it needs to be evaluated what data supports an RCA in the best way. With the selected data, a model of the system might be created that represents the system during normal processing. This can be done even without knowing the semantics of the internal data that may not be disclosable because of confidentiality reasons (for an analysis of potential available data, see ETSI GS NFV-REL 005 [i.12]). In the area of machine learning selection of the pieces of data - called features - to be collected is referred as feature engineering. There are machine learning techniques that help in this selection process; however, domain knowledge could be essential as well.

In the remaining subclauses, the use case for a real-time fault detection and localization function will be explored, since for an operator, it is extremely valuable to have such a function that allows for timely actions to mitigate and resolve faults underlying to detected anomalies.

7.2.2 Using self-organizing maps for root cause analysis

7.2.2.1 Challenge of root cause analysis

A real-time fault detection and localization function would allow for timely actions to mitigate and resolve faults underlying to detected anomalies. The problem is that inferring faults in an automated fashion is an extremely challenging task especially in dynamic environments such as NFV systems. In such systems, many Key Performance Indicators (KPIs), notifications, and alarms can be collected, correlated, and evaluated, while the situation can change quickly and unpredictably. The collected data are also susceptible to noise.

In the literature, different approaches have been proposed such as an automated monitoring analysis [i.27], which relies on a decision-tree based machine learning approach to automate bottleneck detection in networks. Another approach [i.28] uses an unsupervised behavior learning system for predicting and preventing anomalies. This system utilizes a Self-Organizing Map (SOM) model (see clause 7.2.2.2 for more details), which is trained only with normal (non-anomalous) data for the formation of clusters. Using such a model at runtime, the input can be classified as normal or anomalous. However, this approach cannot identify the reason for an anomaly, that is, it cannot localize a fault. In [i.26], this approach is extended towards more flexible 2-layered SOMs, which uses all available training data, that is, normal and anomalous.

In [i.26], the real-time fault localization problem is divided into two parts:

1) real-time anomaly detection, that is, detecting potential faults in the system using different gathered KPIs (e.g. device statistics collected at the server side); and

2) real-time RCA using the collected data to determine possible causes of detected faults.

This approach has been applied to infer performance faults that cause service degradation due to the lack of certain resource(s) in a cloud environment, therefore, the approach may be applied to the NFV environment in a similar manner.

SOM-based techniques have been available for some time. They exist in both supervised and unsupervised versions. SOMs can be trained and used as a classifier or as a model to detect anomalies in the data. From an RCA perspective, it is more important, however, that a trained model can be used as an effective and low overhead method for fault localization using a supervised 2-layered variant of SOMs. [i.26] demonstrated that a single generalized trained map is sufficient to localize different types of resource faults (e.g. CPU, memory, I/O).

7.2.2.2 Traditional SOM

The traditional Self-Organizing Map (SOM) is an unsupervised learning technique introduced in the early 1980s. It can be viewed as a single-layered artificial neural network. Thus, typically, a SOM is represented as, but not limited to, a rectangular or hexagonal grid in two dimensions. Accordingly, the size of the SOM - a hyperparameter usually determined by experimentation - can be defined by the length and the height ($L \times H$) of the grid in terms of the number of nodes. Also, the position of a node can be described by its coordinates in the grid (e.g. *N(l, h)*). SOMs are capable of representing high-dimensionality data in a low-dimensionality view without losing the topological properties of the data. The steps to train a SOM are as follows:

Step 1: Initialize the weights of each node in the SOM, for example, by assigning random values within the range of the input values to each weight vector:

$$W_i = [w_{i1}, w_{i2}, ..., w_{ik}]$$

where W_i is the weight vector of node *i* and *k* is the number of features in the input data.

Step 2: Present a randomly chosen data sample (i.e. input vector $X= [x_1, x_2, ..., x_k]$) from the training set to the SOM.

Step 3: Find in the SOM the node that most closely resembles (or closest to) input vector *X*: this node is referred to as the Best Matching Unit (BMU). The closeness of a node is determined by measuring the distance from input vector *X* to that node in the SOM using, for example, the Euclidean distance of equation (6):

$$d_i = \sqrt{\sum_{j=1}^{j=k}(x_j - w_{ij})^2} \qquad (6)$$

where *k* is the number of features in the input data, x_j represents the *j*th feature component of the current input vector *X*, and w_{ij} represents the weight of the *j*th feature component in the weight vector W_i of node-*i*.

Step 4: Find each node belonging to the neighbourhood of the BMU based on the neighbourhood function. The neighbourhood function of a SOM decreases with each time-step to allow the SOM to reach convergence. In [i.26], only the four nodes with a radial distance of 1 are considered for the neighbourhood.

Step 5: Update the weights of each node found in Step 4, using function (7):

$$W_i(t+1) = W_i(t) + \eta(t) \times Nf_c \times \left(X(t) - W_i(t)\right) \qquad (7)$$

Where *t* is the current step, $\eta(t)$ is the learning rate, which is reduced over time, and Nf_c is the neighbourhood function centred on node *c* (i.e. BMU). Nf_c has a value between 0 and 1 depending on how far the node to be updated is from node *c*. As a result of such an update, the node will more closely resemble the input vector *X*.

Step 6: Repeat Steps 2 to 5 *T* times, where *T* is number of iterations chosen, e.g. the size of the training set.

Since the learning rate is reduced with the number of iterations, the map will converge producing a set of clusters each grouping together similar training data.

The training of a SOM is usually performed with part of the available data, e.g. 70 %. The rest of the data (30 %) is then used to validate the trained SOM. Also, multiple SOMs could be trained from the same data set, such as in case of K-fold cross-validation. In this case, the best SOM is selected during validation based on some performance metrics for the SOM (see Annex D for more details).

7.2.2.3 Anomaly detection using 2-layered SOMs

To trigger the RCA at real-time, first an anomaly, such as a service degradation, needs to be detected. An unsupervised SOM trained as described in clause 7.2.2.2 is capable of fulfilling this task. However, as mentioned earlier, the unsupervised version of SOM does not have the capability of performing RCA. Thus, here a supervised extension of SOM, also referred as 2-layered SOM technique, is described which uses labelled data and can be used to automate fault localization.

The 2-layered SOM is similar to the traditional SOM with few important differences. As its name suggests, the 2-layered SOM contains two maps:

- the first layer for *X* features, as discussed in clause 7.2.2.2 for the traditional SOM; and
- the second layer for *Y* features, which include the labels as used for supervised learning. That is, input features with associated output as applicable to the Y features.

For example, the X features can be KPI metrics related to the virtualisation layer collected by the VIM, while the Y features could be input metrics for Service Level Objectives (SLOs) at the NS layer evaluated by the NFVO or the OSS, for which an output state can be assigned based on whether the SLOs of the SLA have been violated or fulfilled. It is assumed that the data collected for the X and Y features are time-stamped based on a global clock, so that they can be correlated.

Since from an NFV system large amounts of data can be collected, it is recommended to select appropriate subsets as X and Y features. Thus, applying feature engineering is recommended with the use of domain knowledge to reduce training time and improve precision. Feature engineering may include feature aggregation as well as removal of highly-correlated features from the available sets. The goal is to select features that reflect best different underlying faults, therefore helping to localize them.

At training of the 2-layered SOM, the nodes of the two layers are considered, based on their position, as pairs forming a unit. For each input sample extracted from the training set, the distance to each unit is calculated by finding the shortest combined weighted distance to both layers. For this, first the distance is calculated to each node of the first, or X-layer using only the *X* features of the input sample. Then the distance is calculated to each node of the second, or Y-layer using only the *Y* features. The weighted sum of these two distances is the combined weighted distance, which is used to find the BMU (the steps are similar to those described in clause 7.2.2.2). The weight (referred to as x-weight) is a value between 0 and 1 and is a hyperparameter of the model. The state represented by each unit is calculated by averaging the *Y* values of the training samples mapped to those units, which leads to class probabilities. E.g. if two Y values have been mapped to the node, one with the output of a healthy state and the other with an unhealthy state, this means that the node represents the healthy and unhealthy states with probabilities of 50-50 %.

Part of the collected data is used for the training of the 2-layered SOM and the rest for its validation in a similar manner as described in clause 7.2.2.2. Once validated, the trained 2-layered SOM can be used for anomaly detection in new samples as follows:

Step 1. Present a new sample $X = [x_1, x_2, ..., x_k]$ to the SOM.

Step 2. Compare X to the weight vector $W_i = [w_{i1}, w_{i2}, ..., w_{ik}]$ of each node of layer X in the trained map. The BMU for X is found using the Euclidean distance on layer X.

Step 3. Based on the state represented by the BMU, which is either healthy (e.g. SLA fulfilled) or non-healthy (e.g. SLA violated), determine the state for X, which is assumed to have the same value as the BMU.

7.2.2.4 Fault localization using 2-layered SOMs

This fault localization approach is based on the assumption that a fault will manifest in certain individual features of a non-healthy sample when it is compared to a healthy sample. Thus, the approach requires a dissimilarity metric.

The dissimilarity is measured by comparing a non-healthy sample to a healthy sample by evaluating the differences in the individual feature values and the features that have the largest difference (or dissimilarity) are considered as the most likely cause of, for example, an SLA violation.

An inherent property of SOM is that during training, the weight vectors of certain nodes are updated more often; therefore, they more closely resemble each other. As a result, these nodes cluster together and this property makes the SOM well suited for the above dissimilarity measurement scheme, which is at the base of this fault localization approach.

Accordingly, if the anomaly detection method described in clause 7.2.2.3 determines that a new sample *X* represents an unhealthy (e.g. SLA violation) state, it is further analysed for fault localization.

The fault localization steps work as follows:

Step 1. Perform an anomaly detection on sample *X* as described in clause 7.2.2.3.

Step 2. If *X* is mapped to a healthy node, i.e. it is evaluated as healthy then no further processing is needed. Otherwise, proceed to Step 3.

Step 3. Locate ℕ, the set of *n* healthy nodes nearest to *X* using the Manhattan distance of equation (8).

$$m_i = \sum_{j=1}^{j=k} |x_j - w_{ij}| \tag{8}$$

where *k* is the number of features in the input vector *X*, x_j represents the j^{th} feature component of the current input vector, and w_{ij} represents the weight of the j^{th} feature component in the weight vector W_i of node *i*.

Step 4. Calculate the dissimilarity vector DS of equation (9) for X.

$$\text{DS} = \sum_{i=1}^{i=n} DS_i(X, N_i) \tag{9}$$

where:

$$\text{DS}_\text{i}(\text{X}, \text{N}_\text{i}) = [|\text{x}_1 - \text{w}_{\text{i}1}|, |\text{x}_2 - \text{w}_{\text{i}2}|, \ldots, |\text{x}_\text{k} - \text{w}_{\text{ik}}|],$$

with $N_i \in ℕ$ a healthy node in ℕ, $n = |ℕ|$ the number of healthy nodes in ℕ, $[w_{i1}, w_{i2}, \ldots, w_{ik}]$ the weight vector of node N_i and *k* the number of features in input vector *X*.

Step 5. Use the dissimilarity vector *DS* to interpret the nature of the fault manifesting in sample *X*. That is, each element of the dissimilarity vector corresponds to a feature x_j in *X* and each feature is indicative of the manifestation of some fault. Thus, sorting the elements of *DS* in descending order provides a ranking of probable causes of the detected anomaly. The top ranked metrics indicate the most likely cause of the detected anomaly.

7.2.2.5 Evaluation of 2-layered SOM for RCA

Different aspects of the use of 2-layered SOM for RCA have been evaluated in [i.26]. Namely, specialized and generalized maps of different sizes were compared for their accuracy of anomaly/fault detection as well as localization. The results are summarized here, in the present clause 7.2.2.5.

Two sizes (5x5 and 20x20 nodes) of the different maps were evaluated. Since the 20x20 map (or in general a larger map) contains more nodes, it can capture more subtle differences in the training data as well as in the data samples to be analysed. Thus, a larger map size seems to be more optimal. The solution also scales favourably as the map size increases.

A specialized map is trained with data containing information about a given type of faults only, while a generalized map is trained with data including all types of faults. In the latter case, the trained models can be evaluated for accuracy with respect to specific fault types, as well as, combination of different fault types. The accuracy of a machine learning method is usually measured as the classification accuracy (CA) of equation (10):

$$CA \coloneqq \frac{True\ Positives + True\ Negatives}{Total\ Test\ Samples} \tag{10}$$

However, when class data is skewed such as in this case, that is, the data are heavily biased towards normal data (i.e. non-faulty behaviour), it is important to measure also balanced accuracy (BA). BA takes into account both true positive rate and true negative rate as show in (11):

$$BA \coloneqq \frac{True\ Positive\ Rate + True\ Negative\ Rate}{2} \tag{11}$$

where:

the true positive rate (also referred to as sensitivity) gives the probability that an anomaly is detected when there is an underlying fault. It is calculated by equation (12):

$$True\ Positive\ Rate = \frac{number\ of\ true\ positives}{number\ of\ true\ positives + number\ of\ false\ negatives} \qquad (12)$$

the true negative rate (also referred to as specificity) gives the probability that no anomaly is detected when there is no underlying fault. It is calculated by equation (13):

$$True\ Negative\ Rate = \frac{number\ of\ true\ negatives}{number\ of\ true\ negatives + number\ of\ false\ positives} \qquad (13)$$

In this respect, generalized maps prove to have a fault detection accuracy very similar to specialized maps with a difference only up to 3 % considering their BA.

For fault localization, specialized maps have difficulties to localize certain types of faults as the primary cause (this has been observed for both memory and I/O faults), while generalized maps have no such drawback; therefore, generalized maps are the preferred solution.

Thus, the generalized 2-layered SOM can achieve good prediction accuracy with regards to detecting faults in the system. It also has good diagnostic capabilities with regards to localizing any detected faults. That is, there is no need to train specialized maps for each specific fault type, which would add significant data preparation and training time.

In addition, having to train and maintain only one map, a generalized map significantly reduces the system complexity and overhead.

A further benefit of the 2-layered approach is that there is no need to provide data labelled with high granularity (i.e. with the exact type of fault for each sample). Using non-zero x-weight for the information from the Y-layer resulted in a great improvement in localization performance.

7.2.2.6 Runtime use of the 2-layered SOM model for NFVI resource fault localization

7.2.2.6.1 Introduction

Clauses 7.2.2.3 and 7.2.2.4 presented the approach of using 2-layered SOM models at runtime to detect anomalies and localize faults potentially causing service degradation. In clause 7.2.2.6, the use of such models in an NFV system is illustrated.

The Fault Detection and Localization Function (FDLF) is introduced, which can load and use at runtime a Fault Localization Model (FLM) prepared according to the methodology presented in clauses 7.2.2.2 to 7.2.2.4. The FDLF might or might not be part of the NFV-MANO. It is however expected to be able to interact with the VIM (and/or possibly other NFV-MANO FEs) to collect (performance) metrics corresponding to the X features vector used to build the FLM. These features at the NFVI level could include CPU usage, memory usage, disk usage, incoming and/or outgoing packets/bytes of the virtual compute or network, etc.

The FLM at this level could be prepared by the operator according to the physical and virtual resources used in their NFVI deployment. Based on this the X features are composed to be used to train the first layer of the 2-layered SOM. While for the Y features, traces for different NSs set up on the same NFVI deployment could be collected and evaluated against their respective SLAs to prepare the training and validation data for the second layer. At runtime according to the X features, the FDLF needs to subscribe for the corresponding resource performance metrics and evaluate the received performance data against the FLM first if they signal anomaly as described in clause 7.2.2.3. If an anomaly has been detected, then the FDLF evaluates further the data as described in clause 7.2.2.4 to identify the potential faulty(s) and reports them to the VIM. The VIM, in turn, can perform further checks and possibly diagnostics to determine the exact severity of the problem and notify the VNFM and/or NFVO as necessary. This use case scenario, i.e. NFVI resource fault localization, is described in the remaining part of clause 7.2.2.6.

While the use case scenario is based on X features collected at the NFVI level, in the NFV context, the methodology could be adapted to the VNF level as well. For example, VNF vendors could prepare models for their VNFs using for X features VNF level virtual resource metrics and for Y features VNF metrics specified in the SLAs for those VNFs. These Y features could include VNF performance metrics and/or application specific metrics reported via the indicator interface.

7.2.2.6.2 Actors and roles

Table 7.2.2.6.2-1 describes the actors and roles of the NFVI resource fault localization use case scenario.

Table 7.2.2.6.2-1: Actors and roles of NFVI resource fault localization

#	Role	Description
1	VIM	VIM managing the virtualised resources of the NS instance.
2	FDLF	Fault Detection and Localization Function using real-time KPIs collected by the VIM to detect and localize faults. It is part of or has direct communication with the VIM, in which case it might not be part of NFV-MANO.

7.2.2.6.3 Pre-conditions

Table 7.2.2.6.3-1 describes the pre-conditions for the NFVI resource fault localization use case scenario.

Table 7.2.2.6.3-1: NFVI resource fault localization pre-conditions

#	Pre-condition	Additional description
1	The resource fault detection and localization model is available to the FDLF	The trained and validated 2-layered SOM models corresponding to the NFVI resources and to be used by the FDLF have been loaded into the FDLF. The FDLF knows the mapping of the required X features and their mapping to the resources managed by the VIM.
2	The VIM and FDLF are operating correctly	

7.2.2.6.4 Post-conditions

Table 7.2.2.6.4-1 describes the post-conditions for the NFVI resource fault localization use case scenario.

Table 7.2.2.6.4-1: NFVI resource fault localization post-conditions

#	Post-condition	Additional description
1	The resource faults potentially causing NS level service degradation have been identified by the FDLF and reported to the VIM	For full remediation, further action might be necessary.

7.2.2.6.5 Flow description

Table 7.2.2.6.5-1 describes the flow for the runtime detection and localization of faults in the NFVI resources.

Table 7.2.2.6.5-1: Flow description for the NFVI resource fault localization use case

#	Actor/Role	Action/Description
Begins when	FDLF	The FDLF has loaded the FLM and identified the resource performance metrics corresponding to the X features of the model.
Step 1	FDLF <-> VIM	The FDLF creates the appropriate PM job(s) for the VIM concerning the identified resource performance metrics, then subscribes with the VIM for the related notifications.
Step 2	FDLF	The FDLF is in a waiting state to receive PM job notifications from the VIM.
Step 3	VIM -> FDLF	Based on the information collected from the NFVI, the VIM notifies the FDLF about the availability of data. Data contained in the notifications may need to be pre-processed (e.g. normalized, smoothed) by the FDLF in order to prepare the X features vector for the FLM.
Step 4	FDLF	The FDLF runs the FLM using the data received from VIM to determine if the system resources are in a healthy condition or there is an anomaly. If no anomaly is detected, the flow returns to Step 2 to continue waiting for subsequent notifications. Otherwise, it continues with Step 5.
Step 5	FDLF	The FDLF runs the fault localization procedure of the FLM to identify the most likely cause(s) of the anomaly.
Ends when	FDLF -> VIM	The FDLF reports to the VIM that an anomaly has been detected and identifies the faulty resources based on the metrics that show the anomaly. The FDLF can return to Step 2 to continue waiting for subsequent notifications.

7.3 Anomaly prediction

7.3.1 Introduction

In the context of cognitive management, NFV systems (i.e. VNFs, NFV-MANO) can significantly benefit from predictive maintenance to avoid service perturbation or disruption. While root cause analysis is used after the occurrence of an event (e.g. to diagnose a failure once it is detected in order to minimize MTTR), proactive management tasks help to predict malfunctions with two different goals:

- extrapolation of current fulfilment level of service level objectives with the intent to mitigate potential breaches; or
- anticipation of failures leading to outages.

The former approach, studied in clause 7.1 related to service availability assurance, evaluates potential impact of failures and other changes on the network service availability, particularly before QoS is impacted. The latter is defined as a way to prevent critical events, i.e. failures in the NFV system that could lead to service degradation or interruption.

Once such events are inferred with a certain risk level, a straightforward action is to provide recommendations to the NFV system manager in order to start prevention. In an automated environment, a control loop may be exploited to launch proactive management actions. In case of manual remediation as well as for automated reaction, avoiding the occurrence of service outages is the objective.

The lead time, i.e. time between the forecast and the anticipated failure event, depends on the context and the problem to be avoided. On one hand, the earlier a failure can be predicted, the better it is in terms of proactive preparedness and efficiency. On the other hand, the prediction accuracy may be inversely proportional to this time: data collected, e.g. 30 minutes before the issue to occur will lead to a more precise prediction than information gathered few hours before it.

As the amount of field data is generally huge in large telecommunication networks, a lot (if not most) of data are not related to specific failure situations and they constitute noise added to relevant information. Thus, once a targeted failure scenario is defined, appropriate operations data (related to distinctive KPIs) have to be identified, collected and prepared for use to train the cognitive model for that known failure situation. If needed (e.g. scarcity of relevant operations data), the use of fault injection techniques, simulating stress behaviour in a controlled environment, may help to add data for the training phase.

The operations phase includes the recognition of the pattern of previously known failure situations when these patterns occur again, and the identification of new failure situation patterns through the clustering approach. Identification of known failure situations can exploit supervised, or semi-supervised, learning techniques whose task is to learn from training datasets and produce an inference function that can be employed for detecting, within defined confidence intervals, severe situations. Identification of new failure situations may rely on unsupervised machine learning which clusters unlabelled datasets by discovering hidden patterns or data groupings without the need for human intervention, thanks to its ability to detect similarities and differences in information.

There is a large variety of data driven cognitive analyses using algorithms and training data to build models in order to perform predictions in new unseen data using the intrinsic relationships learned. Based on neural networks (NNs), *feed-forward* NNs, a class of deep NNs (i.e. 2+ hidden layers NNs) in which the information flows from the input layer to the output layer without internal loops, are capable of modelling complex non-linear relationships by using training data they receive as input. Such NNs can be used in conjunction with recurrent NNs, this latter benefiting from backpropagation algorithms to optimize the neural weights for finding the coefficients that capture the best relationship between past knowledge and future events.

7.3.2 Use of log messages for anomaly prediction

7.3.2.1 Introduction

To forecast anomalies using log messages, gathering them to build a prediction model is a prerequisite.

Logs are produced during the operation of NFV-MANO FEs such as NFVO, VNFM, VIM or WIM. These logs can be processed through logging jobs using the NFV-MANO log management interface. A logging job represents the filtering criteria for processing and creating log reports from the logs generated by the underlying system of the NFV-MANO functional entity. The interface enables the API consumer to subscribe to and notifies about events related to the availability of the log reports.

NOTE: The term NFV-MANO functional entity (or NFV-MANO FE) is used in the present document in the same meaning as in ETSI GS NFV IFA 031 [i.7].

The interface also enables managing different types of filtered logs, which can be grouped, at large, into two categories (see ETSI GS NFV-SOL 009 [i.15]):

- Messaging logs: these are logs of messages exchanged on an interface between NFV-MANO FEs, and between NFV-MANO FEs and external entities. Examples of such logs include logging of the input and output message parameters of interfaces exposed by the FEs, e.g. input and output messages when NFVO queries the *InstantiateVnf* operation at the VNF LCM interface.

- Provider-specific logs of NFV-MANO FEs which may have security restrictions in place, e.g. encrypted logs.

Conditions and criteria determine when the logs are compiled and when the producer reports about their availability based on:

i) log size;

ii) time information (e.g. every 24 hours);

iii) events such as explicit stop of the logging job, threshold reached, etc.

Typically, the collection of large training sets of data that contain enough information on the different anomalies is not conceivable since anomalies are infrequent and divergent. As a result, an approach of training exclusively based on the exploitation of logs related to normal events ("normal" log messages) is preferred (see clause 7.3.2.2). Such an approach allows for detecting any divergence from the behaviour perceived as normal by the models that may lead to some anomaly; however it is not suitable to predict the actual nature of predicted anomalies. The runtime use of such anomaly prediction models is described in clause 7.3.2.3.

It is noteworthy that besides the logs of NFV-MANO functional entities as sketched above, the logs from diverse subsystems (hardware, VMs, etc.), as well as the ones produced by VNFs themselves, may also be beneficial as input data for anomaly detection using different cognitive approaches. One such example for VNFs is discussed in clause D.1.

7.3.2.2 Training based on the use of "normal" log messages

7.3.2.2.1 Rationale

During the operations of certain VNFs, it has been observed that log messages (e.g. syslogs) related to anomalies often occur before related *trouble tickets* are generated. Usually, trouble tickets are generated in response to signals from various underlying network monitoring systems matching these signals against known problem signatures, via a series of ticket processing logic (e.g. event correlation). Thus, the ticket report time is often at, or after, the first occurrence of a symptom of the network fault. These symptoms may, or may not, be visible right away to the VNFs reporting them as log messages. Thus, the correlation between the logged symptoms and ticket generation is imperfect: it may happen that no symptoms occurred with some tickets, while some symptoms are delayed in reaching the VNFs, i.e. they follow the generation of a ticket.

Empirical studies were performed to filter through these log messages related to anomalies to identify potential early warning signals or predictive signatures which could enable fast, or even proactive, actions against faulty conditions. A runtime predictive analysis system can thus be built, running in parallel with existing reactive monitoring systems to provide network operators timely warnings against such faulty conditions.

In situations where failures data are rare, it may be difficult to train a supervised learning model for fault ticket prediction as little or no training data are available about faults. A solution is to train a deep learning model exclusively with logs related to normal events, i.e. it learns log message patterns of normal operations. Abnormal log patterns, i.e. deviations from the norm, flagged by the model, then may serve as indicators of potential network trouble events. If such prediction is possible, this would allow operators or closed-loop automation to trigger mitigation actions prior to such an event and help minimize its impact.

As a reporting means of the system towards users/programmes, log messages display sequential patterns that an accurate model of log messages should be able to capture. Therefore, examples of techniques that can be used for such anomaly detection approach include autoencoder, one-class Support Vector Machine (SVM), and Long Short-Term Memory (LSTM):

- Autoencoder is a feed-forward multi-layer neural network in which the desired output is the input itself; after training the autoencoder with normal data, the reconstruction error can be used as an anomaly indicator.
- One-class SVM uses shallow learning to build a model of the "normal" log message training data; if a new log message entry deviates significantly from the model, it is marked as anomaly.
- LSTM network is a special case of recurrent neural networks which is equipped with explicit memory cells that have the ability to remember long-term dependencies over sequences.

Because it has the capability of capturing the comprehensive and intricate patterns embedded in sequential data, LSTM network will be considered in what follows. An illustration of results obtained by such approach [i.16] is summarized in clause E.1.

7.3.2.2.2 Data preparation and model training

The procedure described in the present approach (i.e. use of "normal" log messages) is based on the existence of a set of data (e.g. covering several months of field operations) which include:

- log messages; and
- thoroughly analysed trouble tickets.

Existing field log messages contain both events encountered during normal operations ("normal" log messages) and events related to the creation of trouble tickets ("abnormal" log messages). In order to remove log messages related to anomalies from the data used for the model training, a possibility is to discard entries located within a certain interval (e.g. X hours/days) around the active period of each ticket, as the active period is between a ticket's arrival time and when it is marked as resolved. After log message entries occurring within a X-buffer around the active window of actual tickets are pruned, the LSTM network is trained with log messages produced during "ticket-free" network operations.

In addition, care needs to be taken of minority log patterns, i.e. not related to trouble tickets, but reflecting infrequent normal behaviour. Similar to fault patterns, these are also hard to learn given their rare appearances in the training data. This may result in a high false alarm rate because they would be considered as "abnormal" log messages by the trained model.

A solution is to use oversampling for such minority patterns, e.g.:

- after a round of training (initial training, as well as training refinement described in the next clause), test the model by identifying "normal" log message patterns that are misclassified as anomalies;
- oversample these patterns and randomly sample all other patterns;
- use the resulting data to adjust the model weights;
- exit the process when the false positive rate cannot be improved further.

Building the LSTM model starts with a training phase. This needs to be followed up by a validation and fine-tuning phase of the model before it can be used for prediction. To enable model validation, different data sets are necessary, hence the set of log messages is split to different subsets to be used for model training and model validation. After pruning as described above, the first subset is used for the training of the LSTM model.

7.3.2.2.3 Training refinement and validation

Following the training using the first subset of data, the LSTM model is run against the second subset of data available with the purpose of model refinement and optimization.

To determine the efficiency of the anomalies detection, "abnormal" log messages identified by the trained model are validated through mapping these "abnormal" log messages to relevant trouble tickets. Anomalies which cannot be associated with any tickets are considered as *false positive/alarms*. Tickets to which no "abnormal" log message is mapped are considered missed faults. These classification criteria will be used to measure the model performance (see clause D.1).

Log messages associated with a ticket may fall into the active period of the ticked or beyond. Those that precede the active period allow for anomalies prediction. For these, through different trials, a unique time window ahead of the ticket generation is obtained, which is defined as the *predictive period*.

An anomaly detected during the predictive or active period is thus associated to a ticket. The former may be treated as an *early warning signal* (or *ticket-triggering signature* - note that one ticket can possibly have multiple (early) signatures), while the latter is considered as a *post event symptom*.

A final tuning is to identify the minimal statistical set needed for declaring reliable signatures of upcoming trouble tickets. After matching log messages related to anomalies with non-duplicated tickets, each ticket is associated with, e.g. at least two anomalies (in the predictive period) which are close to each other. A detection system can thus be configured to report a warning signature for network trouble tickets upon detecting a small cluster of two or more anomalies.

7.3.2.2.4 Continuous learning

Due to changes in the system (e.g. upgrades, reconfigurations), a trained model can become obsolete over time. To avoid rapid obsolescence, a rolling collection and use of data is conceivable. For example, "normal" log messages of a given month can be exploited to train an LSTM model that will be used to detect anomalies during the next month. That is, the log messages and the trouble tickets of a given month are processed as described in clause 7.3.2.2.2 Then this new set of data is used to update the trained model, which in turn is used to detect/predict anomalies of the next month. Thus, the training is realized in multiple rounds, e.g. use month *i* data to train the model and predict month (*i+1*) anomalies, then using month (*i+1*) data to update the model and predict month (*i+2*) anomalies, etc.

After each training (e.g. using the corresponding month's data), the anomalies detection efficiency is checked as described in clause 7.3.2.2.3. The training refinement then consists in pruning the newly used log messages and add them to the training data to feed the model as described in the previous clauses. If applicable, oversampling minority log patterns is also realized.

7.3.2.3 Runtime use of anomaly prediction models using log messages

7.3.2.3.1 Introduction

Clause 7.3.2.2 has described the use of log messages to train models for providing early warning for potential issues in the network that could result in trouble tickets. To illustrate the runtime use of these models, clauses 7.3.2.3.2 to 7.3.2.3.5 describe the way such prediction framework can interact with a NFV system. The use case exploits logs produced by NFV-MANO FEs, i.e. NFVO, VNFM and VIM during a network service lifetime. As described in clause 7.3.2.2, the models on which the prediction is based were trained and validated using, e.g. laboratory data issued from a digital twin network running this network service. The focus of the prediction is to inform in advance the operations teams that network trouble tickets may occur, leaving them enough time to prepare remediation.

In order to continuously enhance the prediction accuracy, feedback from past predictions is needed. To this end, once the prediction horizon is over, the data used for this task are processed, e.g. the logs, the confirmation that the prediction was correct (see clause 7.3.2.2 for more details). The prediction models are updated consequently through training and replace the previously used ones.

7.3.2.3.2 Actors and roles

Table 7.3.2.3.2-1 describes the anomaly prediction use case actors and roles.

Table 7.3.2.3.2-1: Actors and roles of anomaly prediction using log messages

#	Role	Description
1	NFVO	NFV Orchestrator managing NS instances.
2	VNFM	VNF Manager managing the VNF instances of NS instances.
3	VIM	VIM managing the virtualised resources of NS instances.
4	OSS	OSS in charge of NS instances.
5	APF	Anomaly Prediction Function using log messages issued from NFV-MANO in order to predict anomalies. It could be part, or not, of NFV-MANO.

7.3.2.3.3 Pre-conditions

Table 7.3.2.3.3-1 describes the anomaly prediction use case pre-conditions.

Table 7.3.2.3.3-1: Anomaly prediction using log messages pre-conditions

#	Pre-condition	Additional description
1	The anomaly prediction models have been onboarded and are available	The models on which APF is based have been built, i.e. trained and validated, using real or simulated data (see clause 7.3.2.2).
2	The NFVO, VNFM, VIM, OSS and APF are operating correctly	

7.3.2.3.4 Post-conditions

Table 7.3.2.3.4-1 describes the anomaly prediction use case post-conditions.

Table 7.3.2.3.4-1: Anomaly prediction using log messages post-conditions

#	Post-condition	Additional description
1	Potential for anomalies has been reported to the NFVO or OSS	

7.3.2.3.5 Flow description

Tables 7.3.2.3.5-1, 7.3.2.3.5-2 and 7.3.2.3.5-3 describe respectively the flows for the Anomaly Prediction Function initiation, for the anomaly prediction using log messages, and for the periodic Anomaly Prediction Function models update.

Table 7.3.2.3.5-1: Flow description of the APF initiation

#	Actor/Role	Action/Description
Begins when	NFVO	NFVO decides to initiate the APF.
Step 1	NFVO -> APF	NFVO requests the APF to start the process.
Step 2	APF	The APF loads the onboarded prediction models indicated by the NFVO and determines the input needed to use these models for prediction.
Step 3	APF <-> NFVO	Through the NFV-MANO log management operation "Create Logging Job" (see Table A.3.2-1), the APF defines its request towards NFVO~~.~~ It also subscribes for relevant log management notifications.
Step 4	APF <-> VNFM	Through the NFV-MANO log management operation "Create Logging Job" (see Table A.3.2-1), the APF defines its request towards VNFM. It also subscribes for relevant log management notifications.
Step 5	APF <-> VIM	Through the NFV-MANO log management operation "Create Logging Job" (see Table A.3.2-1), the APF defines its request towards VIM. It also subscribes for relevant log management notifications.
Step 6	APF	Initialization data are received by the APF from NFV-MANO FEs.
Ends when	APF -> NFVO	APF informs the NFVO that it is now active.

Table 7.3.2.3.5-2: Flow description of the anomaly prediction using log messages

#	Actor/Role	Action/Description
Begins when	APF	The steps of this flow are repeated periodically while APF is active.
Step 1	NFVO, VNFM, VIM -> APF	The APF may receive one or more notifications related to the subscriptions initialized in the initiation flow. Data contained in these notifications may need to be pre-processed by the APF in order to get the information needed for the prediction models.
Step 2	APF	The APF runs the prediction models using real-time data received from NFVO, VNFM, and VIM. The models check if there is a potential for an anomaly, i.e. occurrence of a network trouble ticket, within the defined time horizon.
Ends when	APF -> NFVO or OSS	If a potential for an anomaly is predicted by the models, the APF notifies NFVO or OSS.

Table 7.3.2.3.5-3: Flow description of the periodic APF models update

#	Actor/Role	Action/Description
Begins when	APF	APF is informed that updated models are available.
Step 1	APF	APF loads the updated models (see note).
Ends when	APF	APF continues with the new models.
NOTE: Data collection/processing and models retraining, as described in clauses 7.3.2.2.2 to 7.3.2.2.4, are out of scope of the present study.		

7.3.3 Use of KPIs for anomaly prediction

7.3.3.1 Rationale

Key Performance Indicators (KPIs) are metrics which have been used to gauge the functioning of network functions in the well-established physical environment (i.e. PNFs), and can be exploited in the same vein in the virtualised one (i.e. VNFs). KPIs are directly collected by - or computed using values obtained from - different elements of the network at discrete instants of time. They express specific quantitative characteristics, and usually act as measurable benchmarks against defined goals. Examples of KPIs for an IMS system include CPU utilization, calls attempt rate and registrations attempt rate.

As real time metrics characterize the network behaviour, flawless situations, as well as anomalies, can be observed through KPIs. Since anomalies (e.g. VNF overload) are unusual events which may lead to service unavailability, their prediction - as early as possible - by using KPIs is thus of great interest for network/service providers. Such proactive anomalies detection can help to diagnose and correct potential issues before they happen.

To this end, an approach described in [i.17] consists of exploiting the correlation among different KPIs to build a model which predicts the temporal variation of a given KPI, i.e. *forecasted* KPI, based on the observation of other KPIs considered as *predictors*. This approach postulates that there is a correlation between KPIs. In an IMS system, for instance, an increasing trend of the calls attempt rate and the registrations attempt rate (predictors) impact the total CPU load, which in turn impacts the CPU utilization per VNFC instance (forecasted KPI), thus leading to, e.g. a need for automatic scaling out. If such measure is not possible due to, e.g. the resource limitation is reached, an overload may occur. KPIs data are initially discrete values obtained from network operations and can be exploited as such, or they can be represented by functions with the use of a smoothing technique.

In the first case, *linear regression* is a way to analyse data with one measurement per KPI, while data recorded on regular time intervals, i.e. KPIs considered as time series, need approaches such as *distributed-lag regression*, i.e. predicting a forecasted variable based on the past/lagged values of the predictors.

In the second case, a principal component analysis is applied to the functional data in order to obtain the main modes of variation for the KPIs. A classification model is then used to predict anomalies. To this end, simple models such as *logistic regression* can be exploited, but machine learning based classification approaches, e.g. *random forest*, are also applicable. An illustration of results obtained by such approach is summarized in clause D.2.

The different ways to build KPIs-based prediction models, as sketched above, need sufficient amount of data available for training and validating the models. This training phase is generally done offline. Once the first versions of the models are ready, these are deployed and used in the field. The simplest way to update these models with new data taking into account, e.g. new network configurations/environments, is to do it offline, although embedding this update task in the online process of the prediction function can be envisaged at the cost of adding complexity to the picture.

7.3.3.2 Use of discrete data

7.3.3.2.1 Linear regression

As indicated in clause 7.3.3.1, a forecasted KPI Z is predicted with the use of other KPIs (predictors) X = (x_1; …; x_p). The expression of the regression linking Z to the p KPIs, i.e. column vector X, in a given prediction horizon h can be written as equation (14):

$$z_{i+h} = \delta + \ \sum_{j=1}^{P} k_j\, x_{j,i} + \varepsilon_{i+h} \tag{14}$$

where i : observation time index:

$x_{j,i}$: ith observation of the jth KPI;

ε_{i+h} : model error, which is to be minimized for h;

θ_h = (δ; k_1; ... ; k_p) is the column vector of the model parameters for prediction horizon h.

For another time instant i', the prediction for horizon h is calculated by (15):

$$\hat{z}_{i'+h} = \hat{\delta} + \sum_{j=1}^{P} \hat{k}_j\, x_{i',j} \tag{15}$$

where $\widehat{\theta_h}$ = ($\hat{\delta}$; $\hat{k}_1$; ...; $\hat{k}_p$), representing the trained model, is the estimated column vector of the model parameters using a least square estimator at the training phase. Note that there is a column vector $\widehat{\theta_h}$ for each value h, the subscript "h" is not shown for the related vector entries for readability purposes, i.e. all formulas are based on a certain value of "h".

The anomaly of the forecasted KPI is predicted by comparing $\hat{z}_{i'+h}$ to an appropriate threshold for this forecasted KPI, e.g. an anomaly is predicted if it is higher than this threshold.

Predicting KPI Z provides good results if it is strongly correlated with the p predictor KPIs X. The prediction is obviously sensitive to the value of the prediction horizon h: as h increases, the correlation decreases. This linear regression model is thus valid when the variables are strongly correlated and for a near prediction horizon.

7.3.3.2.2 Distributed-lag regression

In the case where the variables are not strongly correlated or for a longer prediction horizon, it may be judicious to consider a series of observations per KPI and perform the calculation based on the variation trend of these measures. The anomaly prediction is then executed with the use of a set of measurements collected during a time window, e.g. 24 hours with a frequency of 15 min (i.e. a total of 96 KPI values per day). In this case, a distributed-lag regression model can be exploited: the method thus predicts a forecasted variable based on the past/lagged values of the predictors X = ($X_{1,}$; …; $X_{p,}$;) = ($x_{1,1,}$; …; $x_{1,m-1}$; $x_{2,1,}$; … $x_{2,m-1,}$; … ; $x_{p,1,}$;…; $x_{p,m-1}$) (16):

$$z_{i+h} = \delta + \sum_{j=1}^{P} \sum_{l=1}^{m-1} k_{j,l}\, x_{j,i-l} + \varepsilon_{i+h} \tag{16}$$

where θ_h = (δ; $\kappa_{1,0}$; ...; $\kappa_{1,m-1}$; ... ; $\kappa_{p,0}$; …; $\kappa_{p,m-1}$) is the column vector of the model parameters.

If h > 0, a future anomaly is predicted using the m previous values.

For another time instant i', the prediction of Z (17) is:

$$\hat{z}_{i'+h} = \hat{\delta} + \sum_{j=1}^{P} \sum_{l=0}^{m-1} \hat{k}_{j,l}\, x_{j,i'-l} \tag{17}$$

where $\hat{\theta}$ = (δ; $\hat{k}_{1,0}$; ...; $\hat{k}_{1,m-1}$; ... ; $\hat{k}_{p,0}$; …; $\hat{k}_{p,m-1}$) is the estimated column vector of the model parameters using a least square estimator at the training phase.

As for the linear regression model, the anomaly of the forecasted KPI is predicted by comparing $\hat{z}_{i'+h}$ to a threshold. This method catches more correlation between the variable of interest and the predictors as several measurements per KPI are used. However, the measurements within each KPI can be noisy. The addition of the corresponding noise terms in the present distributed-lag regression model may lead to the degradation in the prediction performance.

7.3.3.3 Use of functional data

7.3.3.3.1 Logistic regression

To cope with the noise issue mentioned in clause 7.3.3.2.2, the KPI discrete values can be considered as functional data in order to find the latent relationship between the variables and reduce the amount of noise. A smoothing technique, e.g. B-spline, is thus used to build a functional form from the KPI discrete observations. In addition to the enhancement of noisy observations, such technique can deal with missing data. Furthermore, thanks to a principal component analysis, the linear combination between the variables expressing their modes of variation can be displayed. Using the resulting data, a logistic regression model is finally trained and validated for anomaly prediction.

Functional data X = (X_1(t); …; X_p(t)), t ∈ [0,T] created from the p KPIs which are collected at regular time steps constitute the observation of a stochastic process during a continuous time interval T. Each observation at time instant i is a set of p functions. The functions of each KPI are obtained based on the collected samples of the corresponding KPI during the time window T. For the KPI of interest Z, a label Y ∈ {0,1} is created by comparing its values to the threshold, where Y = 1 indicates that there is an anomaly in the prediction horizon h while Y = 0 indicates that the future behaviour of the network is normal at h. The different steps of such model are as follows:

- The noisy KPI observations collected from the network, which may contain missing values, are transformed into functional data using a smoothing technique: each observation x_i is then described by a set of p functions or curves - the functions are functional estimates that describe the (e.g. daily) evolution of all KPIs.

- A Functional Principal Component Analysis (FPCA) is performed next, allowing to optimize the representation of the functional data by computing their principal components - all the observations (training and test) are projected on the same smoothing basis and FPCA space.

- The last step consists in using a proportion of the labelled dataset (training observations) to develop the training model with the help of a logistic regression method for functional data. This simplifies the prediction model by considering regression which presents a much lower complexity compared to machine learning classification methods (such as random forest - see clause 7.3.3.3.2), especially when the number of observations and/or variables is great. At the inference phase, the new observation, after being transformed with smoothing and FPCA, is used by the training model to predict a possible future anomaly in the prediction horizon.

7.3.3.3.2 Random forest

Random Forest (RF) is a tree-based supervised learning method which can be used for prediction. As part of ensemble learning, RF averages the results of a great number (e.g. hundreds, thousands) of de-correlated decision trees for this purpose. This approach, used to reduce variance within a noisy dataset, also mitigates the risk of over-fitting/over-generalization of individual trees to training data (i.e. cancelling out the biases inherent in "deep" decision trees). Usually from 70 % to 90 % of the available dataset is used as training data from which random samples are drawn. The rest is used as validation data (called *out-of-bag sample* and used for cross-validation). Each individual tree is trained by using:

- a random sample with replacement of the training data set (technique known as *bootstrap aggregation* or *tree bagging*); and

- a random subset of input features/variables selected at each split point in the construction of the tree (approach called *feature sampling/randomness/bagging*). The size of the random subset (e.g. 30 %) is a hyper parameter as mentioned below.

Thanks to feature sampling, RF can handle missing values by using median values to replace continuous variables, or by computing the proximity-weighted average of missing values. Three hyper parameters are needed to run RF:

- number of trees, i.e. the number of estimators. At the price of slowing down the computation, a higher number of trees increases the performance and makes the predictions more stable, i.e. when multiple decision trees form an ensemble in the RF algorithm, they predict more accurate results, particularly when the individual trees are uncorrelated with each other;
- size of the random subset used to split a node;
- node size or tree size/depth.

To measure/check the accuracy of RF, it is common to compare the actual and the predicted values/outcome for the validation data, and to use techniques such as mean square error reduction. Needless to say, the accuracy increases with the increase of the number of trees in the forest and the use of relevant features.

It is noteworthy that, based on this RF accuracy measurement, RF can be used for feature selection, i.e. selecting the most important features out of the available features. With the use of the *variable importance* or *contribution* measure, one can identify the most significant features, i.e. the relative importance of features and their contribution to the model. The process consists of removing features one at a time, training the algorithm, and verifying how the prediction accuracy changes. If there is no significant change, the removed feature can be omitted permanently.

Besides its low interpretability (e.g. as compared to a single decision tree), the main disadvantage of RF is its computational complexity slowing the prediction process when a large number of decision trees is used, all of which being run for the same input, i.e. a time-consuming process. More details about RF can be found in [i.25].

7.3.3.4 Runtime use of anomaly prediction models based on KPIs

7.3.3.4.1 Introduction

Clauses 7.3.3.2 and 7.3.3.3 have described the use of runtime discrete KPI data or its derivate functional one for predicting anomalies in a defined time horizon. This prediction process is based on models built based on data collected in the past in a comparable context.

To illustrate the runtime use of these models, clauses 7.3.3.4.2 to 7.3.3.4.5 rely on an IMS use case. The network service (i.e. IMS) is composed of several VNFs, and the overload prediction of one of them, say VNF_i, is the target of the study. As different causes can lead to the VNF_i overload, the surveillance of different KPIs such as CPU utilization of its VNFCs is necessary.

NOTE: It is noteworthy that for a pool of redundant VNFC instances, the mean CPU utilization value of the pool needs to be computed following each reception of individual VNFC CPU data.

As the maximum number of VNFC instances (see VduProfile in Table A.7.4-2) can be limited, overload can occur if this maximum value and the CPU utilization threshold are both reached simultaneously. It is thus helpful to anticipate such coincidence to be able to avoid undesirable VNF_i overload due to VNFC scaling out limitation.

In order to predict such undesirable coincidences in a certain time horizon, the past evolution of the CPU utilization KPI, scaling, and of other correlated KPIs is needed. Examples of other KPIs within the IMS system if VNF_i represents the Call Session Control Function (CSCF) include the current number of calls, and the current number of registrations. Based on this information, the prediction model(s) is/are built for the IMS NS and subsequently used at runtime to analyse real-time data.

7.3.3.4.2 Actors and roles

Table 7.3.3.4.2-1 describes the VNF_i overload prediction use case actors and roles.

Table 7.3.3.4.2-1: Actors and roles of anomaly prediction using KPIs

#	Role	Description
1	NFVO	NFV Orchestrator managing the NS instance. VNF$_i$ is part of the VNFs composing this NS.
2	VNFM	VNF Manager managing the VNF instances of the NS instance, including the instances of VNF$_i$.
3	VIM	VIM managing the virtualised resources of the NS instance.
4	EM	Element manager responsible for managing the VNF instances of the NS instance, including the instances of VNF$_i$.
5	OSS	OSS in charge of the NS instance.
6	APF	Anomaly Prediction Function using real-time KPIs issued from NFV-MANO and/or EM in order to predict anomalies. It could be part, or not, of NFV-MANO.

7.3.3.4.3 Pre-conditions

Table 7.3.3.4.3-1 describes the VNF$_i$ overload prediction use case pre-conditions.

Table 7.3.3.4.3-1: Anomaly prediction using KPIs pre-conditions

#	Pre-condition	Additional description
1	The NS has been onboarded and it is available.	The models on which APF is based have been built, i.e. trained and validated, using real or simulated data (see clause 7.3.3.4.1). Within this use case, these models cover at least the VNF$_i$ of the NS. The models are onboarded with the NS artefacts.
2	The NFVO, VNFM, VIM, EM, OSS and APF are operating correctly.	
3	The NS has been instantiated and provides its services.	

7.3.3.4.4 Post-conditions

Table 7.3.3.4.4-1 describes the VNF$_i$ overload prediction use case post-conditions.

Table 7.3.3.4.4-1: Anomaly prediction using KPIs post-conditions

#	Post-condition	Additional description
1	Undesirable overload situations have been reported for remediation.	

7.3.3.4.5 Flow description

Tables 7.3.3.4.5-1 and 7.3.3.4.5-2 describe respectively the flows for the Anomaly Prediction Function initiation and for anomaly prediction using KPIs.

Table 7.3.3.4.5-1: Flow description of the APF initiation

#	Actor/Role	Action/Description
Begins when	NFVO -> APF	NFVO decides to initiate the APF.
Step 1	NFVO -> APF	NFVO requests the APF to start the process within the scope of VNF$_i$ and a particular VNFC$_j$ using the onboarded prediction models.
Step 2	APF	The APF loads the onboarded prediction models indicated by the NFVO and determines the input needed to use these models for prediction.
Step 3	APF <-> NFVO	Through the "Query VNF Package Info" operation (see Table A.6.6-1), the APF gets the maximum number of authorized VNFC$_j$ instances (VduProfile in Table A.7.4-2) for the related VNFi deployment flavour. It also subscribes to events that can change this information.
Step 4	APF <-> VNFM	Through the "Query VNF" operation (see Table A.6.5-1), the APF gets information such as the current number of VNFC$_j$ instances and their number for the scale level. It also subscribes to any further change of such information.

#	Actor/Role	Action/Description
Step 5	APF <-> VIM	Through the performance management operation "Create PM Job" (see Table A.2.2-1), the APF defines its request for VNFC$_j$ CPU utilization, together with the appropriate collection period. It also subscribes for relevant performance management notifications (see note).
Step 6	APF <-> EM	The APF gets from the EM any necessary information for the analysis, e.g. current number of calls, current number of registrations, configuration data.
Ends when	APF -> NFVO	APF informs the NFVO that it is now active.
NOTE: For a pool of redundant VNFCj instances, the mean CPU value of the pool has to be computed, e.g. by the APF, following the reception of individual VNFCj CPU data.		

Table 7.3.3.4.5-2: Flow description of the anomaly prediction using KPIs

#	Actor/Role	Action/Description
Begins when	APF	The steps of this flow are repeated periodically while APF is active.
Step 1	NFVO, VNFM, VIM -> APF	The APF may receive one or more notifications related to the subscriptions initialized in the initiation flow. Data contained in these notifications may need to be pre-processed (e.g. mean/average calculation) by the APF in order to get the information needed for the prediction models.
Step 2	APF <-> EM	The APF gets from the EM any necessary information for the analysis, e.g. current number of calls, current number of registrations, configuration data.
Step 3	APF	The APF runs the prediction models using real-time/updated data received from the NFVO, VNFM, VIM and EM. The models check if a potential anomaly can occur in the defined time horizon.
Ends when	APF -> NFVO or OSS	If a potential anomaly is predicted by the models, the APF notifies the NFVO or the OSS.

8 Recommendations

8.1 Introduction

This clause provides recommendations for NFV-MANO which have been derived from the use cases discussed in clause 7. The recommendations are made from a reliability point of view.

The following terminology is used:

- "It is recommended that a requirement be specified" means that the recommendation should be addressed in subsequent specifications by creating requirements using the auxiliary "shall".
- "It is recommended that" means that the recommendation should be addressed in subsequent specifications by creating recommendations using the auxiliary "should".

8.2 Recommendations related to service availability assurance

The term "Service Availability Assurance Function" is used for the purpose to describe the functionality and to address the entity that is providing that functionality. No assumption is made about what entity or entities can play such a role. This functionality may be a standalone function or may be part of another function, for example Management Data Analytics Function.

Table 8.2-1 provides recommendations related to service availability assurance.

Table 8.2-1: Recommendations related to service availability assurance

Identifier	Recommendation description	Use case reference
Saa.001	It is recommended that a Service Availability Assurance Function is provided to support the fulfilment of availability expectations related to NS instances.	Clause 7.1
Saa.002	It is recommended that a requirement be specified that the Service Availability Assurance Function is capable of executing a (composite) model to evaluate the actual availability characteristics of an NS instance against its availability expectations and determine if configuration adjustments are necessary.	Clause 7.1.3
Saa.003	It is recommended that a requirement be specified to provide ways to describe the input necessary for the Service Availability Assurance Function to evaluate the actual availability characteristics of entities an NS instance depends on (see note 1).	Clause 7.1.3
Saa.004	It is recommended that a requirement be specified that the Service Availability Assurance Function has the capability of collecting the input necessary to evaluate the actual availability characteristics of entities an NS instance depends on (see notes 2 and 3).	Clause 7.1.3
Saa.005	It is recommended that a requirement be specified that the Service Availability Assurance Function has the capability of evaluating on demand (e.g. when a notification is received) the actual availability characteristics of entities an NS instance depends on.	Clause 7.1.3
Saa.006	It is recommended that a requirement be specified that the Service Availability Assurance Function has the capability of evaluating periodically the actual availability characteristics of entities an NS instance depends on.	Clause 7.1.3
Saa.007	It is recommended that a requirement be specified that the Service Availability Assurance Function has the capability of initiating configuration adjustments when necessary to ensure fulfilment of availability related expectations.	Clause 7.1.3
Saa.008	It is recommended that the Service Availability Assurance Function is provided distributedly according to the levels (i.e. VIM for the NFVI, VNFM for the VNFs, NFVO for the NSs) of the NFV-MANO functional entities.	Clause 7.1.3
Saa.009	It is recommended that the Service Availability Assurance Function is capable of interacting with the NFV-MANO functional entity of an individual level to collect the information necessary about entities managed by the NFV-MANO functional entity.	Clause 7.1.3
Saa.010	It is recommended that the Service Availability Assurance Function is capable of interacting with the NFV-MANO functional entity of an individual level to initiate changes to entities managed by the NFV-MANO functional entity.	Clause 7.1.3
Saa.011	It is recommended that a requirement be specified to provide the means for the NFVO to forward availability related expectations to Service Availability Assurance Function, when Service Availability Assurance Function is not part of NFVO.	Clause 7.1
Saa.012	It is recommended that a requirement be specified to provide the means for the Service Availability Assurance Function to receive availability related expectations for an NS.	Clause 7.1
Saa.013	It is recommended that a requirement be specified to provide the means to notify when availability related expectations of an NS instance cannot be fulfilled.	Clause 7.1.3
Saa.014	It is recommended to provide the means to receive availability related expectations for entities managed by the NFV-MANO functional entities of their levels.	Clause 7.1.3
Saa.015	It is recommended to provide the means to report on the fulfilment of availability related expectations of entities managed by the NFV-MANO functional entities of their levels.	Clause 7.1.3
Saa.016	It is recommended to provide the means to notify when availability related expectations cannot be fulfilled for entities managed by the NFV-MANO functional entities of individual levels.	Clause 7.1.3

NOTE 1: The description of the input necessary for the Service Availability Assurance function includes the parameters to be monitored in the system.

NOTE 2: The capability of collecting input can mean the capability of subscribing for and receiving notifications about current state, state changes, scaling, etc. as well as collecting configuration and other availability related information.

NOTE 3: The entities related to an NS instance are nested NS, VNF, PNF and VL instances composing the NS and the resources supporting them.

8.3 Recommendations related to RCA

The term "Fault Detection and Localization Function" is used for the purpose to describe the functionality and to address the entity that is providing that functionality. No assumption is made about what entity or entities can play such a role. This functionality may be a standalone function or may be part of another function, for example Management Data Analytics Function.

Table 8.3-1 provides general recommendations related to RCA.

Table 8.3-1: Recommendations related to root cause analysis

Identifier	Recommendation description	Use case reference
Rca.001	It is recommended that a Fault Detection and Localization Function is provided to support the detection and localization of faults that can cause degradation in provided network services.	Clause 7.2
Rca.002	It is recommended that a requirement be specified that the Fault Detection and Localization Function is capable of loading a model associated with virtual and physical resources to be evaluated for their health.	Clause 7.2
Rca.003	It is recommended that a requirement be specified that the Fault Detection and Localization Function is capable of executing a model, which was built to evaluate the health of virtual and physical resources.	Clause 7.2
Rca.004	It is recommended that a requirement be specified that the Fault Detection and Localization Function is capable of identifying unhealthy resources using a model.	Clause 7.2
Rca.005	It is recommended that a requirement be specified to provide ways to describe the input necessary for the Fault Detection and Localization Function to evaluate the health of resources and identify those that are unhealthy (see note 1).	Clause 7.2
Rca.006	It is recommended that a requirement be specified that the Fault Detection and Localization Function has the capability of collecting the input necessary to evaluate the health of resources and identify those that are unhealthy (see note 2).	Clause 7.2
Rca.007	It is recommended that a requirement be specified that the Fault Detection and Localization Function has the capability of evaluating on demand (e.g. when a notification is received) the health of resources.	Clause 7.2
Rca.008	It is recommended that a requirement be specified that the Fault Detection and Localization Function is capable of reporting any identified unhealthy resources to the NFV-MANO.	Clause 7.2
Rca.009	It is recommended that the Fault Detection and Localization Function is provided distributedly according to the levels (e.g. the VIM, the VNFM) of the NFV-MANO.	Clause 7.2
Rca.010	It is recommended that the Fault Detection and Localization Function is capable of interacting with the NFV-MANO functional entity of an individual level to collect the information necessary to evaluate the entities managed by this NFV-MANO functional entity (see note 2).	Clause 7.2
Rca.011	It is recommended that the Fault Detection and Localization Function is capable of reporting to the NFV-MANO functional entity of an individual level a fault detected on any of the entities managed by this NFV-MANO functional entity.	Clause 7.2
NOTE 1: The description of the input necessary for the Fault Detection and Localization Function includes the parameters to be monitored in the system. NOTE 2: The capability of collecting input can mean the capability of subscribing for and receiving notifications about performance metrics at the VIM and/or VNFM level, and about VNF indicators as well as collecting configuration related information for resource identification.		

8.4 Recommendations related to anomaly prediction

The term "Anomaly prediction Function" is used for the purpose to describe the functionality and to address the entity that is providing that functionality. No assumption is made about what entity or entities can play such a role. This functionality may be a standalone function or may be part of another function, for example Management Data Analytics Function.

Table 8.4-1 provides general recommendations related to anomaly prediction.

Table 8.4-1: General recommendations related to anomaly prediction

Identifier	Recommendation description	Use case reference
Apf.001	It is recommended that an Anomaly Prediction Function is provided to support the forecast of anomalies which occur during the functioning of the managed network.	Clauses 7.3.2.3 and 7.3.3.4
Apf.002	It is recommended that a single Anomaly Prediction Function is capable of anomaly predictions based on multiple approaches and technologies.	Clauses 7.3.2.3 and 7.3.3.4

Table 8.4-2 provides recommendations related to anomaly prediction based on log management notifications.

Table 8.4-2: Recommendations related to anomaly prediction based on log management notifications

Identifier	Recommendation description	Use case reference
Lap.001	It is recommended that the Anomaly Prediction Function is capable of executing a model using the content of logs to provide early warnings for potential issues in the network managed by NFV-MANO.	Clause 7.3.2.3
Lap.002	It is recommended that the Anomaly Prediction Function has the capability to register with all NFV-MANO functional entities to receive relevant log management notifications for logs that are created by these NFV-MANO functional entities.	Clause 7.3.2.3
Lap.003	It is recommended that the Anomaly Prediction Function is capable of replacing the model used through training refinement (i.e. retraining) to enhance prediction accuracy.	Clause 7.3.2.3
Lap.004	It is recommended that the Anomaly Prediction Function has the capability of collecting the input necessary to run the forecast model (see note).	Clause 7.3.2.3
NOTE: The capability of collecting input can mean the capability of collecting configuration and other availability related information, as well as subscribing for and receiving notifications about current state, state changes, scaling, etc.		

Table 8.4-3 provides recommendations related to anomaly prediction based on KPIs.

Table 8.4-3: Recommendations related to anomaly prediction based on KPIs

Identifier	Recommendation description	Use case reference
Kap.001	It is recommended that an Anomaly Prediction Function is provided that uses real-time KPIs issued from the managed network in order to predict anomalies.	Clause 7.3.3.4
Kap.002	It is recommended that the Anomaly Prediction Function is capable of executing a model to forecast the temporal variation of a given KPI which characterizes anomalies occurrence.	Clause 7.3.3.4
Kap.003	It is recommended that the Anomaly Prediction Function has the capability to register with all NFV-MANO functional entities to receive relevant KPIs produced by these NFV-MANO functional entities.	Clause 7.3.3.4
Kap.004	It is recommended that the Anomaly Prediction Function has the capability of collecting the input necessary to run the KPI forecast model (see note).	Clause 7.3.3.4
NOTE: The capability of collecting input can mean the capability of collecting configuration and other availability related information, as well as subscribing for and receiving notifications about current state, state changes, scaling, etc.		

9 Conclusion

The present document has studied the cognitive use of operations data for reliability and availability purposes. Such data (alarms, notifications, syslogs, …), from a local data centre or from the Cloud, arise from operations launched at the NFV-MANO interfaces (clause 5).

Data driven techniques which can be used for a cognitive analysis were presented in clause 6, together with their possible exploitation for reliability and availability in an NFV environment.

In clause 7, three use cases for the application of machine learning have been developed and analysed:

- A Service Availability Assurance Function using ANNs to process operational data was shown to provide better guarantees for maintaining requested service availability with more efficient use of resources.

- Through a Fault Detection and Localization Function deploying 2-layered SOM, the capability of classifying KPI metrics with respect to service degradation/failure was presented that pinpoints the most probable location of resource failures and by that speeding up and improving RCA.

- An Anomaly Prediction Function based on different supervised learning techniques was shown to use log messages and KPI metrics to predict potential anomalies and trigger preventative measure to avoid those anomalies to happen.

From the three use cases mentioned above, a set of recommendations were derived in clause 8 calling for the specification of requirements and preferences to provide these new functionalities and their different aspects in an NFV environment. Implementing these recommendations would significantly improve the reliability and availability of network services and virtualised network functions supported in the NFV environment.

Although the use cases studied in the present document are important, it is worth mentioning that they are not exhaustive, i.e. more use cases exist which can benefit from the application of cognitive management to ensure a dependable NFV environment.

Annex A: NFVI interfaces and operations

A.1 Introduction

The present annex lists most types of operations data in use in the NFV ecosystem. As such data are initially issued from operations launched at the different NFV interfaces, the main part of the present Annex (clauses A.2 to A.6) lists all these interfaces. The interfaces are classified according to the domains identified in ETSI GR NFV-IFA 015 [i.9]: NFV common domain (clause A.2), NFV-MANO OAM (clause A.3), NS (clause A.4), resource (clause A.5), and VNF (clause A.6).

Based on the set of NFV specifications ([i.1] to [i.8]), for each interface, the first table lists the related operations with an indication of the reference points (if applicable) where these are applied.

NOTE: Since ETSI GR NFV-IFA 015 [i.9] served as basis for inter-relating the interface specifications [i.1] to [i.8], only the Release 3 versions of these specifications were considered for consistency.

The input/output parameters, followed by their type, are also provided for the operations. The second table develops for each interface, the different information elements in use, associated with their attributes and types. The notation applied to the whole Annex is the following:

- 'xor' means that only one parameter:type or attribute:type can be used at a time from the list;
- regular typeset means a primitive/predefined type;
- bold typeset means a type/information element whose data structure is given in the same clause - unless a reference is provided;
- italic typeset means Enum whose values are listed in clause A.8;
- bold+italic typeset means abstract type in place of which an appropriate specialization/child type can be used - note that not all specializations are included in the present annex.

Clause A.7 presents the common information elements used for the operations at the interfaces, and shown using the domain classification of ETSI GR NFV-IFA 015 [i.9]: NFV (clause A.7.1), NS (clause A.7.2), resource (clause A.7.3), and VNF (clause A.7.4).

The last clause of the present annex (clause A.8) lists the enumeration values for all NFV's operations data.

A.2 NFV common domain

A.2.1 Fault management

Table A.2.1-1 shows the input/output parameters for each operation of the fault management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.2.1-2, otherwise a reference is provided.

Table A.2.1-1: Fault management operations

Operation	Or-Vi	Vnfm-Vi	Or-Vnfm	Ve-Vnfm	Os-Ma-Nfvo	Or-Or	Consumer-NFV-MANO FE	Consumer-WIM	Input/output Parameter:Type
Subscribe	X	X	X	X	X	X	X	X	filter:Filter subscriptionId:Identifier
Terminate Subscription			X	X	X	X	X	X	subscriptionId:Identifier none
Get Alarm List	X	X	X	X	X	X	X	X	filter:Filter alarm:**Alarm** xor **AlarmWithRpInfo**
Query Subscription Info			X	X	X	X	X	X	filter:Filter queryResult:<not specified>
Escalate Perceived Severity				X					alarmId:Identifier perceivedSeverity:*PerceivedSeverity* none
Acknowledge Alarms			X		X	X	X	X	alarmId:Identifier acknowledgedAlarmId:Identifier
Notify	X	X	X	X	X	X	X	X	alarmNotification:**AlarmNotification** xor alarmWithRpNotification:**AlarmWith RpNotification** xor alarmClearedNotification:**AlarmClearedNotification** xor alarmClearedWithRpNotification:**AlarmClearedWith RpNotification**
			X	X	X	X	X	X	alarmListRebuiltNotification:AlarmListRebuiltNotification

Table A.2.1-2 shows the information elements used in the operations data related to fault management, unless a reference is provided.

Table A.2.1-2: Information elements used in the fault management operations data

Information element	Attribute:Type
Alarm	alarmId:Identifier managedObjectId:Identifier vnfcId:Identifier rootCauseFaultyComponent:**FaultyComponentInfo** rootCauseFaultyResource:**FaultyResourceInfo** alarmRaisedTime:DateTime alarmChangedTime:DateTime alarmClearedTime:DateTime ackState:*AckState* perceivedSeverity:*PerceivedSeverity* eventTime:DateTime eventType:*EventType* faultType:String probableCause:String isRootCause:Boolean correlatedAlarmId:Identifier faultDetails:<not specified> rootCauseFaultyObject:Identifier state:*AlarmState*
AlarmWithRpInfo	resourceProviderId:Identifier alarmId:Identifier managedObjectId:Identifier vnfcId:Identifier rootCauseFaultyComponent:**FaultyComponentInfo** rootCauseFaultyResource:**FaultyResourceInfo** alarmRaisedTime:DateTime alarmChangedTime:DateTime alarmClearedTime:DateTime ackState:*AckState* perceivedSeverity:*PerceivedSeverity* eventTime:DateTime eventType:*EventType* faultType:String probableCause:String isRootCause:Boolean correlatedAlarmId:Identifier faultDetails:<not specified> rootCauseFaultyObject:Identifier state:*AlarmState*
AlarmNotification	alarm:**Alarm**
AlarmWithRpNotification	resourceProviderId:Identifier alarm:**Alarm**
AlarmClearedNotification	alarmId:Identifier alarmClearedTime:DateTime
AlarmClearedWith RpNotification	resourceProviderId:Identifier alarmId:Identifier alarmClearedTime:DateTime
FaultyComponentInfo	faultyNestedNsInstanceId:**NsInfo** (see Table 4.2-2) faultyNsVirtualLinkInstanceId:Identifier faultyVnfInstanceId:Identifier
FaultyResourceInfo	faultyResource:**ResourceHandle** (see Table A.7.1-4) faultyResourceType:*FaultyResourceType*

A.2.2 Performance management

Table A.2.2-1 shows the input/output parameters for each operation of the performance management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.2.2-2, otherwise a reference is provided.

Table A.2.2-1: Performance management operations

Operation	Or-Vi	Vnfm-Vi	Or-Vnfm	Ve-Vnfm	Os-Ma-Nfvo	Or-Or	Consumer-NFV-MANO FE	Consumer-WIM	Input/output Parameter:Type
Subscribe	X	X	X	X	X	X	X	X	filter:Filter
									subscriptionId:Identifier
Terminate Subscription			X	X			X	X	subscriptionId:Identifier
									none
Query Subscription			X	X			X	X	filter:Filter
									queryResult:<not specified>
Create PM Job	X	X	X	X	X	X	X	X	sourceSelector:**ObjectSelection** performanceMetric:String performanceMetricGroup:String collectionPeriod:<not specified> reportingPeriod:<not specified> reportingBoundary:<not specified>
									pmJobId:Identifier
Delete PM Jobs	X	X	X	X	X	X	X	X	pmJobId:Identifier
									deletedPmJobId:Identifier
Query PM Job	X	X	X	X	X	X	X	X	filter:Filter
									pmJob:**PmJob**
Create Threshold	X	X	X	X	X	X	X	X	objectInstanceId:Identifier sourceSelector:**ObjectSelection** performanceMetric:String thresholdType:*ThresholdType* thresholdDetails:<not specified>
									thresholdId:Identifier
Delete Thresholds	X	X	X	X	X	X	X	X	thresholdId:Identifier
									deletedThresholdId:Identifier
Query Threshold	X	X	X	X	X	X	X	X	filter:Filter
									threshold:**Threshold**
Notify	X	X	X	X	X	X	X	X	performanceInformationAvailableNotification:**PerformanceInformationAvailableNotification** xor performanceInformationWithRpAvailableNotification:**PerformanceInformationWithRpAvailableNotification** xor thresholdCrossedNotification: **ThresholdCrossedNotification** xor thresholdCrossedNotification: **ThresholdCrossedWithRpNotification**

Table A.2.2-2 shows the information elements used in the operations data related to performance management, unless a reference is provided.

Table A.2.2-2: Information elements used in the performance management operations data

Information element	Attribute:Type
ObjectSelection	objectType:String objectFilter:Filter objectInstanceId:Identifier
PmJob	pmJobId:Identifier objectSelector:**ObjectSelection** performanceMetric:String performanceMetricGroup:String collectionPeriod:<not specified> reportingPeriod:<not specified> reportingBoundary:<not specified> objectInstanceId:Identifier
Threshold	thresholdId:Identifier objectSelector:**ObjectSelection** performanceMetric:String thresholdType:*ThresholdType* thresholdDetails:<not specified> objectInstanceId:Identifier
PerformanceInformationAvailableNotification	objectInstanceId:Identifier
PerformanceInformationWithRpAvailableNotification	resourceProviderId:Identifier objectInstanceId:Identifier
ThresholdCrossedNotification	thresholdId:Identifier crossingDirection:*ThresholdCrossing* objectInstanceId:Identifier performanceMetric:String performanceValue:Value measurementContext:<not specified>
ThresholdCrossedWithRpNotification	resourceProviderId:Identifier thresholdId:Identifier crossingDirection:*ThresholdCrossing* objectInstanceId:Identifier performanceMetric:String performanceValue:Value measurementContext:<not specified>
PerformanceReport	performanceReport:**PerformanceReportEntry**
PerformanceReportEntry	objectType:String objectInstanceId:Identifier performanceMetric:String performanceValue:**PerformanceValueEntry**
PerformanceValueEntry	timeStamp:DateTime performanceValue:Value measurementContext:<not specified>

A.2.3 Policy management

Table A.2.3-1 shows the input/output parameters for each operation of the policy management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.2.3-2, otherwise a reference is provided.

Table A.2.3-1: Policy management operations

Operation	Or-Vi	Vnfm-Vi	Or-Vnfm	Ve-Vnfm	Os-Ma-Nfvo	Or-Or	Input/output Parameter:Type
Transfer Policy	X	X	X	X	X	X	designer:String name:String version:Version policy:<not specified>
							policyInfoId:Identifier
Delete Policy	X	X	X	X	X	X	policyInfoId:Identifier
							deletedPolicyInfoId:Identifier
Query Policy	X	X	X	X	X	X	filter:Filter attributeSelector:String
							queryNsPolicyInfoResult:**PolicyInfo**
Activate Policy	X	X	X	X	X	X	policyInfoId:Identifier
							activatedPolicyInfoId:Identifier
Deactivate Policy	X	X	X	X	X	X	policyInfoId:Identifier
							deactivatedPolicyInfoId:Identifier
Associate Policy			X	X	X	X	policyInfoId:Identifier nsInstanceId:Identifier
							nsInstanceId:Identifier
Disassociate Policy			X	X	X	X	policyInfoId:Identifier nsInstanceId:Identifier
							queryResult:<not specified>
Subscribe	X	X	X	X	X	X	filter:Filter
							subscriptionId:Identifier
Terminate Subscription	X	X	X	X	X	X	subscriptionId:Identifier
							none
Query Subscription Info	X	X	X	X	X	X	filter:Filter
							queryResult:<not specified>
Notify	X	X	X	X	X	X	policyChangeNotification:**PolicyChangeNotification** xor policyConflictNotification:**PolicyConflictNotification**

Table A.2.3-2 shows the information elements used in the operations data related to policy management, unless a reference is provided.

Table A.2.3-2: Information elements used in the policy management operations data

Information element	Attribute:Type
PolicyInfo	policyInfoId:Identifier designer:String name:String version:Version policy:<not specified> activationStatus:*ActivationStatus*
PolicyChangeNotification	policyInfoId:Identifier operation:*PolicyChangeOperations*
PolicyConflictNotification	policyInfoId:Identifier conflictDescription:<not specified>

A.3 NFV-MANO OAM domain

A.3.1 NFV-MANO configuration and information management

Table A.3.1-1 shows the input/output parameters for each operation of the NFV-MANO configuration and information management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.3.1-2, otherwise a reference is provided.

Table A.3.1-1: NFV-MANO configuration and information management operations

Operation	**Consumer-NFV-MANO FE**	**Input/output Parameter:Type**
Subscribe	X	filter:Filter
		subscriptionId:Identifier
Terminate Subscription	X	subscriptionId:Identifier
		none
Query Subscription Information	X	filter:Filter
		queryResult:<not specified>
Modify Config	X	newValues:KeyValuePair
		modifiedValues:KeyValuePair
Query Config Info	X	filter:Filter attributeSelector:String
		manoEntityInfo:**ManoEntityInfo**
Change State	X	manoEntityInterfaceId:Identifier changeOperation:<not specified>
		none
Notify	X	informationChangedNotification:**InformationChangedNotification** xor stateChangeNotification:**StateChangeNotification**

Table A.3.1-2 shows the information elements used in the operations data related to NFV-MANO configuration and information management, unless a reference is provided.

Table A.3.1-2: Information elements used in the NFV-MANO configuration and information management operations data

Information element	Attribute:Type
ManoEntityInfo	manoEntityId:Identifier manoEntityType:*ManoEntityType* manoEntityName:String manoEntityDescription:String manoEntityProvider:String manoEntitySoftwareVersion:Version manoEntityComponent:**ManoEntityComponent** manoEntityInterface:**ManoEntityInterface** manoConfigurableParam:**ManoConfigurableParam** manoApplicationState:<not specified> manoMonitoringConfigParameter:<not specified> manoService:**ManoServiceInfo** nfvoSpecificInfo:**NfvoSpecificInfo** xor vnfmSpecificInfo:**VnfmSpecificInfo** xor vimSpecificInfo:**VimSpecificInfo**
InformationChangedNotification	informationChangedTime:DateTime manoEntityChangedInfo:KeyValuePair
StateChangeNotification	manoEntityInterfaceId:Identifier stateChange:<not specified>
ManoEntityInterface	manoEntityInterfaceId:Identifier manoEntityInterfaceName:String manoEntityInterfaceType:*ManoEntityInterfaceType* standardVersion:Version providerSpecificApiVersion:Version apiEndpoint:<not specified> supportedOperation:**SupportedOperation** maxConcurrentIntOpNumber:Integer securityInfo:<not specified> manoEntityInterfaceState:<not specified>
ManoEntityComponent	manoEntityComponentId:Identifier manoServiceId:Identifier
ManoServiceInfo	manoServiceId:Identifier manoServiceName:String manoServiceDescription:String manoEntityInterfaceId:Identifier
NfvoSpecificInfo	maxOnboardedNsdNum:Integer maxOnboardedVnfPkgNum:Integer maxNsInstanceNum:<not specified> supportedVnfdFormat:String supportedNsdFormat:String
VnfmSpecificInfo	resourceMgmtModeSupport:*ResourceMgmtModeSupport* managedVnfInstanceInfo:String maxVnfInstanceNum:<not specified> supportedVnfdFormat:String
VimSpecificInfo	maxVirtualResourceNum:<not specified>
ManoConfigurableParam	manoPeerConfig:**ManoPeerConfig** ntpServer:<not specified>
ManoPeerConfig	peerManoEntityType:*ManoEntityType* peerManoEntityId:Identifier apiDiscoveryEndpoint:<not specified> manoConsumerInterface:**ManoConsumerInterfaceInfo** statePeerManoEntity:<not specified>
SupportedOperation	operationName:String maxConcurrentOpNumber:Integer
ManoConsumerInterfaceInfo	manoConsumerInterfaceId:Identifier manoConsumerInterfaceName:String manoConsumerInterfaceType:*ManoConsumerInterfaceType* standardVersion:Version providerSpecificApiVersion:Version apiEndpoint:<not specified> securityInfo:<not specified> consumerOpTimeout:Integer maxConcurrentConsumerOpNumber:Integer

A.3.2 NFV-MANO log management

Table A.3.2-1 shows the input/output parameters for each operation of the NFV-MANO log management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.3.2-2, otherwise a reference is provided.

Table A.3.2-1: NFV-MANO log management operations

Operation	Consumer-NFV-MANO FE	Input/output Parameter:Type
Subscribe	X	filter:Filter
		subscriptionId:Identifier
Terminate Subscription	X	subscriptionId:Identifier
		none
Query Subscription Info	X	filter:Filter
		queryResult:<not specified>
Create Logging Job	X	startTime:DateTime endTime:DateTime logObjectSelector:<not specified> isEncrypted:Boolean loggingConfig:KeyValuePair reportingCondition:<not specified>
		loggingJobId:Identifier
Stop Logging Job	X	loggingJobId:Identifier
		none
Query Logging Job	X	filter:Filter
		loggingJobDetails:**LoggingJob**
Notify	X	logReportAvailabilityNotification:**LogReportAvailabilityNotification**

Table A.3.2-2 shows the information elements used in the operations data related to NFV-MANO log management, unless a reference is provided.

Table A.3.2-2: Information elements used in the NFV-MANO log management operations data

Information element	Attribute:Type
LoggingJob	loggingJobId:Identifier startTime:DateTime endTime:DateTime logObjectSelector:<not specified> isEncrypted:Boolean loggingConfig:KeyValuePair reportingCondition:<not specified>
LogReportAvailabilityNotification	objectInstanceId:Identifier loggingJobId:Identifier location:<not specified>

A.4 NS domain

A.4.1 NSD management

Table A.4.1-1 shows the input/output parameters for each operation of the NSD management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.4.1-2, otherwise a reference is provided.

Table A.4.1-1: NSD management operations

Operation	Os-Ma-Nfvo	Or-Or	Input/output Parameter:Type
Upload NSD	X		nsdInfoId:Identifier nsd:**Nsd** (see Table A.7.2-2)
			none
Update NSD Info	X		nsdInfoId:Identifier operationalState:*OperationalState*
			none
Delete NSD	X		nsdInfoId:Identifier
			deletedNsdInfoId:Identifier
Create NSD Info	X		userDefinedData:KeyValuePair
			nsdInfoId:Identifier
Query NSD Info	X	X	filter:Filter attributeSelector:String
			queryResult:**NsdInfo** (see Table A.4.2-2)
Fetch NSD	X		nsdInfoId:Identifier
			Nsd:**Nsd** (see Table A.7.2-2)
Fetch NSD Archive Artifacts	X		nsdInfoId:Identifier artifactSelector:<not specified>
			nsdArchiveArtifact:<not specified>
Upload PNFD	X		pnfdInfoId:Identifier pnfdArchive:Binary
			none
Update PNFD Info	X		pnfdInfoId:Identifier userDefinedData:KeyValuePair
			none
Delete PNFD	X		pnfdInfoId:Identifier applyOnAllVersions:Boolean
			deletedPnfdInfoId:Identifier
Create PNFD Info	X		userDefinedData:KeyValuePair
			pnfdInfoId:Identifier
Query PNFD Info	X		filter:Filter attributeSelector:String
			queryResult:**PnfdInfo** (see Table A.4.2-2)
Fetch PNFD	X		pnfdInfoId:Identifier
			pnfdArchive:Binary
Fetch PNFD Archive Artifacts	X		pnfdInfoId:Identifier artifactSelector:<not specified>
			pnfdArchiveArtifact:<not specified>
Subscribe	X		filter:Filter
			subscriptionId:Identifier
Terminate Subscription	X		subscriptionId:Identifier
			none
Query Subscription Info	X		filter:Filter
			queryResult:<not specified>
Notify	X		nsdOnBoardingNotification:**NsdOnBoardingNotification** xor nsdChangeNotification:**NsdChangeNotification** xor nsdDeletionNotification:**NsdDeletionNotification** xor pnfdOnBoardingNotification:**PnfdOnBoardingNotification** xor pnfdDeletionNotification:**PnfdDeletionNotification**

Table A.4.1-2 shows the information elements used in the operations data related to NSD management.

Table A.4.1-2: Information elements used in the NSD management operations data

Information element	Attribute:Type
NsdChangeNotification	nsdInfoId:Identifier nsdId:Identifier operationalState:*OperationalState*
NsdDeletionNotification	nsdInfoId:Identifier nsdId:Identifier
NsdOnBoardingNotification	nsdInfoId:Identifier nsdId:Identifier
PnfdDeletionNotification	pnfdInfoId:Identifier pnfdId:Identifier
PnfdOnBoardingNotification	pnfdInfoId:Identifier pnfdId:Identifier

A.4.2 NS lifecycle management

Table A.4.2-1 shows the input/output parameters for each operation of the NS lifecycle management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.4.2-2, otherwise a reference is provided.

Table A.4.2-1: NS lifecycle management operations

Operation	Os-Ma-Nfvo	Or-Or	Input/output Parameter:Type
Subscribe	X	X	filter:Filter
			subscriptionId:Identifier
Terminate Subscription	X	X	subscriptionId:Identifier
			none
Query Subscription Info	X	X	filter:Filter
			queryResult:<not specified>
Create NS Identifier	X	X	nsdId:Identifier nsName:String nsDescription:String
			nsInstanceId:Identifier
Delete NS Identifier	X	X	nsInstanceId:Identifier
			none
Instantiate NS	X	X	nsInstanceId:Identifier flavourId:Identifier sapData:**SapData** addPnfData:**AddPnfData** vnfInstanceData:**VnfInstanceData** nestedNsInstanceData:**NestedNsInstanceData** locationConstraints:**VnfLocationConstraint** nestedNsLocationConstraints:**NestedNsLocationConstraint** additionalParamForNs:KeyValuePair additionalParamForNestedNs:**ParamsForNestedNs** additionalParamForVnf:**ParamsForVnf** startTime:DateTime nsInstantiationLevelId:Identifier wanConnectionData:**WanConnectionData** additionalAffinityOrAntiAffinityRule:**AffinityOrAntiAffinityRule**
			lifecycleOperationOccurrenceId:Identifier

Operation	Os-Ma-Nfvo	Or-Or	Input/output Parameter:Type
Scale NS	X	X	nsInstanceId:Identifier scaleType:ScaleType scaleNsData:**ScaleNsData** scaleVnfData:**ScaleVnfData** scaleTime:DateTime
			lifecycleOperationOccurrenceId:Identifier
Update NS	X		nsInstanceId:Identifier updateType:<not specified> addVnfInstance:**VnfInstanceData** removeVnfInstanceId:Identifier instantiateVnfData:**InstantiateVnfData** changeVnfFlavourData:**ChangeVnfFlavourData** operateVnfData:**OperateVnfData** modifyVnfInfoData:**ModifyVnfInfoData** changeExtVnfConnectivityData:**ChangeExtVnfConnectivityData** addSap:**SapData** removeSapId:Identifier addNestedNsData:**NestedNsInstanceData** removeNestedNsId:Identifier assocNewNsdVersionData:**AssocNewNsdVersionData** moveVnfInstanceData:**MoveVnfInstanceData** addVnffg:**AddVnffgData** removeVnffgId:Identifier updateVnffg:**UpdateVnffgData** changeNsFlavourData:**ChangeNsFlavourData** updateTime:DateTime addPnfData:**AddPnfData** modifyPnfData:**ModifyPnfData** removePnfId:Identifier createSnapshotVnfInstanceId:Identifier revertToSnapshotData:**RevertToSnapshotData** deleteSnapshotData:**DeleteSnapshotData** associatePnfWithPnfProfile:**PnfProfileData** associateVnfWithVnfProfile:**VnfProfileData** changeVnfPkgData:**ChangeVnfPackageData** nsVirtualLinkProfile:**VirtualLinkProfile** (see Table A.7.2-2) deleteNsVirtualLinkId:Identifier modifyWanConnectionInfoData:**ModifyWanConnectionInfoData**
			vnfInstanceId:Identifier pnfId:Identifier vnffgId:Identifier sapId:Identifier vnfSnapshotInfoId:Identifier lifecycleOperationOccurrenceId:Identifier
Heal NS	X	X	nsInstanceId:Identifier healNsData:**HealNsData** healVnfData:**HealVnfData**
			lifecycleOperationOccurrenceId:Identifier
Terminate NS	X	X	nsInstanceId:Identifier terminateTime:DateTime
			lifecycleOperationOccurrenceId:Identifier
Query NS	X	X	filter:Filter attributeSelector:String
			queryNsResult:**NsInfo**
Get Operation Status	X	X	lifecycleOperationOccurrenceId:Identifier
			operationStatus:*LcmOperationStatus*
Notify	X	X	nsLcmOperationOccurrenceNotification:**NsLcmOperationOccurrenceNotification** xor nsChangeNotification:**NsChangeNotification** xor nsIdentifierCreationNotification:**NsIdentifierCreationNotification** xor nsIdentifierDeletionNotification:**NsIdentifierDeletionNotification** xor nsLcmCapacityShortageNotification:**NsLcmCapacityShortageNotification**

Table A.4.2-2 shows the information elements used in the operations data related to NS lifecycle management.

Table A.4.2-2: Information elements used in the NS lifecycle management operations data

Information element	Attribute:Type
AddVnffgData	vnffgdId:Identifier vnffgName:String description:String
AffinityOrAntiAffinityRule	descriptorId:Identifier vnfInstanceId:Identifier affinityOrAntiAffinity:*AffinityOrAntiAffinity* scope:*AffinityOrAntiAffinityScope*
AssocNewNsdVersionData	newNsdId:Identifier sync:Boolean
ChangeExtVnfConnectivityData	vnfInstanceId:Identifier extVirtualLink:**ExtVirtualLinkData** (see Table A.7.1-4) additionalParam:KeyValuePair
ChangeNsFlavourData	newFlavourId:Identifier nsInstantiationLevelId:Identifier
ChangeVnfFlavourData	vnfInstanceId:Identifier newFlavourId:Identifier instantiationLevelId:Identifier extVirtualLink:**ExtVirtualLinkData** (see Table A.7.1-4) extManagedVirtualLink:**ExtManagedVirtualLinkData** (see Table A.7.1-4) additionalParam:KeyValuePair extension:KeyValuePair vnfConfigurableProperty:KeyValuePair
ChangeVnfPackageData	vnfInstanceId:Identifier vnfdId:Identifier extVirtualLink:**ExtVirtualLinkData** (see Table A.7.1-4) extManagedVirtualLink:**ExtManagedVirtualLinkData** (see Table A.7.1-4) additionalParam:KeyValuePair extension:KeyValuePair vnfConfigurableProperties:KeyValuePair
DeleteSnapshotData	vnfSnapshotInfoId:Identifier vnfInstanceId:Identifier
HealNsData	degreeHealing:*NsDegreeHealing* actionsHealing:String healScript:**LifeCycleManagementScript** (see Table A.7.2-2) additionalParamForNs:KeyValuePair
HealVnfData	vnfInstanceId:Identifier cause:String additionalParam:KeyValuePair
InstantiateVnfData	vnfdId:Identifier flavourId:Identifier instantiationLevelId:Identifier vnfInstanceName:String vnfInstanceDescription:String extVirtualLink:**ExtVirtualLinkData** (see Table A.7.1-4) extManagedVirtualLink:**ExtManagedVirtualLinkData** (see Table A.7.1-4) localizationLanguage:<not specified> additionalParam:KeyValuePair locationConstraint:**VnfLocationConstraint** metadata:KeyValuePair extension!KeyValuePair

Information element	Attribute:Type
ModifyVnfInfoData	vnfInstanceId:Identifier newValues:KeyValuePair
ModifyWanConnectionInfoData	wanConnectionInfoId:Identifier newProtocolData:<not specified>
MoveVnfInstanceData	targetNsInstanceId:Identifier vnfInstanceId:Identifier
NestedNsInstanceData	nestedNsInstanceId:Identifier nsProfileId:Identifier
NestedNsLocationConstraint	nsProfileId:Identifier locationConstraints:<not specified>
OperateVnfData	vnfInstanceId:Identifier changeStateTo:*VnfState* stopType:*VnfStopType* gracefulStopTimeout:TimeDuration additionalParam:KeyValuePair
ParamsForNestedNs	nsProfileId:Identifier additionalParam:KeyValuePair
ParamsForVnf	vnfProfileId:Identifier additionalParam:KeyValuePair
PnfProfileData	pnfId:Identifier pnfProfileId:Identifier
RevertToSnapshotData	vnfInstanceId:Identifier vnfSnapshotInfoId:Identifier
SapData	sapdId:Identifier sapName:String description:String address:<not specified>
ScaleNsData	vnfInstanceToBeAdded:**VnfInstanceData** vnfInstanceToBeRemoved:Identifier scaleNsByStepsData:**ScaleNsByStepsData** scaleNsToLevelData:**ScaleNsToLevelData** additionalParamForNs:KeyValuePair additionalParamForVnf:**ParamsForVnf** locationConstraints:**VnfLocationConstraint** nestedNsLocationConstraints:**NestedNsLocationConstraint**
ScaleNsToLevelData	nsInstantiationLevel:**NsLevel** (see Table A.7.2-2) nsScaleInfo:**NsScaleInfo**
ScaleVnfData	vnfInstanceId:Identifier type:<not specified> scaleToLevelData:**ScaleToLevelData** scaleByStepData:**ScaleByStepData**
UpdateVnffgData	vnffgId:Identifier nfp:**NfpData** nfpId:Identifier
VnfInstanceData	vnfInstanceId:Identifier vnfProfileId:Identifier
VnfLocationConstraint	vnfProfileId:Identifier locationConstraints:<not specified>
VnfProfileData	vnfInstanceId:Identifier vnfProfileId:Identifier
WanConnectionData	virtualLinkDescId:Identifier protocolData:<not specified>

Information element	Attribute:Type
NsInfo	nsInstanceId:Identifier nsName:String description:String nsdId:Identifier nsdInfoId:Identifier flavourId:Identifier vnfInfo:**VnfInfo** (see Table A.6.5-2) pnfInfo:**PnfInfo** virtualLinkInfo:**NsVirtualLinkInfo** vnffgInfo:**VnffgInfo** sapInfo:**SapInfo** nestedNsInfoId:Identifier vnfSnapshotInfo:**VnfSnapshotInfo** (see Table A.6.5-2) nsState:*InstantiationState* monitoringParameter:**MonitoringParameter** (see Table 7.1-1) nsScaleStatus:**NsScaleInfo** additionalAffinityOrAntiAffinityRule:**AffinityOrAntiAffinityRule** wanConnectionInfo:**WanConnectionInfo**
AddPnfData	pnfId:Identifier pnfName:String pnfdId:Identifier pnfProfileId:Identifier cpData:**PnfExtCpData**
CpGroupInfo	cpPairInfo:**CpPairInfo** forwardingBehaviour:*ForwardingBehaviourType* forwardingBehaviourInputParameters:<not specified>
ModifyPnfData	pnfId:Identifier pnfName:String cpData:**PnfExtCpData**
NsChangeNotification	nsInstanceId:Identifier nsComponentType:*NsComponentType* nsComponentId:Identifier lcmOpOccIdImpactingNsComponent:Identifier lcmOpOccNameImpactingNsComponent:String lcmOpOccStatusImpactingNsComponent:<not specified>
NsIdentifierCreationNotification	nsInstanceId:Identifier
NsIdentifierDeletionNotification	nsInstanceId:Identifier
NsLcmOperationOccurrenceNotification	nsInstanceId:Identifier lifecycleOperationOccurrenceId:Identifier operation:String status:*OperationStatus* isAutomaticInvocation:Boolean affectedVnf:**AffectedVnf** affectedPnf:**AffectedPnf** affectedVl:**AffectedVirtualLink** affectedVnffg:**AffectedVnffg** affectedNs:**AffectedNs** affectedSap:**AffectedSap**
NsLcmCapacityShortageNotification	lifecycleOperationOccurrenceId:Identifier nsInstanceId:Identifier status:<not specified> shortageType:<not specified> affectedNs:**AffectedNs** capacityInformation:<not specified>
NsVirtualLinkInfo	nsVirtualLinkInstanceId:Identifier nsVirtualLinkDescId:Identifier virtualLinkProfileId:Identifier resourceHandle:**ResourceHandle** (see Table A.7.1-4) linkPort:**NsLinkPortInfo**
NsLinkPortInfo	nsLinkPortId:Identifier resourceHandle:**ResourceHandle** (see Table A.7.1-4) cpId:Identifier

Information element	Attribute:Type
NfpInfo	nfpId:Identifier nfpdId:Identifier nfpName:String description:String cpGroup:**CpGroupInfo** totalCp:Integer nfpRule:**NfpRule** nfpState:*OperationalState*
NfpData	nfpId:Identifier nfpName:String description:String cpGroup:**CpGroupInfo** nfpRule:**NfpRule**
ScaleByStepData	aspectId:Identifier numberOfSteps:Integer additionalParam:KeyValuePair
ScaleNsByStepsData	scalingDirection:*NsScaleDirection* aspectId:Identifier numberOfSteps:Integer
ScaleToLevelData	instantiationLevelId:Identifier scaleInfo:**ScaleInfo** (see Table A.6.5-2) additionalParam:KeyValuePair
PnfExtCpInfo	cpInstanceId:Identifier cpdId:Identifier cpProtocolInfo:**CpProtocolInfo** (see Table A.7.1-4)
PnfInfo	pnfId:Identifier pnfName:String pnfdId:Identifier pnfdInfoId:Identifier pnfProfileId:Identifier cpInfo:**PnfExtCpInfo**
SapInfo	sapInstanceId:Identifier sapdId:Identifier sapName:String description:String cpProtocolInfo:**CpProtocolInfo** (see Table A.7.1-4)
VnffgInfo	vnffgId:Identifier vnffgdId:Identifier vnfId:Identifier pnfId:Identifier virtualLinkId:Identifier cpId:Identifier nfpInfo:**NfpInfo**
WanConnectionInfo	wanConnectionInfoId:Identifier protocolData:<not specified> virtualLinkInstanceId:Identifier
CpPairInfo	cpInfo:***CpInfo***
NfpRule	etherType:*IpVersion* protocol:String sourcePortRange:**PortRange** destinationPortRange:**PortRange** sourceIPAddressPrefix:IpAddress destinationIPAddressPrefix:IpAddress etherDestinationAddress:MacAddress etherSourceAddress:MacAddress vlanTag:String dscp:String extendedCriteria:<not specified>
NsScaleInfo	nsScalingAspectId:Identifier nsScaleLevelId:Identifier
PnfExtCpData	cpInstanceId:Identifier cpdId:Identifier address:<not specified>
PortRange	lowerPort:Integer upperPort:Integer

Information element	Attribute:Type
AffectedNs	nsInstanceId:Identifier nsdId:Identifier changeType:*NestedNsChangeType* changeResult:*ChangeResultType*
AffectedPnf	pnfId:Identifier pnfName:String pnfdId:Identifier pnfProfileId:Identifier cpInstanceId:Identifier changeType:*PnfChangeType* changeResult:*ChangeResultType*
AffectedSap	sapInstanceId:Identifier sapdId:Identifier sapName:String changeType:*SapChangeType* changeResult:*ChangeResultType*
AffectedVirtualLink	nsVirtualLinkId:Identifier nsVirtualLinkDescId:Identifier virtualLinkProfileId:Identifier changeType:*VirtualLinkChangeType* changeResult:*ChangeResultType*
AffectedVnf	vnfInstanceId:Identifier vnfdId:Identifier vnfProfileId:Identifier vnfName:String changeType:*VnfChangeType* changeResult:*ChangeResultType* changedInfo:<not specified>
AffectedVnffg	vnffgId:Identifier vnffgdId:Identifier changeType:*VnffgChangeType* changeResult:*ChangeResultType*
NsdInfo	nsdInfoId:Identifier nsdId:Identifier name:String version:Version designer:String nsd:**Nsd** (see Table A.7.2-2) vnfPkgInfoId:Identifier pnfdInfoId:Identifier nestedNsdInfoId:Identifier onboardingState:*OnboardingState* operationalState:*OperationalState* usageState:*UsageState* userDefinedData:KeyValuePair artifacts:**NsdArchiveArtifactInformation**
PnfdInfo	pnfdInfoId:Identifier pnfdId:Identifier name:String version:Version provider:String pnfdInvariantId:Identifier pnfd:**Pnfd** (see Table A.7.2-2) onboardingState:*OnboardingState* usageState:*UsageState* userDefinedData:KeyValuePair artifacts:**PnfdArchiveArtifactInformation**
NsdArchiveArtifactInformation	selector:<not specified> metadata:<not specified>
PnfdArchiveArtifactInformation	selector:<not specified> metadata:<not specified>

A.4.3 NS instance usage notification

Table A.4.3-1 shows the input/output parameters for each operation of the NS instance usage notification interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.4.3-2, otherwise a reference is provided.

Table A.4.3-1: NS instance usage notification operations

Operation	Or-Or	Input/output Parameter:Type
Subscribe	X	filter:Filter
		subscriptionId:Identifier
Terminate Subscription	X	subscriptionId:Identifier
		none
Query Subscription	X	filter:Filter
		queryResult:<not specified>
Notify	X	nsInstanceUsageNotification:**NsInstanceUsageNotification**

Table A.4.3-2 shows the information elements used in the operations data related to NS instance usage notification.

Table A.4.3-2: Information elements used in the NS instance usage notification operations data

Information element	Attribute:Type
NsInstanceUsageNotification	nsInstanceId:Identifier status:*NsInstanceUsageStatus*

A.4.4 NS lifecycle operation granting

Table A.4.4-1 shows the input/output parameters for the operation of the NS lifecycle operation granting interface.

Table A.4.4-1: NS lifecycle operation granting operations

Operation	Or-Or	Input/output Parameter:Type
Grant NS Lifecycle	X	nsInstanceId:Identifier nsdId:Identifier lifecycleOperation:**NsLifecycleOperation** (see Table A.8-1) additionalParam:KeyValuePair
		none

A.4.5 LCM coordination

Table A.4.5-1 shows the input/output parameters for the operation of the LCM coordination interface.

Table A.4.5-1: LCM coordination operations

Operation	Os-Ma-Nfvo	Input/output Parameter:Type
CoordinateLcmOperation	X	nsInstanceId:Identifier lifecycleOperationOccurrenceId:Identifier operationType:<not specified> operationStage:<not specified> operationParam:<not specified>
		operationAction:*OperationAction* operationResumeDelay:TimeDuration additionalInfo:<not specified>

A.5 Resource domain

A.5.1 Software image management

Table A.5.1-1 shows the input/output parameters for each operation of the software image management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.1-2, otherwise a reference is provided.

Table A.5.1-1: Software image management operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Add Image	X		name:String provider:String version:<not specified> userMetadata:KeyValuePair softwareImage:<not specified> resourceGroupId:Identifier visibility:*Visibility*
			softwareImageMetadata:**SoftwareImageInformation**
Query Images	X	X	imageQueryFilter:Filter
			softwareImageInformation:**SoftwareImageInformation**
Query Image	X	X	softwareImageId:Identifier
			softwareImageInformation:**SoftwareImageInformation**
Update Image	X		softwareImageId:Identifier userMetadata:KeyValuePair
			softwareImageMetadata:**SoftwareImageInformation**
Delete Image	X		softwareImageId:Identifier
			deletedId:Identifier

Table A.5.1-2 shows the information elements used in the operations data related to software image management, unless a reference is provided.

Table A.5.1-2: Information elements used in the software image management operations data

Information element	Attribute:Type
SoftwareImageInformation	softwareImageId:Identifier name:<not specified> provider:<not specified> version:<not specified> checksum:<not specified> containerFormat:<not specified> diskFormat:<not specified> createdAt:<not specified> minDisk:<not specified> minRam:<not specified> size:<not specified> userMetadata:KeyValuePair updatedAt:<not specified> status:<not specified>

A.5.2 Virtualised compute

A.5.2.1 Virtualised compute resources management

Table A.5.2.1-1 shows the input/output parameters for each operation of the virtualised compute resources management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.2.1-2, otherwise a reference is provided.

Table A.5.2.1-1: Virtualised compute resources management operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Allocate Virtualised Compute Resource	X	X	computeName:String reservationId:Identifier affinityOrAntiAffinityConstraint:**AffinityOrAntiAffinityConstraint** computeFlavourId:Identifier vcImageId:Identifier interfaceData:**VirtualInterfaceData** metaData:KeyValuePair resourceGroupId:Identifier locationConstraints:<not specified> userData:**UserData**
			computeData:**VirtualCompute** (see Table A.7.3-1) xor computeData:**ComputeResourceWithRpInfo**
Query Virtualised Compute Resource	X	X	queryComputeFilter:Filter
			queryResult:**VirtualCompute** (see Table A.7.3-1) xor queryResult:**ComputeResourceWithRpInfo**
Update Virtualised Compute Resource	X	X	computeId:Identifier networkInterfaceNew:**VirtualNetworkInterfaceData** networkInterfaceUpdate:**VirtualNetworkInterface** (see Table A.7.3-2) metaData:KeyValuePair
			computeId:Identifier computeData:**VirtualCompute** (see Table A.7.3-1) xor computeData:**ComputeResourceWithRpInfo**
Terminate Virtualised Compute Resource	X	X	computeId:Identifier xor computeId:**IdComputeResourceWithRpId**
			computeId:Identifier xor computeId:**IdComputeResourceWithRpId**
Operate Virtualised Compute Resource	X	X	computeId:Identifier computeOperation:String computeOperationInputData:KeyValuePair
			computeData:**VirtualCompute** (see Table A.7.3-1) xor computeData:**ComputeResourceWithRpInfo** computeOperationOutputData:KeyValuePair
Scale Virtualised Compute Resource	X	X	computeId:Identifier computeFlavourId:Identifier
			computeData:**VirtualCompute** (see Table A.7.3-1) xor computeData:**ComputeResourceWithRpInfo**
Migrate Virtualised Compute Resource	X	X	computeId:Identifier migrationConstraint:<not specified> affinityOrAntiAffinityConstraint:**AffinityOrAntiAffinityConstraint** migrationType:*MigrationType*
			computeData:**VirtualCompute** (see Table A.7.3-1) xor computeData:**ComputeResourceWithRpInfo**
Create Virtualised Compute Resource Affinity Or AntiAffinity Constraints Group	X	X	groupName:String type:*AffinityOrAntiAffinity* scope:*AffinityOrAntiAffinityScope*
			groupId:Identifier
Attach Virtualised Storage Resource	X	X	computeId:Identifier storageId:Identifier mountpoint:String
			computeData:**VirtualCompute** (see Table A.7.3-1) xor computeData:**ComputeResourceWithRpInfo**
Detach Virtualised Storage Resource	X	X	computeId:Identifier storageId:Identifier
			none

Table A.5.2.1-2 shows the information elements used in the operations data related to virtualised compute resources management, unless a reference is provided.

Table A.5.2.1-2: Information elements used in the virtualised compute resources management operations data

Information element	Attribute:Type
AffinityOrAntiAffinityConstraint	type:AffinityOrAntiAffinityConstraintType scope:AffinityOrAntiAffinityScope affinityOrAntiAffinityResourceList:**AffinityOrAntiAffinityResourceList** (see Table A.7.1-4) affinityOrAntiAffinityResourceGroupId:Identifier
UserData	content:String method:UserDataTransportationMethod
VirtualInterfaceData	ipAddress:IpAddress macAddress:MacAddress
VirtualNetworkInterfaceData	networkId:Identifier networkPortId:Identifier typeVirtualNic:<not specified> typeConfiguration:<not specified> bandwidth:Number accelerationCapability:<not specified> metadata:KeyValuePair
IdComputeResourceWithRpId	resourceProviderId:Identifier computeId:Identifier
ComputeResourceWithRpInfo	resourceProviderId:Identifier computeId:Identifier computeName:String flavourId:Identifier accelerationCapability:<not specified> virtualCpu:**VirtualCpu** (see Table A.7.3-2) virtualMemory:**VirtualMemory** (see Table A.7.3-2) virtualNetworkInterface:**VirtualNetworkInterface** (see Table A.7.3-2) virtualDisks:**VirtualStorage** (see Table A.7.3-2) vcImageId:Identifier zoneId:Identifier hostId:Identifier operationalState:OperationalState metadata:KeyValuePair

A.5.2.2 Virtualised compute resources change notification

Table A.5.2.2-1 shows the input/output parameters for each operation of the virtualised compute resources change notification interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.2.2-2, otherwise a reference is provided.

Table A.5.2.2-1: Virtualised compute resources change notification operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Subscribe	X	X	inputFilter:Filter
			subscriptionId:Identifier
Notify	X	X	virtualisedResourceChangeNotification:**VirtualisedResourceChangeNotification** xor virtualisedResourceWithRpChangeNotification:**VirtualisedResourceWithRpChangeNotification**

Table A.5.2.2-2 shows the information elements used in the operations data related to virtualised compute resources change notification, unless a reference is provided.

Table A.5.2.2-2: Information elements used in the virtualised compute resources change notification operations data

Information element	Attribute:Type
VirtualisedResourceChangeNotification	changeId:Identifier virtualisedResourceId:Identifier virtualisedResourceGroupId:Identifier endOfChange:Boolean changeTime:DateTime vimId:Identifier changeType:String changedResourceData:<not specified>
VirtualisedResourceWithRpChangeNotification	resourceProviderId:Identifier changeId:Identifier virtualisedResourceId:Identifier virtualisedResourceGroupId:Identifier endOfChange:Boolean changeTime:DateTime vimId:Identifier changeType:String changedResourceData:<not specified>

A.5.2.3 Virtualised compute resources information management

Table A.5.2.3-1 shows the input/output parameters for each operation of the virtualised compute resources information management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.2.3-2, otherwise a reference is provided.

Table A.5.2.3-1: Virtualised compute resources information management operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Subscribe	X	X	filter:Filter
			subscriptionID:Identifier
Query Virtualised Compute Resource Information	X	X	informationQueryFilter:Filter
			virtualisedResourceInformation:**VirtualComputeResourceInformation** xor virtualisedResourceInformation:**VirtualComputeResourceWithRpInfo**
Notify	X	X	informationChangeNotification:**InformationChangeNotification** xor informationChangeNotification:**InformationWithRpChangeNotification**

Table A.5.2.3-2 shows the information elements used in the operations data related to virtualised compute resources information management, unless a reference is provided.

Table A.5.2.3-2: Information elements used in the virtualised compute resources information management operations data

Information element	Attribute:Type
VirtualComputeResourceInformation	virtualMemory:VirtualMemoryResourceInformation virtualCpu:VirtualCpuResourceInformation accelerationCapability:<not specified> computeResourceTypeId:Identifier
VirtualComputeResourceWithRpInfo	resourceProviderId:Identifier virtualMemory:VirtualMemoryResourceInformation virtualCpu:VirtualCpuResourceInformation accelerationCapability:<not specified> computeResourceTypeId:Identifier
InformationChangeNotification	changeId:Identifier resourceTypeId:Identifier vimId:Identifier changeType:*InformationChangeType* changedResourceData:<not specified>
InformationWith RpChangeNotification	resourceProviderId:Identifier changeId:Identifier resourceTypeId:Identifier vimId:Identifier changeType:*InformationChangeType* changedResourceData:<not specified>
VirtualMemoryResourceInformation	virtualMemSize:Number virtualMemOversubscriptionPolicy:<not specified> numaSupported:Boolean
VirtualCpuResourceInformation	cpuArchitecture:String numVirtualCpu:Number cpuClock:Number virtualCpuOversubscriptionPolicy:<not specified> virtualCpuPinningSupported:Boolean

A.5.2.4 Virtualised compute resources capacity management

Table A.5.2.4-1 shows the input/output parameters for each operation of the virtualised compute resources capacity management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.2.4-2, otherwise a reference is provided.

Table A.5.2.4-1: Virtualised compute resources capacity management operations

Operation	Or-Vi	Input/output Parameter:Type
Query Compute Capacity	X	zoneId:Identifier computeResourceTypeId:Identifier resourceCriteria:<not specified> attributeSelector:String timePeriod:**TimePeriodInformation** (see Table A.7.1-3)
		capacityResponse:**CapacityInformation**
Subscribe	X	zoneId:Identifier computeResourceTypeId:Identifier resourceCriteria:<not specified> threshold:**ResourceCapacityThreshold** (see Table A.5.4.4-2) attributeSelector:String
		capacityChangeSubscriptionId:Identifier
Query Compute Resource Zone	X	filter:Filter
		zoneInfo:**ResourceZone** (see Table A.7.3-2)
Query NFVI-PoP Compute Information	X	filter:Filter
		nfviInfo:**NfviPop** (see Table A.7.3-2)
Notify	X	capacityChangeNotification:**CapacityChangeNotification**

Table A.5.2.4-2 shows the information elements used in the operations data related to virtualised compute resources capacity management, unless a reference is provided.

Table A.5.2.4-2: Information elements used in the virtualised compute resources capacity management operations data

Information element	Attribute:Type
CapacityInformation	availableCapacity:<non specified> reservedCapacity:<non specified> totalCapacity:<non specified> allocatedCapacity:<non specified>
CapacityChangeNotification	changeId:Identifier zoneId:Identifier resourceDescriptor:<non specified> capacityInformation:**CapacityInformation**

A.5.2.5 Virtualised compute flavour management

Table A.5.2.5-1 shows the input/output parameters for each operation of the virtualised compute flavour management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.2.5-2, otherwise a reference is provided.

Table A.5.2.5-1: Virtualised compute flavour management operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Create Compute Flavour	X	X	flavour:VirtualComputeFlavour
			flavourId:Identifier
Query Compute Flavour	X	X	queryComputeFlavourFilter:Filter
			flavours:VirtualComputeFlavour
Delete Compute Flavour	X	X	computeFlavourId:Identifier
			none

Table A.5.2.5-2 shows the information elements used in the operations data related to virtualised compute flavour management, unless a reference is provided.

Table A.5.2.5-2: Information elements used in the virtualised compute flavour management operations data

Information element	Attribute:Type
VirtualComputeFlavour	flavourId:Identifier accelerationCapability:<not specified> virtualMemory:**VirtualMemoryData** virtualCpu:**VirtualCpuData** storageAttributes:**VirtualStorageData** virtualNetworkInterface:**VirtualNetworkInterfaceData** (see Table A.5.2.1-2)
VirtualMemoryData	virtualMemSize:Number virtualMemOversubscriptionPolicy:<not specified> numaEnabled:Boolean
VirtualCpuData	cpuArchitecture:String numVirtualCpu:Integer cpuClock:Number virtualCpuOversubscriptionPolicy:<not specified> virtualCpuPinning:**VirtualCpuPinningData**
VirtualStorageData	typeOfStorage:String sizeOfStorage:Number rdmaEnabled:Boolean
VirtualCpuPinningData	virtualCpuPinningPolicy:*CpuPinningPolicy* virtualCpuPinningRules:<not specified>

A.5.3 Virtualised network

A.5.3.1 Virtualised network resources management

Table A.5.3.1-1 shows the input/output parameters for each operation of the virtualised network resources management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.3.1-2, otherwise a reference is provided.

Table A.5.3.1-1: Virtualised network resources management operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Allocate Virtualised Network Resource	X	X	networkResourceName:String reservationId:Identifier networkResourceType:*NetworkResourceType* typeNetworkData:**VirtualNetworkData** typeNetworkPortData:**VirtualNetworkPortData** typeSubnetData:**NetworkSubnetData** (see Table A.5.3.1-2) affinityOrAntiAffinityConstraint:**AffinityOrAntiAffinityConstraint** (see Table A.5.2.1-2) locationConstraints:<not specified> metaData:KeyValuePair resourceGroupId:Identifier
			networkData:**VirtualNetwork** (see Table A.7.3-2) xor networkData:**NetworkResourceWithRpInfo** subnetData:**NetworkSubnet** (see Table A.7.3-2) networkPortData:**VirtualNetworkPort** (see Table A.7.3-2)
Query Virtualised Network Resource	X	X	queryNetworkFilter:Filter
			queryResult:**VirtualNetwork** (see Table A.7.3-2) xor queryResult:**NetworkResourceWithRpInfo**
Update Virtualised Network Resource	X	X	networkResourceId:Identifier updateNetworkData:**VirtualNetworkData** updateSubnetData:**NetworkSubnetData** (see Table A.5.3.1-2) updateNetworkPort:**VirtualNetworkPortData** metaData:KeyValuePair
			networkResourceId:Identifier networkData:**VirtualNetwork** (see Table A.7.3-2) xor networkData:**NetworkResourceWithRpInfo** subnetData:**NetworkSubnet** (see Table A.7.3-2) networkPortData:**VirtualNetworkPort** (see Table A.7.3-2)
Terminate Virtualised Network Resource	X	X	networkResourceId:Identifier xor networkResourceId:**NetworkResourceWithRpId**
			networkResourceId:Identifier xor networkResourceId:**NetworkResourceWithRpId**
Create Virtualised Network Resource Affinity Or AntiAffinity Constraints Group	X	X	groupName:String type:*AffinityOrAntiAffinity* scope:*AffinityOrAntiAffinityScope*
			groupId:Identifier

Table A.5.3.1-2 shows the information elements used in the operations data related to virtualised network resources management, unless a reference is provided.

Table A.5.3.1-2: Information elements used in the virtualised network resources management operations data

Information element	Attribute:Type
VirtualNetworkData	bandwidth:Number networkType:String segmentType:String networkQoS:**NetworkQoS** (see Table A.7.3-2) isShared:Boolean sharingCriteria:<not specified> layer3Attributes:**NetworkSubnetData** metadata:KeyValuePair
NetworkSubnetData	networkId:Identifier ipVersion:*IpVersion* gatewayIp:IpAddress cidr:<not specified> isDhcpEnabled:Boolean addressPool:<not specified> metadata:KeyValuePair
VirtualNetworkPortData	portType:String networkId:Identifier segmentId:Identifier bandwidth:Number metadata:KeyValuePair
NetworkResourceWithRpId	resourceProviderId:Identifier networkResourceId:Identifier
NetworkResourceWithRpInfo	resourceProviderId:Identifier networkResourceId:Identifier networkResourceName:String subnetId:Identifier networkPort:**VirtualNetworkPort** (see Table A.7.3-2) bandwidth:Number networkType:String segmentType:String networkQoS:**NetworkQoS** (see Table A.7.3-2) isShared:Boolean sharingCriteria:<not specified> zoneId:Identifier operationalState:*OperationalState* metadata:KeyValuePair

A.5.3.2 Virtualised network resources change notification

Table A.5.3.2-1 shows the input/output parameters for each operation of the virtualised network resources change notification interface, i.e. for each kind of notification and request-response.

Table A.5.3.2-1: Virtualised network resources change notification operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Subscribe	X	X	inputFilter:Filter
			subscriptionId:Identifier
Notify	X	X	virtualisedResourceChangeNotification:**VirtualisedResourceChangeNotification** (see Table A.5.2.2-2) xor virtualisedResourceWithRpChangeNotification:**VirtualisedResourceWithRpChangeNotification** (see Table A.5.2.2-2)

A.5.3.3 Virtualised network resources information management

Table A.5.3.3-1 shows the input/output parameters for each operation of the virtualised network resources information management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.3.3-2, otherwise a reference is provided.

Table A.5.3.3-1: Virtualised network resources information management operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Subscribe	X	X	filter:Filter
			subscriptionID:Identifier
Query Virtualised Network Resource Information	X	X	informationQueryFilter:Filter
			virtualisedResourceInformation:**VirtualNetworkResourceInformation** xor virtualisedResourceInformation:**VirtualNetworkResourceWithRpInfo**
Notify	X	X	informationChangeNotification:**InformationChangeNotification** (see Table A.5.2.3-2) xor informationChangeNotification:**InformationWithRpChangeNotification** (see Table A.5.2.3-2)

Table A.5.3.3-2 shows the information elements used in the operations data related to virtualised network resources information management, unless a reference is provided.

Table A.5.3.3-2: Information elements used in the virtualised network resources information management operations data

Information element	Attribute:Type
VirtualNetworkResourceInformation	bandwidth:Number networkType:String networkQoS:**NetworkQoS** (see Table A.7.3-2) networkResourceTypeId:Identifier
VirtualNetworkResourceWithRpInfo	resourceProviderId:Identifier bandwidth:Number networkType:String networkQoS:**NetworkQoS** (see Table A.7.3-2) networkResourceTypeId:Identifier

A.5.3.4 Virtualised network resources capacity management

Table A.5.3.4-1 shows the input/output parameters for each operation of the virtualised network resources capacity management interface, i.e. for each kind of notification and request-response.

Table A.5.3.4-1: Virtualised network resources capacity management operations

Operation	Or-Vi	Input/output Parameter:Type
Query Network Capacity	X	zoneId:Identifier networkResourceTypeId:Identifier resourceCriteria:<not specified> attributeSelector:String timePeriod:**TimePeriodInformation** (see Table A.7.1-3)
		capacityResponse:**CapacityInformation** (see Table A.5.2.4-2)
Subscribe	X	zoneId:Identifier networkResourceTypeId:Identifier resourceCriteria:<not specified> threshold:**ResourceCapacityThreshold** (see Table A.5.4.4-2) attributeSelector:String
		capacityChangeSubscriptionId:Identifier
Query NFVI-PoP Network Information	X	filter:Filter
		nfviInfo:**NfviPop** (see Table A.7.3-2)
Notify	X	capacityChangeNotification:**CapacityChangeNotification** (see Table A.5.2.4-2)

A.5.3.5 Network forwarding path management

Table A.5.3.5-1 shows the input/output parameters for each operation of the network forwarding path management interface, i.e. for each kind of notification and request-response.

Table A.5.3.5-1: Network forwarding path management operations

Operation	Or-Vi	Input/output Parameter:Type
Create NFP	X	virtualNetworkPortGroup:**VirtualNetworkPortGroup** (see Table A.7.3-1) totalVnp:Integer nfpRule:**NfpRule** (see Table A.4.2-2)
		nfpId:Identifier
Query NFP	X	queryFilter:Filter
		nfpResult:**Nfp** (see Table A.7.3-1)
Delete NFP	X	nfpId:Identifier
		deletedNfpId:Identifier
Change NFP State	X	nfpId:Identifier desiredState:*OperationalState*
		changedNfpId:Identifier
Update NFP	X	nfpId:Identifier nfpRule:**NfpRule** (see Table A.4.2-2)
		nfpInfo:**Nfp** (see Table A.7.3-1)

A.5.4 Virtualised storage

A.5.4.1 Virtualised storage resources management

Table A.5.4.1-1 shows the input/output parameters for each operation of the virtualised storage resources management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.4.1-2, otherwise a reference is provided.

Table A.5.4.1-1: Virtualised storage resources management operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Allocate Virtualised Storage Resource	X	X	storageName:String reservationId:Identifier affinityOrAntiAffinityConstraint:**AffinityOrAntiAffinityConstraint** (see Table A.5.2.1-2) storageData:**VirtualStorageFlavour** locationConstraints:<not specified> metaData:KeyValuePair resourceGroupId:Identifier
			storageResource:**VirtualStorage** (see Table A.7.3-2) xor storageResource:**StorageResourceWithRpInfo**
Query Virtualised Storage Resource	X	X	storageQueryFIlter:Filter
			queryResult:**VirtualStorage** (see Table A.7.3-2) xor queryResult:**StorageResourceWithRpInfo**
Update Virtualised Storage Resource	X	X	storageId:Identifier updateStorageData:**VirtualStorageFlavour** metaData:KeyValuePair
			storageId:Identifier storageData:**VirtualStorage** (see Table A.7.3-2) xor storageData:**StorageResourceWithRpInfo**
Terminate Virtualised Storage Resource	X	X	storageId:Identifier xor storageId:**StorageResourceWithRpId**
			storageId:Identifier xor storageId:**StorageResourceWithRpId**
Operate Virtualised Storage Resource	X	X	storageId:Identifier storageOperation:String storageOperationInputData:KeyValuePair
			storageData:**VirtualStorage** (see Table A.7.3-2) xor storageData:**StorageResourceWithRpInfo** storageOperationOutputData:KeyValuePair
Scale Virtualised Storage Resource	X	X	storageId:Identifier newSize:Number
			storageData:**VirtualStorage** (see Table A.7.3-2)xor storageData:**StorageResourceWithRpInfo**
Migrate Virtualised Storage Resource	X	X	storageId:Identifier affinityOrAntiAffinityConstraint:**AffinityOrAntiAffinityConstraint** (see Table A.5.2.1-2) migrationConstraint:<not specified>
			storageData:**VirtualStorage** (see Table A.7.3-2) xor storageData:**StorageResourceWithRpInfo**
Create Virtualised Storage Resource Affinity Or AntiAffinity Constraints Group	X	X	groupName:String type:*AffinityOrAntiAffinity* scope:*AffinityOrAntiAffinityScope*
			groupId:Identifier

Table A.5.4.1-2 shows the information elements used in the operations data related to virtualised storage resources management, unless a reference is provided.

Table A.5.4.1-2: Information elements used in the virtualised storage resources management operations data

Information element	Attribute:Type
VirtualStorageFlavour	flavourId:Identifier storageAttributes:**VirtualStorageData** (see Table A.5.2.5-2)
StorageResourceWithRpId	resourceProviderId:Identifier storageId:Identifier
StorageResourceWithRpInfo	resourceProviderId:Identifier storageId:Identifier storageName:String flavourId:Identifier typeOfStorage:String sizeOfStorage:Number rdmaEnabled:Boolean ownerId:Identifier zoneId:Identifier hostId:Identifier operationalState:*OperationalState* metadata:KeyValuePair

A.5.4.2 Virtualised storage resources change notification

Table A.5.4.2-1 shows the input/output parameters for each operation of the virtualised storage resources change notification interface, i.e. for each kind of notification and request-response.

Table A.5.4.2-1: Virtualised storage resources change notification operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Subscribe	X	X	inputFilter:Filter
			subscriptionId:Identifier
Notify	X	X	virtualisedResourceChangeNotification:**VirtualisedResourceChangeNotification** (see Table A.5.2.2-2) xor virtualisedResourceWithRpChangeNotification:**VirtualisedResourceWithRpChangeNotification** (see Table A.5.2.2-2)

A.5.4.3 Virtualised storage resources information management

Table A.5.4.3-1 shows the input/output parameters for each operation of the virtualised storage resources information management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.4.3-2, otherwise a reference is provided.

Table A.5.4.3-1: Virtualised storage resources information management operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Subscribe	X	X	filter:Filter
			subscriptionID:Identifier
Query Virtualised Storage Resource Information	X	X	informationQueryFilter:Filter
			virtualisedResourceInformation:**VirtualStorageResourceInformation** xor virtualisedResourceInformation:**VirtualStorageResourceWithRpInfo**
Notify	X	X	informationChangeNotification:**InformationChangeNotification** (see Table A.5.2.3-2) xor informationChangeNotification:**InformationWithRpChangeNotification** (see Table A.5.2.3-2)

Table A.5.4.3-2 shows the information elements used in the operations data related to virtualised storage resources information management, unless a reference is provided.

Table A.5.4.3-2: Information elements used in the virtualised storage resources information management operations data

Information element	Attribute:Type
VirtualStorageResourceInformation	typeOfStorage:String sizeOfStorage:Number rdmaSupported:Boolean storageResourceTypeId:Identifier
VirtualStorageResourceWithRpInfo	resourceProviderId:Identifier typeOfStorage:String sizeOfStorage:Number rdmaSupported:Boolean storageResourceTypeId:Identifier

A.5.4.4 Virtualised storage resources capacity management

Table A.5.4.4-1 shows the input/output parameters for each operation of the virtualised storage resources capacity management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.4.4-2, otherwise a reference is provided.

Table A.5.4.4-1: Virtualised storage resources capacity management operations

Operation	Or-Vi	Input/output Parameter:Type
Query Storage Capacity	X	zoneId:Identifier storageResourceTypeId:Identifier resourceCriteria:<not specified> attributeSelector:String timePeriod:**TimePeriodInformation** (see Table A.7.1-3)
		capacityResponse:**CapacityInformation** (see Table A.5.2.4-2)
Subscribe	X	zoneId:Identifier storageResourceTypeId:Identifier resourceCriteria:<not specified> threshold:**ResourceCapacityThreshold** attributeSelector:String
		capacityChangeSubscriptionId:Identifier
Query NFVI-PoP Storage Information	X	filter:Filter
		nfviPop:**NfviPop** (see Table A.7.3-2)
Query Storage Resource Zone	X	filter:Filter
		zoneInfo:**ResourceZone** (see Table A.7.3-2)
Notify	X	capacityChangeNotification:**CapacityChangeNotification** (see Table A.5.2.4-2)

Table A.5.4.4-2 shows the information elements used in the operations data related to virtualised storage resources capacity management, unless a reference is provided.

Table A.5.4.4-2: Information elements used in the virtualised storage resources capacity management operations data

Information element	Attribute:Type
ResourceCapacityThreshold	thresholdType:*ThresholdType* threshold:<not specified>

A.5.5 Virtualised resource reservation

A.5.5.1 Virtualised compute resources reservation management

Table A.5.5.1-1 shows the input/output parameters for each operation of the virtualised compute resources reservation management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.5.1-2, otherwise a reference is provided.

Table A.5.5.1-1: Virtualised compute resources reservation management operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Create Compute Resource Reservation	X		computePoolReservation:**ComputePoolReservation** virtualisationContainerReservation:**VirtualisationContainerReservation** affinityConstraint:**AffinityOrAntiAffinityConstraint** (see Table A.5.2.1-2) antiAffinityConstraint:**AffinityOrAntiAffinityConstraint** (see Table A.5.2.1-2) startTime:DateTime endTime:DateTime expiryTime:DateTime locationConstraints:<not specified> resourceGroupId:Identifier
			reservationData:**ReservedVirtualCompute** xor reservationData:**ReservedVirtualComputeWithRpInfo**
Query Compute Resource Reservation	X	X	queryReservationFilter:Filter
			queryResult:**ReservedVirtualCompute** xor queryResult:**ReservedVirtualComputeWithRpInfo**
Update Compute Resource Reservation	X		reservationId:Identifier computePoolReservation:**ComputePoolReservation** virtualisationContainerReservation:**VirtualisationContainerReservation** startTime:DateTime endTime:DateTime expiryTime:DateTime
			reservationData:**ReservedVirtualCompute** xor reservationData:**ReservedVirtualComputeWithRpInfo**
Terminate Compute Resource Reservation	X		reservationId:Identifier
			reservationId:Identifier

Table A.5.5.1-2 shows the information elements used in the operations data related to virtualised compute resources reservation management, unless a reference is provided.

Table A.5.5.1-2: Information elements used in the virtualised compute resources reservation management operations data

Information element	Attribute:Type
ComputePoolReservation	numCpuCores:Integer numVcInstances:Integer virtualMemSize:Number computeAttributes:**VirtualComputeAttributesReservationData**
ReservedVirtualCompute	reservationId:Identifier computePoolReserved:**ReservedComputePool** virtualisationContainerReserved:**ReservedVirtualisationContainer** reservationStatus:*ReservationStatus* startTime:DateTime endTime:DateTime expiryTime:DateTime
ReservedVirtualComputeWithRpInfo	resourceProviderId:Identifier reservationId:Identifier computePoolReserved:**ReservedComputePool** virtualisationContainerReserved:**ReservedVirtualisationContainer** reservationStatus:*ReservationStatus* startTime:DateTime endTime:DateTime expiryTime:DateTime
VirtualisationContainerReservation	containerId:Identifier containerFlavour:**VirtualComputeFlavour** (see Table A.5.2.5-2)
VirtualComputeAttributesReservationData	accelerationCapability:<not specified> cpuArchitecture:<not specified> virtualCpuOversubscriptionPolicy:<not specified>
ReservedComputePool	numCpuCores:Integer numVcInstances:Integer virtualMemSize:Number computeAttributes:**ReservedVirtualComputeAttributes** zoneId:Identifier
ReservedVirtualComputeAttributes	accelerationCapability:<not specified> cpuArchitecture:<not specified> virtualCpuOversubscriptionPolicy:<not specified>
ReservedVirtualisationContainer	containerId:Identifier flavourId:Identifier accelerationCapability:<not specified> virtualMemory:**VirtualMemory** (see Table A.7.3-2) virtualCpu:**VirtualCpu** (see Table A.7.3-2) virtualDisks:**VirtualStorage** (see Table A.7.3-2) virtualNetworkInterface:**VirtualNetworkInterface** (see Table A.7.3-2) zoneId:Identifier

A.5.5.2 Virtualised network resources reservation management

Table A.5.5.2-1 shows the input/output parameters for each operation of the virtualised network resources reservation management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.5.2-2, otherwise a reference is provided.

Table A.5.5.2-1: Virtualised network resources reservation management operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Create Network Resource Reservation	X		networkReservation:VirtualNetworkReservation startTime:DateTime endTime:DateTime expiryTime:DateTime affinityConstraint:AffinityOrAntiAffinityConstraint (see Table A.5.2.1-2) antiAffinityConstraint:AffinityOrAntiAffinityConstraint (see Table A.5.2.1-2) locationConstraints:<not specified> resourceGroupId:Identifier
			reservationData:ReservedVirtualNetwork xor reservationData:ReservedVirtualNetworkWithRpInfo
Query Network Resource Reservation	X	X	queryReservationFilter:Filter
			queryResult:ReservedVirtualNetwork xor queryResult:ReservedVirtualNetworkWithRpInfo
Update Network Resource Reservation	X		reservationId:Identifier networkReservation:VirtualNetworkReservation startTime:DateTime endTime:DateTime expiryTime:DateTime
			reservationData:ReservedVirtualNetwork xor reservationData:ReservedVirtualNetworkWithRpInfo
Terminate Network Resource Reservation	X		reservationId:Identifier
			reservationId:Identifier

Table A.5.5.2-2 shows the information elements used in the operations data related to virtualised network resources reservation management, unless a reference is provided.

Table A.5.5.2-2: Information elements used in the virtualised network resources reservation management operations data

Information element	Attribute:Type
ReservedVirtualNetwork	reservationId:Identifier publicIpAddresses:**ReservedPublicIpAddresses** networkAttributes:**ReservedVirtualNetworkAttributes** networkPorts:**ReservedVirtualNetworkPort** reservationStatus:*ReservationStatus* startTime:DateTime endTime:DateTime expiryTime:DateTime zoneId:Identifier
ReservedVirtualNetworkWithRpInfo	resourceProviderId:Identifier reservationId:Identifier networkAttributes:**ReservedVirtualNetworkAttributes** networkPorts:**ReservedVirtualNetworkPort** reservationStatus:*ReservationStatus* startTime:DateTime endTime:DateTime expiryTime:DateTime zoneId:Identifier publicIpAddresses:**ReservedPublicIpAddresses**
VirtualNetworkReservation	networkAttributes:**VirtualNetworkAttributesReservationData** networkPorts:**VirtualNetworkPortReservationData** publicIpAddresses:**PublicIpAddressesReservationData**
ReservedPublicIpAddresses	networkId:Identifier publicIps:IpAddress
ReservedVirtualNetworkAttributes	bandwidth:Number networkType:String segmentType:String isShared:Boolean metadata:KeyValuePair
ReservedVirtualNetworkPort	portId:Identifier portType:<not specified> segmentId:Identifier bandwidth:Number metadata:KeyValuePair
VirtualNetworkAttributesReservationData	bandwidth:Number networkType:String segmentType:String isShared:Boolean metadata:KeyValuePair
VirtualNetworkPortReservationData	portId:Identifier portType:<not specified> segmentId:Identifier bandwidth:Number metadata:KeyValuePair
PublicIpAddressesReservationData	numPublicIps:Integer networkId:Identifier publicIps:IpAddress

A.5.5.3 Virtualised storage resources reservation management

Table A.5.5.3-1 shows the input/output parameters for each operation of the virtualised storage resources reservation management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.5.3-2, otherwise a reference is provided.

Table A.5.5.3-1: Virtualised storage resources reservation management operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Create Storage Resource Reservation	X		storagePoolReservation:**StoragePoolReservation** startTime:DateTime endTime:DateTime expiryTime:DateTime affinityConstraint:**AffinityOrAntiAffinityConstraint** (see Table A.5.2.1-2) antiAffinityConstraint:**AffinityOrAntiAffinityConstraint** (see Table A.5.2.1-2) locationConstraints:<not specified> resourceGroupId:Identifier
			reservationData:**ReservedVirtualStorage** xor reservationData:**ReservedVirtualStorageWithRpInfo**
Query Storage Resource Reservation	X	X	queryReservationFilter:Filter
			queryResult:**ReservedVirtualStorage** xor queryResult:**ReservedVirtualStorageWithRpInfo**
Update Storage Resource Reservation	X		reservationId:Identifier storagePoolReservation:**StoragePoolReservation** startTime:DateTime endTime:DateTime expiryTime:DateTime
			reservationData:**ReservedVirtualStorage** xor reservationData:**ReservedVirtualStorageWithRpInfo**
Terminate Storage Resource Reservation	X		reservationId:Identifier
			reservationId:Identifier

Table A.5.5.3-2 shows the information elements used in the operations data related to virtualised storage resources reservation management, unless a reference is provided.

Table A.5.5.3-2: Information elements used in the virtualised storage resources reservation management operations data

Information element	Attribute:Type
ReservedVirtualStorage	reservationId:Identifier storagePoolReserved:**ReservedStoragePool** reservationStatus:*ReservationStatus* startTime:DateTime endTime:DateTime expiryTime:DateTime
ReservedVirtualStorageWithRpInfo	resourceProviderId:Identifier reservationId:Identifier storagePoolReserved:**ReservedStoragePool** reservationStatus:*ReservationStatus* startTime:DateTime endTime:DateTime expiryTime:DateTime
StoragePoolReservation	storageSize:Number numSnapshots:Integer numVolumes:Integer
ReservedStoragePool	storageSize:Number numSnapshots:Integer numVolumes:Integer zoneId:Identifier

A.5.5.4 Virtualised resources reservation change notification

Table A.5.5.4-1 shows the input/output parameters for each operation of the virtualised resources reservation change notification interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.5.4-2, otherwise a reference is provided.

Table A.5.5.4-1: Virtualised resources reservation change notification operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Subscribe	X	X	inputFilter:Filter
			inputFilter:Identifier
Notify	X	X	virtualisedResourceReservationChangeNotification: **VirtualisedResourceReservationChangeNotification** xor virtualisedResourceReservationWithRpChangeNotification: **VirtualisedResourceReservationWithRpChange Notification**

Table A.5.5.4-2 shows the information elements used in the operations data related to virtualised resources reservation change notification, unless a reference is provided.

Table A.5.5.4-2: Information elements used in the virtualised resources reservation change notification operations data

Information element	Attribute:Type
VirtualisedResourceReservationChangeNotification	changeId:Identifier reservationId:Identifier vimId:Identifier changeType:String changedReservationData:<not specified>
VirtualisedResourceReservationWithRpChangeNotification	resourceProviderId:Identifier changeId:Identifier reservationId:Identifier vimId:Identifier changeType:String changedReservationData:<not specified>

A.5.6 Virtualised resource quota

A.5.6.1 Virtualised compute resources quota management

Table A.5.6.1-1 shows the input/output parameters for each operation of the virtualised compute resources quota management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.6.1-2, otherwise a reference is provided.

Table A.5.6.1-1: Virtualised compute resources quota management operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Create Compute Resource Quota	X		resourceGroupId:Identifier virtualComputeQuota:**VirtualComputeQuotaData**
			quotaData:**VirtualComputeQuota** xor quotaData:**VirtualComputeQuotaWithRpInfo**
Query Compute Resource Quota	X	X	queryQuotaFilter:Filter
			queryResult:**VirtualComputeQuota** xor queryResult:**VirtualComputeQuotaWithRpInfo**
Update Compute Resource Quota	X		resourceGroupId:Identifier virtualComputeQuota:**VirtualComputeQuotaData**
			quotaData:**VirtualComputeQuota** xor quotaData:**VirtualComputeQuotaWithRpInfo**
Terminate Compute Resource Quota	X		resourceGroupId:Identifier
			resourceGroupId:Identifier

Table A.5.6.1-2 shows the information elements used in the operations data related to virtualised compute resources quota management, unless a reference is provided.

Table A.5.6.1-2: Information elements used in the virtualised compute resources quota management operations data

Information element	Attribute:Type
VirtualComputeQuota	resourceGroupId:Identifier numVCPUs:Integer numVcInstances:Integer virtualMemSize:Number
VirtualComputeQuotaWithRpInfo	resourceProviderId:Identifier resourceGroupId:Identifier numVCPUs:Integer numVcInstances:Integer virtualMemSize:Number
VirtualComputeQuotaData	numVCPUs:Integer numVcInstances:Integer virtualMemSize:Number

A.5.6.2 Virtualised network resources quota management

Table A.5.6.2-1 shows the input/output parameters for each operation of the virtualised network resources quota management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.6.2-2, otherwise a reference is provided.

Table A.5.6.2-1: Virtualised network resources quota management operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Create Network Resource Quota	X		resourceGroupId:Identifier virtualNetworkQuota:**VirtualNetworkQuotaData**
			quotaData:**VirtualNetworkQuota** xor quotaData:**VirtualNetworkQuotaWithRpInfo**
Query Network Resource Quota	X	X	queryQuotaFilter:Filter
			queryResult:**VirtualNetworkQuota** xor queryResult:**VirtualNetworkQuotaWithRpInfo**
Update Network Resource Quota	X		resourceGroupId:Identifier virtualNetworkQuota:**VirtualNetworkQuotaData**
			quotaData:**VirtualNetworkQuota xor** quotaData:**VirtualNetworkQuotaWithRpInfo**
Terminate Network Resource Quota	X		resourceGroupId:Identifier
			resourceGroupId:Identifier

Table A.5.6.2-2 shows the information elements used in the operations data related to virtualised network resources quota management, unless a reference is provided.

Table A.5.6.2-2: Information elements used in the virtualised network resources quota management operations data

Information element	Attribute:Type
VirtualNetworkQuota	resourceGroupId:Identifier numPublicIps:Integer numPorts:Integer numSubnets:Integer
VirtualNetworkQuotaWithRpInfo	resourceProviderId:Identifier resourceGroupId:Identifier numPublicIps:Integer numPorts:Integer numSubnets:Integer
VirtualNetworkQuotaData	numPublicIps:Integer numPorts:Integer numSubnets:Integer

A.5.6.3 Virtualised storage resources quota management

Table A.5.6.3-1 shows the input/output parameters for each operation of the virtualised storage resources quota management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.6.3-2, otherwise a reference is provided.

Table A.5.6.3-1: Virtualised storage resources quota management operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Create Storage Resource Quota	X		resourceGroupId:Identifier virtualStorageQuota:**VirtualStorageQuotaData**
			quotaData:**VirtualStorageQuota** xor quotaData:**VirtualStorageQuotaWithRpInfo**
Query Storage Resource Quota	X	X	queryQuotaFilter:Filter
			queryResult:**VirtualStorageQuota** xor queryResult:**VirtualStorageQuotaWithRpInfo**
Update Storage Resource Quota	X		resourceGroupId:Identifier virtualStorageQuota:**VirtualStorageQuotaData**
			quotaData:**VirtualStorageQuota** xor quotaData:**VirtualStorageQuotaWithRpInfo**
Terminate Storage Resource Quota	X		resourceGroupId:Identifier
			resourceGroupId:Identifier

Table A.5.6.3-2 shows the information elements used in the operations data related to virtualised storage resources quota management, unless a reference is provided.

Table A.5.6.3-2: Information elements used in the virtualised storage resources quota management operations data

Information element	Attribute:Type
VirtualStorageQuota	resourceGroupId:Identifier storageSize:Number numSnapshots:Integer numVolumes:Integer
VirtualStorageQuotaWithRpInfo	resourceProviderId:Identifier resourceGroupId:Identifier storageSize:Number numSnapshots:Integer numVolumes:Integer
VirtualStorageQuotaData	storageSize:Number numSnapshots:Integer numVolumes:Integer

A.5.6.4 Virtualised resources quota change notification

Table A.5.6.4-1 shows the input/output parameters for each operation of the virtualised resources quota change notification interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.6.4-2, otherwise a reference is provided.

Table A.5.6.4-1: Virtualised resources quota change notification operations

Operation	Or-Vi	Vi-Vnfm	Input/output Parameter:Type
Subscribe	X	X	inputFilter:Filter
			subscriptionId:Identifier
Notify	X	X	virtualisedResourceQuotaChangeNotification: **VirtualisedResourceQuotaChangeNotification** xor virtualisedResourceQuotaWithRpChangeNotification:**VirtualisedResourceQuotaWithRpChangeNotification**

Table A.5.6.4-2 shows the information elements used in the operations data related to virtualised resources quota change notification, unless a reference is provided.

Table A.5.6.4-2: Information elements used in the virtualised resources quota change notification operations data

Information element	Attribute:Type
VirtualisedResourceQuotaChangeNotification	changeId:Identifier resourceGroupId:Identifier vimId:Identifier changeType:String changedQuotaData:<not specified>
VirtualisedResourceQuotaWithRpChangeNotification	resourceProviderId:Identifier changeId:Identifier resourceGroupId:Identifier vimId:Identifier changeType:String changedQuotaData:<not specified>

A.5.7 NFVI capacity information

Table A.5.7-1 shows the input/output parameters for each operation of the NFVI capacity information interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.7-2, otherwise a reference is provided.

Table A.5.7-1: NFVI capacity information operations

Operation	Os-Ma-Nfvo	Input/output Parameter:Type
Query NFVI capacity	X	filter:Filter
		capacityResponse:<not specified>
Subscribe	X	filter:Filter
		subscriptionId:Identifier
Terminate Subscription	X	subscriptionId:Identifier
		none
Query Subscription Info	X	filter:Filter
		queryResult:<not specified>
Create Capacity Threshold	X	thresholdType:*ThresholdType* thresholdDetails:<not specified>
		thresholdId:Identifier
Delete Capacity Thresholds	X	thresholdId:Identifier
		deletedThresholdId:Identifier
Query Capacity Threshold	X	filter:Filter
		thresholdDetails:**NfviCapacityThreshold**
Notify	X	capacityThresholdCrossedNotification: **CapacityThresholdCrossedNotification**

Table A.5.7-2 shows the information elements used in the operations data related to NFVI capacity information, unless a reference is provided.

Table A.5.7-2: Information elements used in the NFVI capacity information operations data

Information element	Attribute:Type
NfviCapacityThreshold	thresholdId:Identifier objectInstanceId:Identifier thresholdType:*ThresholdType* thresholdDetails:<not specified>
CapacityThresholdCrossedNotification	subscriptionId:Identifier resourceZoneId:Identifier vimId:Identifier direction:*ThresholdCrossing* capacityInformation:<not specified>

A.5.8 Multi-site Connectivity Services (MSCS)

A.5.8.1 MSCS management

Table A.5.8.1-1 shows the input/output parameters for each operation of the Multi-Site Connectivity Services (MSCS) management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.8.1-2, otherwise a reference is provided.

Table A.5.8.1-1: MSCS management operations

Operation	Consumer-WIM	Input/output Parameter:Type
Create MSCS	X	reservationId:Identifier mscsData:**MscsData**
		mscs:**Mscs**
Query MSCS	X	filter:Filter attributeSelector:String
		queryResult:**Mscs**
Update MSCS	X	connectivityServiceId:Identifier addMscsEndpoint:**MscsEndpointData** removeMscsEndpoint:Identifier modifyMscsEndpoint:**MscsEndpointInfo** modifyMscsProfile:**MscsProfile** mscsName:String mscsDescription:String
		mscs:**Mscs**
Terminate MSCS	X	connectivityServiceId:Identifier
		connectivityServiceId:Identifier
Subscribe	X	filter:Filter
		subscriptionId:Identifier
Query Subscription Info	X	filter:Filter
		queryResult:<not specified>
Terminate Subscription	X	subscriptionId:Identifier
		none
Create MSCS Reservation	X	mscsReservation:**MscsReservationData** startTime:DateTime endTime:DateTime expiryTime:DateTime
		reservationData:**ReservedMscs**
Query MSCS Reservation	X	queryReservationFilter:Filter attributeSelector:String
		queryResult:**ReservedMscs**
Update MSCS Reservation	X	reservationId:Identifier addEndpoint:Identifier removeEndpoint:Identifier modifyMscsProfile:**MscsProfile** startTime:DateTime endTime:DateTime expiryTime:DateTime
		reservationData:**ReservedMscs**
Terminate MSCS Reservation operation	X	reservationId:Identifier
		reservationId:Identifier
Notify	X	mscsChangeNotification:**MscsChangeNotification** xor mscsReservationChangeNotification:**MscsReservation ChangeNotification**

Table A.5.8.1-2 shows the information elements used in the operations data related to MSCS management, unless a reference is provided.

Table A.5.8.1-2: Information elements used in the MSCS management operations data

Information element	Attribute:Type
Mscs	mscsId:Identifier mscsName:String mscsDescription:String mscsEndpoint:**MscsEndpointInfo** mscsProfile:**MscsProfile** msnc:**Msnc**
MscsEndpointInfo	mscsEndpointId:Identifier connectivityServiceEndpointId:Identifier directionality:*Directionality* networkAddressing:<not specified> lag:<not specified>
MscsProfile	bandwidthIn:Number bandwidthOut:Number qosMetric:<not specified> directionality:*Directionality* mtu:Number protectionScheme:*ProtectionScheme* connectivityMode:*ConnectivityMode* numSegment:Number segmentId:Identifier
Msnc	msncId:Identifier msncEndpointId:Identifier path:**NodeInfo** msncProfile:**MsncProfile** msncLayerProtocol:<not specified>
MsncProfile	bandwidthIn:Number bandwidthOut:Number qosMetric:<not specified> directionality:*MSNCDirectionality* mtu:Number protectionScheme:*ProtectionScheme* connectionMode:*ConnectivityMode*
ReservedMscs	reservationId:Identifier mscsLayerProtocol:<not specified> connectivityServiceEndpointId:Identifier mscsProfile:**MscsProfile** reservationStatus:<not specified> startTime:DateTime endTime:DateTime expiryTime:DateTime
MscsChangeNotification	mscsId:Identifier changedMscsProfile:<not specified> affectedComponent:<not specified>
MscsReservationChangeNotification	reservationId:Identifier changedMscsProfile:<not specified> affectedComponent:<not specified> changedTime:KeyValuePair reservationStatus:<not specified>
MscsData	mscsName:String mscsDescription:String mscsEndpoint:**MscsEndpointData** mscsProfile:**MscsProfile** mscsLayerProtocol:<not specified>
MscsReservationData	mscsLayerProtocol:<not specified> connectivityServiceEndpointId:Identifier mscsProfile:**MscsProfile**
NodeInfo	nodeId:Identifier layerProtocol:*LayerProtocol* transferCapability:<not specified> networkEdgePointId:Identifier
MscsEndpointData	connectivityServiceEndpointId:Identifier directionality:*Directionality* networkAddressing:<not specified> lag:<not specified>

A.5.8.2 MSCS capacity management

Table A.5.8.2-1 shows the input/output parameters for each operation of the MSCS capacity management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.8.2-2, otherwise a reference is provided.

Table A.5.8.2-1: MSCS capacity management operations

Operation	**Consumer-WIM**	**Input/output Parameter:Type**
Query Capacity	X	filter:Filter attributeSelector:String timePeriod:**TimePeriodInformation** (see Table A.7.1-3)
		capacityInfo:**CapacityInfo**
Create Capacity Threshold	X	objectInstanceId:Identifier thresholdType:*ThresholdType* thresholdDetails:<not specified>
		thresholdId:Identifier
Delete Capacity Thresholds	X	thresholdId:Identifier
		deletedThresholdId:Identifier
Query Capacity Threshold	X	filter:Filter
		thresholdDetails:**NetworkCapacityThreshold**
Query Topology Information	X	filter:Filter attributeSelector:String
		topologyInfo:**TopologyInfo**
Query Node Information	X	filter:Filter attributeSelector:String
		nodeInfo:**NodeInfo** (see Table A.5.8.1-2)
Query Link Information	X	filter:Filter attributeSelector:String
		linkInfo:**LinkInfo**
Query Network Edge Point Information	X	filter:Filter attributeSelector:String
		edgePointInfo:**NetworkEdgePointInfo**
Subscribe	X	filter:Filter
		subscriptionId:Identifier
Terminate Subscription	X	subscriptionId:Identifier
		none
Query Subscription	X	filter:Filter
		queryResult:<not specified>
Notify	X	networkCapacityChangeNotification:**NetworkCapacityChangeNotification** xor topologyChangeNotification:**TopologyChangeNotification**

Table A.5.8.2-2 shows the information elements used in the operations data related to MSCS capacity management, unless a reference is provided.

Table A.5.8.2-2: Information elements used in the MSCS capacity management operations data

Information element	Attribute:Type
CapacityInfo	objectType:*ObjectType* objectInstanceId:Identifier capacityValue:**CapacityValueEntry**
NetworkCapacityThreshold	thresholdId:Identifier thresholdType:*ThresholdType* thresholdDetails:<not specified> objectInstanceId:Identifier
TopologyInfo	topologyId:Identifier layerProtocol:*LayerProtocol* nodeId:Identifier linkId:Identifier networkEdgePointId:Identifier
LinkInfo	linkId:Identifier nodeId:**NodeInfo** (see Table A.5.8.1-2) isNetworkEdgeLink:Boolean networkEdgePointId:**NetworkEdgePointInfo** connectivityServiceEndpointId:Identifier
NetworkCapacityChangeNotification	thresholdId:Identifier crossingDirection:*ThresholdCrossing* objectInstanceId:Identifier capacityValueEntry:**CapacityValueEntry**
TopologyChangeNotification	networkId:Identifier changedInfo:<not specified>
CapacityValueEntry	capacityMetricName:String capacityValue:Value
NetworkEdgePointInfo	networkEdgePointId:Identifier layerProtocol:<not specified>
ConnectivityServiceEndpointInfo	connectivityServiceEndpointId:Identifier layerProtocol:<not specified> linkId:Identifier
NetworkInfo	networkId:Identifier topology:**TopologyInfo** node:**NodeInfo** (see Table A.5.8.1-2) link:**LinkInfo**

A.5.9 Compute host reservation management

Table A.5.9-1 shows the input/output parameters for each operation of the compute host reservation management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.5.9-2, otherwise a reference is provided.

Table A.5.9-1: Compute host reservation management operations

Operation	Or-Vi	Input/output Parameter:Type
Create Compute Host Reservation	X	minAmount:Integer maxAmount:Integer startTime:DateTime endTime:DateTime computeHostProperties:<not specified> locationConstraints:<not specified>
		reservationData:**ReservedComputeHosts**
Query Compute Host Reservation	X	queryReservationFilter:Filter
		queryResult:**ReservedComputeHosts**
Update Compute Host Reservation	X	reservationId:Identifier minAmount:Integer maxAmount:Integer startTime:DateTime endTime:DateTime computeHostProperties:<not specified> locationConstraints:<not specified>
		reservationData:**ReservedComputeHosts**
Terminate Compute Host Reservation operation	X	reservationId:Identifier
		reservationId:Identifier

Table A.5.9-2 shows the information elements used in the operations data related to the compute host reservation management, unless a reference is provided.

Table A.5.9-2: Information elements used in the compute host reservation management operations data

Information element	Attribute:Type
ReservedComputeHosts	reservationId:Identifier minAmount:Integer maxAmount:Integer startTime:DateTime endTime:DateTime reservationStatus:*ReservationStatus* computeHostProperties:<not specified> zoneId:Identifier

A.5.10 Compute host capacity management

Table A.5.10-1 shows the input/output parameters for each operation of the compute host capacity management interface, i.e. for each kind of notification and request-response.

Table A.5.10-1: Compute host capacity management operations

Operation	Or-Vi	Input/output Parameter:Type
Query Compute Host Capacity	X	inputFilter:Filter
		capacityResponse:<not specified>
Subscribe	X	inputFilter:Filter
		subscriptionId:Identifier
Notify	X	capacityChangeNotification:**CapacityChangeNotification** (see Table A.5.2.4-2)

A.6 VNF domain

A.6.1 Virtualised resources quota available notification

Table A.6.1-1 shows the input/output parameters for each operation of the virtualised resources quota available notification interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.6.1-2, otherwise a reference is provided.

Table A.6.1-1: Virtualised resources quota available notification operations

Operation	Or-Vnfm	Input/output Parameter:Type
Subscribe	X	filter:Filter
		subscriptionId:Identifier
Terminate Subscription	X	subscriptionId:Identifier
		none
Query Subscription Info	X	filter:Filter
		queryResult:<not specified>
Notify	X	virtualisedResourceQuotaAvailableNotification:**VirtualisedResourceQuotaAvailableNotification**

Table A.6.1-2 shows the information elements used in the operations data related to virtualised resources quota available notification, unless a reference is provided.

Table A.6.1-2: Information element used in the virtualised resources quota available notification operations data

Information element	Attribute:Type
VirtualisedResourceQuotaAvailableNotification	resourceGroupId:Identifier vimConnectionInfo:**VimConnectionInfo** (see Table A.7.1-4) resourceProviderId:Identifier

A.6.2 VNF configuration interface

Table A.6.2-1 shows the input/output parameters for the operation of the VNF configuration interface. The information elements used in the operations data are shown in Table A.6.2-2, otherwise a reference is provided.

Table A.6.2-1: VNF configuration operations

Operation	Ve-Vnfm	Input/output Parameter:Type
Set Configuration	X	vnfInstanceId:Identifier vnfConfigurationData:**VnfConfiguration** vnfcConfigurationData:**VnfcConfiguration**
		vnfConfigurationData:**VnfConfiguration** vnfcConfigurationData:**VnfcConfiguration**

Table A.6.2-2 shows the information elements used in the operations data related to VNF configuration, unless a reference is provided.

Table A.6.2-2: Information elements used in the VNF configuration operations data

Information element	Attribute:Type
VnfConfiguration	cpConfiguration:**CpConfiguration** dhcpServer:<not specified> vnfSpecificData:KeyValuePair
VnfcConfiguration	vnfcId:Identifier cp:**CpConfiguration** dhcpServer:<not specified> vnfcSpecificData:KeyValuePair
CpConfiguration	cpId:Identifier cpLabel:<not specified> address:**CpAddress**
CpAddress	layerProtocol:*LayerProtocol* address:<not specified> useDynamicAddress:Boolean port:<not specified>

A.6.3 VNF lifecycle operation granting

Table A.6.3-1 shows the input/output parameters for the operation of the VNF lifecycle operation granting interface. The information elements used in the operations data are shown in Table A.6.3-2, otherwise a reference is provided.

Table A.6.3-1: VNF lifecycle operation granting operation

Operation	Or-Vnfm	Input/output Parameter:Type
Grant VNF Lifecycle Operation	X	vnfInstanceId:Identifier vnfdId:Identifier dstVnfdId:Identifier flavourId:Identifier lifecycleOperation:*VnfLifecycleOperation* isAutomaticInvocation:Boolean lifecycleOperationOccurrenceId:Identifier instantiationLevelId:Identifier addResource:**ResourceDefinition** tempResource:**ResourceDefinition** removeResource:**ResourceDefinition** updateResource:**ResourceDefinition** placementConstraint:**PlacementConstraint** vimConstraint:**VimConstraint** additionalParam:KeyValuePair
		vimConnection:**VimConnectionInfo** (see Table A.7.1-4) zone:**ZoneInfo** zoneGroup:**ZoneGroupInfo** addResource:**GrantInfo** tempResource:**GrantInfo** removeResource:**GrantInfo** updateResource:**GrantInfo** vimAssets:**VimAssets** extVirtualLink:**ExtVirtualLinkData** (see Table A.7.1-4) extManagedVirtualLink:**ExtManagedVirtualLinkData** (see Table A.7.1-4) additionalParam:KeyValuePair

Table A.6.3-2 shows the information elements used in the operations data related to VNF lifecycle operation granting, unless a reference is provided.

Table A.6.3-2: Information elements used in the VNF lifecycle operation granting operations data

Information element	Attribute:Type
GrantInfo	resourceDefinitionId:Identifier vimConnectionId:Identifier resourceProviderId:Identifier zoneId:Identifier resourceGroupId:Identifier reservationId:Identifier
ResourceDefinition	resourceDefinitionId:Identifier type:*ResourceDefinitionType* vduId:Identifier resourceTemplateId:Identifier resourceHandle:**ResourceHandle** (see Table A.7.1-4) vnfdId:Identifier snapshotResDef:**SnapshotResourceDefinition**
VimAssets	computeResourceFlavour:**VimComputeResourceFlavour** softwareImage:**VimSoftwareImage** snapshotResource:**VimSnapshotResource**
ZoneGroupInfo	zoneId:Identifier
ZoneInfo	zoneInfoId:Identifier zoneId:Identifier vimConnectionId:Identifier resourceProviderId:Identifier
PlacementConstraint	affinityOrAntiAffinity:*AffinityOrAntiAffinity* scope:*AffinityOrAntiAffinityScope* resource:**ConstraintResourceRef** fallbackBestEffort:Boolean
VimConstraint	sameResourceGroup:Boolean resource:**ConstraintResourceRef**
SnapshotResourceDefinition	vnfSnapshotId:Identifier vnfcSnapshotId:Identifier storageSnapshotId:Identifier snapshotResource:**ResourceHandle** (see Table A.7.1-4)
VimSnapshotResource	resourceProviderId:Identifier vnfSnapshotId:Identifier vnfcSnapshotId:Identifier vimSnapshotResourceId:Identifier storageSnapshotId:Identifier vimConnectionId:Identifier
ConstraintResourceRef	idType:*ConstraintResourceReferenceType* resourceId:Identifier vimConnectionId:Identifier resourceProviderId:Identifier
VimComputeResourceFlavour	vimConnectionId:Identifier resourceProviderId:Identifier vnfdVirtualComputeDescId:Identifier vimFlavourId:Identifier
VimSoftwareImage	vimConnectionId:Identifier resourceProviderId:Identifier vnfdSoftwareImageId:Identifier vimSoftwareImageId:Identifier

A.6.4 VNF indicator

Table A.6.4-1 shows the input/output parameters for each operation of the VNF indicator interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.6.4-2, otherwise a reference is provided.

Table A.6.4-1: VNF indicator operations

Operation	Or-Vnfm	Ve-Vnfm	Input/output Parameter:Type
Subscribe	X	X	filter:Filter
			subscriptionId:Identifier
Get Indicator Value	X	X	filter:Filter
			indicatorInformation:**IndicatorInformation**
Terminate Subscription	X	X	subscriptionId:Identifier
			none
Query Subscription Info	X	X	filter:Filter
			queryResult:<not specified>
Notify	X		indicatorValueChangeNotification:**IndicatorValueChangeNotification** xor supportedIndicatorsChangeNotification:**SupportedIndicatorsChangeNotification**

Table A.6.4-2 shows the information elements used in the operations data related to VNF indicator, unless a reference is provided.

Table A.6.4-2: Information elements used in the VNF indicator operations data

Information element	Attribute:Type
IndicatorInformation	vnfInstanceId:Identifier indicatorId:Identifier indicatorValue:Value indicatorName:String
SupportedIndicatorsChangeNotification	supportedIndicator:**SupportedIndicatorInformation** vnfInstanceId:Identifier
IndicatorValueChangeNotification	indicatorInformation:IndicatorInformation
SupportedIndicatorInformation	indicatorId:Identifier indicatorName:String

A.6.5 VNF lifecycle management

Table A.6.5-1 shows the input/output parameters for each operation of the VNF lifecycle management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.6.5-2, otherwise a reference is provided.

Table A.6.5-1: VNF lifecycle management operations

Operation	Or-Vnfm	Ve-Vnfm	Input/output Parameter:Type
Create VNF Identifier	X	X	vnfdId:Identifier vnfInstanceName:String vnfInstanceDescription:String metadata:KeyValuePair
			vnfInstanceId:Identifier
Instantiate VNF	X	X	vnfInstanceId:Identifier flavourId:Identifier instantiationLevelId:Identifier extVirtualLink:**ExtVirtualLinkData** (see Table A.7.1-4) extManagedVirtualLink:**ExtManagedVirtualLinkData** (see Table A.7.1-4) vimConnectionInfo:**VimConnectionInfo** (see Table A.7.1-4) localizationLanguage:<not specified> additionalParam:KeyValuePair extension:KeyValuePair vnfConfigurableProperty:KeyValuePair
			lifecycleOperationOccurrenceId:Identifier
Scale VNF	X	X	vnfInstanceId:Identifier

Operation	Or-Vnfm	Ve-Vnfm	Input/output Parameter:Type
			type:*VnfScaleType* aspectId:Identifier numberOfSteps:Integer additionalParam:KeyValuePair
			lifecycleOperationOccurrenceId:Identifier
Scale VNF to Level	X	X	vnfInstanceId:Identifier instantiationLevelId:Identifier scaleInfo:**ScaleInfo** additionalParam:KeyValuePair
			lifecycleOperationOccurrenceId:Identifier
Change VNF Flavour	X	X	vnfInstanceId:Identifier newFlavourId:Identifier instantiationLevelId:Identifier extVirtualLink:**ExtVirtualLinkData** (see Table A.7.1-4) extManagedVirtualLink:**ExtManagedVirtualLinkData** (see Table A.7.1-4) vimConnectionInfo:**VimConnectionInfo** (see Table A.7.1-4) additionalParam:KeyValuePair extension:KeyValuePair vnfConfigurableProperty:KeyValuePair
			lifecycleOperationOccurrenceId:Identifier
Terminate VNF	X	X	vnfInstanceId:Identifier terminationType:*VnfStopType* gracefulTerminationTimeout:TimeDuration additionalParam:KeyValuePair
			lifecycleOperationOccurrenceId:Identifier
Delete VNF Identifier	X	X	vnfInstanceId:Identifier
			none
Query VNF	X	X	filter:Filter attributeSelector:String
			vnfInfo:**VnfInfo**
Heal VNF	X	X	vnfInstanceId:Identifier cause:String additionalParam:KeyValuePair
			lifecycleOperationOccurrenceId:Identifier
Operate VNF	X	X	vnfInstanceId:Identifier changeStateTo:*VnfState* stopType:*VnfStopType* gracefulStopTimeout:TimeDuration additionalParam:KeyValuePair
			lifecycleOperationOccurrenceId:Identifier
Modify VNF Information	X	X	vnfInstanceId:Identifier newValues:KeyValuePair vnfcConfigurationData:**VnfcConfigurationKvp**
			lifecycleOperationOccurrenceId:Identifier
Get Operation Status	X	X	lifecycleOperationOccurrenceId:Identifier
			operationStatus:*LmcOperationStatus*
Subscribe	X	X	filter:Filter
			subscriptionId:Identifier
Terminate Subscription	X	X	subscriptionId:Identifier
			none
Query Subscription Info	X	X	filter:Filter
			queryResult:<not specified>
Change External VNF Connectivity	X	X	vnfInstanceId:Identifier extVirtualLink:**ExtVirtualLinkData** (see Table A.7.1-4) additionalParam:KeyValuePair vimConnectionInfo:**VimConnectionInfo** (see Table A.7.1-4)
			none

Operation	Or-Vnfm	Ve-Vnfm	Input/output Parameter:Type
Query Snapshot Information	X	X	filter:Filter attributeSelector:String
			vnfSnapshotInfo:**VnfSnapshotInfo** (see Table A.6.5-2)
Create Snapshot	X	X	vnfInstanceId:Identifier additionalParam:KeyValuePair userDefinedData:KeyValuePair
			vnfSnapshotInfoId:Identifier
Revert-to Snapshot	X	X	vnfInstanceId:Identifier vnfSnapshotInfoId:Identifier additionalParam:KeyValuePair
			none
Delete Snapshot Information	X	X	vnfSnapshotInfoId:Identifier
			none
Change current VNF package	X	X	vnfInstanceId:Identifier vnfdId:Identifier extVirtualLink:**ExtVirtualLinkData** (see Table A.7.1-4) extManagedVirtualLink:**ExtManagedVirtualLinkData** (see Table A.7.1-4) vimConnectionInfo:**VimConnectionInfo** (see Table A.7.1-4) additionalParam:KeyValuePair extension:KeyValuePair vnfConfigurableProperties:KeyValuePair
			lifecycleOperationOccurrenceId:Identifier
Fetch VNF state snapshot	X		vnfSnapshotInfoId:Identifier
			vnfStateSnapshot:<not specified>
Notify	X	X	vnfLcmOperationOccurrenceNotification:**VnfLcmOperationOccurrenceNotification** xor vnfIdentifierCreationNotification:**VnfIdentifierCreationNotification** xor vnfIdentifierDeletionNotification:**VnfIdentifierDeletionNotification**

Table A.6.5-2 shows the information elements used in the operations data related to VNF lifecycle management, unless a reference is provided.

Table A.6.5-2: Information elements used in the VNF lifecycle management operations data

Information element	Attribute:Type
VnfInfo	vnfInstanceId:Identifier vnfInstanceName:String vnfInstanceDescription:String vnfdId:Identifier vnfProvider:String vnfProductName:String vnfSoftwareVersion:Version vnfdVersion:Version vnfConfigurableProperty:KeyValuePair vimConnectionInfo:**VimConnectionInfo** (see Table A.7.1-4) instantiationState:*InstantiationState* instantiatedVnfInfo:**InstantiatedVnfInfo** metadata:KeyValuePair extension:KeyValuePair
InstantiatedVnfInfo	flavourId:Identifier vnfState:*VnfState* scaleStatus:**ScaleInfo** extCpInfo:**VnfExtCpInfo** extVirtualLinkInfo:**ExtVirtualLinkInfo** extManagedVirtualLinkInfo:**ExtManagedVirtualLinkInfo** monitoringParameter:**MonitoringParameter** (see Table A.7.1-1)

Information element	Attribute:Type
	localizationLanguage:<not specified> vnfcResourceInfo:**VnfcResourceInfo** vnfVirtualLinkResourceInfo:**VnfVirtualLinkResourceInfo** virtualStorageResourceInfo:**VirtualStorageResourceInfo** vnfcInfo:**VnfcInfo** vimId:Identifier maxScaleLevel:**ScaleInfo**
ExtVirtualLinkInfo	extVirtualLinkId:Identifier resourceHandle:**ResourceHandle** (see Table A.7.1-4) extLinkPort:**ExtLinkPortInfo**
ExtLinkPortInfo	extLinkPortId:Identifier resourceHandle:**ResourceHandle** (see Table A.7.1-4) cpInstanceId:Identifier
ExtManagedVirtualLinkInfo	extManagedVirtualLinkId:Identifier vnfVirtualLinkDescId:Identifier networkResource:**ResourceHandle** (see Table A.7.1-4) vnfLinkPort:**VnfLinkPortInfo** extManagedMultisiteVirtualLinkId:Identifier vnfdId:Identifier
VnfExtCpInfo	cpInstanceId:Identifier cpdId:Identifier cpProtocolInfo:**CpProtocolInfo** (see Table A.7.1-4) associatedVnfcCpId:Identifier associatedVnfVirtualLinkId:Identifier extLinkPortId:Identifier metadata:KeyValuePair vnfdId:Identifier
VnfLinkPortInfo	vnfLinkPortId:Identifier resourceHandle:**ResourceHandle** (see Table A.7.1-4) associatedExtCpId:Identifier vnfcCpInstanceId:Identifier
VnfVirtualLinkResourceInfo	virtualLinkInstanceId:Identifier vnfVirtualLinkDescId:Identifier networkResource:**ResourceHandle** (see Table A.7.1-4) reservationId:Identifier vnfLinkPort:**VnfLinkPortInfo** metadata:KeyValuePair vnfdId:Identifier
VnfcCpInfo	cpInstanceId:Identifier cpdId:Identifier vnfExtCpId:Identifier vnfLinkPortId:Identifier cpProtocolInfo:**CpProtocolInfo** (see Table A.7.1-4) metadata:KeyValuePair
VnfcInfo	vnfcInstanceId:Identifier vduId:Identifier vnfcState:*VnfcState* vnfcConfigurableProperty:KeyValuePair vnfcResourceInfoId:Identifier
VnfcResourceInfo	vnfcInstanceId:Identifier vduId:Identifier computeResource:**ResourceHandle** (see Table A.7.1-4) storageResourceId:Identifier reservationId:Identifier vnfcCpInfo:**VnfcCpInfo** metadata:KeyValuePair vnfdId:Identifier
ScaleInfo	aspectId:Identifier scaleLevel:Integer vnfdId:Identifier
VnfSnapshotInfo	vnfSnapshotInfoId:Identifier createdAt:DateTime vnfInstanceId:Identifier vnfInfo:**VnfInfo** vnfcSnapshotInfo:**VnfcSnapshotInfo** userDefinedData:KeyValuePair triggeredAt:DateTime

Information element	Attribute:Type
	vnfStateSnapshotInfo:**VnfStateSnapshotInfo** vnfdId:Identifier
VnfcSnapshotInfo	vnfcSnapshotInfoId:Identifier createdAt:DateTime vnfcInstanceId:Identifier computeSnapshotResource:**ResourceHandle** (see Table A.7.1-4) storageSnapshotResource:**StorageSnapshotResource** userDefinedData:KeyValuePair triggeredAt:DateTime vnfcInfoId:Identifier
StorageSnapshotResource	storageSnapshotResource:**ResourceHandle** (see Table A.7.1-4) storageResourceId:Identifier
VnfcConfigurationKvp	vnfcId:Identifier vnfcConfigKvp:KeyValuePair
VnfLcmOperationOccurrenceNotification	status:*OperationStatus* vnfInstanceId:Identifier operation:String isAutomaticInvocation:Boolean lifecycleOperationOccurrenceId:Identifier affectedVnfc:**AffectedVnfc** affectedVirtualLink:**AffectedVirtualLink** affectedVirtualStorage:**AffectedVirtualStorage** changedExtConnectivity:**ExtVirtualLinkInfo** changedInfo:<not specified>
VnfIdentifierCreationNotification	vnfInstanceId:Identifier
VnfIdentifierDeletionNotification	vnfInstanceId:Identifier
VirtualStorageResourceInfo	virtualStorageInstanceId:Identifier virtualStorageDescId:Identifier storageResource:**ResourceHandle** (see Table A.7.1-4) reservationId:Identifier metadata:KeyValuePair vnfdId:Identifier
AffectedVirtualLink	virtualLinkInstanceId:Identifier vnfVirtualLinkDescId:Identifier changeType:*VirtualLinkChangeType* networkResource:**ResourceHandle** (see Table A.7.1-4) metadata:KeyValuePair vnfdId:Identifier
AffectedVirtualStorage	virtualStorageInstanceId:Identifier virtualStorageDescId:Identifier changeType:*VirtualStorageChangeType* storageResource:**ResourceHandle** (see Table A.7.1-4) metadata:KeyValuePair vnfdId:Identifier
AffectedVnfc	vnfcInstanceId:Identifier vduId:Identifier changeType:*VnfcChangeType* computeResource:**ResourceHandle** (see Table A.7.1-4) affectedVnfcCpInstances:**VnfcCpInfo** metadata:KeyValuePair addedStorageResourceIds:Identifier removedStorageResourceIds:Identifier vnfdId:Identifier
VnfStateSnapshotInfo	accessInformation:<not specified> metadata:<not specified>

A.6.6 VNF package management

Table A.6.6-1 shows the input/output parameters for each operation of the VNF package management interface, i.e. for each kind of notification and request-response. The information elements used in the operations data are shown in Table A.6.6-2, otherwise a reference is provided.

Table A.6.6-1: VNF package management operations

Operation	Or-Vnfm	Os-Ma-Nfvo	Input/output Parameter:Type
Upload VNF Package		X	vnfPkgInfoId:Identifier vnfPackage:Binary vnfPackagePath:URL
			none
Delete VNF Package		X	VnfPkgInfoId:Identifier
			none
Create VNF Package Info		X	userDefinedData:KeyValuePair
			vnfPkgInfoId:Identifier
Update VNF Package Info		X	vnfPkgInfoId:Identifier operationalState:*OperationalState* userDefinedData:KeyValuePair
			none
Query VNF Package Info	X	X	filter:Filter attributeSelector:String
			queryResult:**VnfPkgInfo**
Subscribe	X	X	filter:Filter
			subscriptionId:Identifier
Fetch VNF Package	X	X	vnfPkgInfoId:Identifier
			vnfPackage:Binary
Fetch VNF Package Artifacts	X	X	vnfPkgInfoId:Identifier artifactSelector:<not specified>
			vnfPackageArtifact:<not specified>
Terminate Subscription	X	X	subscriptionId:Identifier
			none
Query Subscription Info	X	X	filter:Filter
			queryResult:<not specified>
Notify	X	X	vnfPackageOnBoardingNotification:**VnfPackageOnBoardingNotification** xor vnfPackageChangeNotification:**VnfPackageChangeNotification**

Table A.6.6-2 shows the information elements used in the operations data related to VNF package management, unless a reference is provided.

Table A.6.6-2: Information elements used in the VNF package management operations data

Information element	Attribute:Type
VnfPkgInfo	vnfPkgInfoId:Identifier vnfdId:Identifier vnfProvider:String vnfProductName:String vnfSoftwareVersion:Version vnfdVersion:Version checksum:<not specified> vnfd:**Vnfd** (see Table A.7.4-2) softwareImage:**VnfPackageSoftwareImageInfo** additionalArtifact:**VnfPackageArtifactInformation** onboardingState:*OnboardingState* operationalState:*OperationalState* usageState:*UsageState* userDefinedData:KeyValuePair
VnfPackageChangeNotification	onboardedVnfPkgInfoId:Identifier vnfdId:Identifier changeType:*VnfPackageChangeType* operationalState:*OperationalState*
VnfPackageOnBoardingNotification	onboardedVnfPkgInfoId:Identifier vnfdId:Identifier
VnfPackageSoftwareImageInfo	name:<not specified> provider:<not specified> version:<not specified> checksum:<not specified> containerFormat:<not specified> diskFormat:<not specified> createdAt:<not specified> minDisk:<not specified> minRam:<not specified> size:<not specified> userMetadata:KeyValuePair accessInformation:<not specified>
VnfPackageArtifactInformation	selector:<not specified> metadata:<not specified>

A.6.7 VNF snapshot package management

Table A.6.7-1 shows the input/output parameters for each operation of the VNF snapshot package management interface, i.e. for each kind of request-response. The information elements used in the operations data are shown in Table A.6.7-2, otherwise a reference is provided.

Table A.6.7-1: VNF snapshot package management operations

Operation	Or-Vnfm	Os-Ma-Nfvo	Input/output Parameter:Type
Fetch VNF Snapshot Package	X	X	vnfSnapshotPkgInfoId:Identifier
			vnfSnapshotPackage:Binary
Fetch VNF Snapshot Package Artifacts	X	X	vnfSnapshotPkgInfoId:Identifier artifactSelector:<not specified>
			vnfSnapshotPackageArtifact:<not specified>
Query VNF Snapshot Package Information	X	X	filter:Filter attributeSelector:String
			queryResult:**VnfSnapshotPkgInfo**
Create VNF Snapshot Package Info		X	name:String userDefinedData:KeyValuePair
			vnfSnapshotPkgInfoId:Identifier
Build VNF Snapshot Package		X	vnfSnapshotPkgInfoId:Identifier vnfSnapshotInfoId:Identifier
			none
Upload VNF Snapshot Package		X	vnfSnapshotPkgInfoId:Identifier vnfSnapshotPkg:Binary vnfSnapshotPkgPath:<not specified>
			none
Extract VNF Snapshot Package		X	vnfSnapshotPkgInfoId:Identifier vnfSnapshotInfoId:Identifier vnfInstanceId:Identifier
			vnfSnapshotInfoId:Identifier
Delete VNF Snapshot Package		X	vnfSnapshotPkgInfoId:Identifier
			none
Update VNF Snapshot Package		X	vnfSnapshotPkgInfoId:Identifier name:String userDefinedData:KeyValuePair
			none

Table A.6.7-2 shows the information elements used in the operations data related to VNF snapshot package management, unless a reference is provided.

Table A.6.7-2: Information elements used in the VNF snapshot package management operations data

Information element	Attribute:Type
VnfSnapshotPkgInfo	vnfSnapshotPkgInfoId:Identifier vnfSnapshotPkgId:Identifier name:String checksum:<not specified> createdAt:DateTime vnfSnapshotInfoId:Identifier vnfd:**Vnfd** (see Table A.7.4-2) vnfInfo:**VnfInfo** (see Table A.6.5-2) vnfcSnapshotInfoId:Identifier vnfcSnapshotImage:**VnfcSnapshotImageInfo** additionalArtifact:**SnapshotPkgArtifactInformation** state:*SnapshotPkgState* userDefinedData:KeyValuePair accessInformation:<not specified> isFullSnapshot:Boolean
SnapshotPkgArtifactInformation	selector:<not specified> metadata:<not specified>
VnfcSnapshotImageInfo	vnfcSnapshotImageId:Identifier name:<not specified> checksum:<not specified> vnfcInstanceId:Identifier containerFormat:<not specified> diskFormat:<not specified> createdAt:DateTime minDisk:<not specified> minRam:<not specified> size:<not specified> userMetadata:KeyValuePair accessInformation:<not specified>

A.6.8 LCM coordination

Table A.6.8-1 shows the input/output parameters for the operation of the LCM coordination interface.

Table A.6.8-1: LCM coordination operation

Operation	Ve-Vnfm	Input/output Parameter:Type
CoordinateLcmOperation	X	vnfInstanceId:Identifier lifecycleOperationOccurrenceId:Identifier operationType:<not specified> operationStage:<not specified> operationParam:<not specified>
		operationAction:*OperationAction* operationResumeDelay:TimeDuration additionalInfo:<not specified>

A.7 Information elements of the NFV core model used in the interfaces and operations

A.7.1 NFV common domain

Table A.7.1-1 shows the information elements related to common template, together with their attribute:type.

Table A.7.1-1: Common template information elements

Information element	Attribute:Type
AffinityOrAntiAffinityGroup	groupId:Identifier affinityOrAntiAffinity:*AffinityOrAntiAffinity* scope:*AffinityOrAntiAffinityScope*
Cpd	cpdId:Identifier layerProtocol:*LayerProtocol* cpRole:String description:String trunkMode:Boolean securityGroupRuleId:IdentifiercpProtocol:**CpProtocol Data**
ExtCpd	cpdId:Identifier layerProtocol:*LayerProtocol* cpRole:String description:String trunkMode:Boolean securityGroupRuleId:Identifier cpProtocol:**CpProtocolData**
VirtualLinkDesc	virtualLinkDescId:Identifier connectivityType:**ConnectivityType** testAccess:String description:String
AddressData	addressType:*AddressType* l2AddressData:**L2AddressData** l3AddressData:**L3AddressData**
ChecksumData	algorithm:String hash:String
ConnectivityType	layerProtocol:*LayerProtocol* flowPattern:String
CpProtocolData	associatedLayerProtocol:*LayerProtocol* addressData:**AddressData**
L2AddressData	macAddressAssignment:Boolean
L3AddressData	iPAddressAssignment:Boolean floatingIpActivated:Boolean iPAddressType:*IpAddressType* numberOfIpAddress:Integer fixedIpAddress:String
LinkBitrateRequirements	root:Number leaf:Number
LocalAffinityOrAntiAffinityRule	type:*AffinityOrAntiAffinity* scope:*AffinityOrAntiAffinityScope* nfviMaintenanceGroupInfo:**NfviMaintenanceInfo** (see Table A.7.4-2)
MonitoringParameter	monitoringParameterId:Identifier name:String performanceMetric:String collectionPeriod:<not specified>
QoS	latency:Number packetDelayVariation:Number packetLossRatio:Number
ScaleInfo	aspectId:Identifier scaleLevel:Integer
SecurityGroupRule	securityGroupRuleId:Identifier description:String direction:*Direction* etherType:*EtherType* protocol:*Protocol* portRangeMin:Integer portRangeMax:Integer
SecurityParameters	signature:String algorithm:String certificate:<not specified>

Table A.7.1-2 shows the information elements related to common topology, together with their attribute:type.

Table A.7.1-2: Common topology information elements

Information element	Attribute:Type
Cp	cpId:Identifier
VirtualLink	linkPort:***LinkPort***

Table A.7.1-3 shows the information elements related to common types, together with their attribute:type.

Table A.7.1-3: Common types information elements

Information element	Attribute:Type
KeyValuePair	key:String value:Value
TimePeriod	startTime:DateTime stopTime:DateTime
TimePeriodInformation	startTime:DateTime stopTime:DateTime

Table A.7.1-4 shows the information elements related to common elements, together with their attribute:type.

Table A.7.1-4: Common elements information elements

Information element	Attribute:Type
AffinityOrAntiAffinityResourceList	resourceId:Identifier
CpProtocolInfo	layerProtocol:*LayerProtocol* address:<not specified>
VimConnectionInfo	vimConnectionInfoId:Identifier vimId:Identifier interfaceInfo:<not specified> accessInfo:<not specified> extra:<not specified>
ExtLinkPortData	extLinkPortId:Identifier resourceHandle:**ResourceHandle**
ExtManagedVirtualLinkData	extManagedVirtualLinkId:Identifier vnfVirtualLinkDescId:Identifier vimConnectionId:Identifier resourceProviderId:Identifier resourceId:Identifier vnfLinkPort:**VnfLinkPortData** extManagedMultisiteVirtualLinkId:Identifier vimId:Identifier
ExtVirtualLinkData	extVirtualLinkId:Identifier vimConnectionId:Identifier resourceProviderId:Identifier resourceId:Identifier extCp:**VnfExtCpData** extLinkPorts:**ExtLinkPortData** vimId:Identifier
ResourceHandle	vimConnectionId:Identifier resourceProviderId:Identifier resourceId:Identifier vimLevelResourceType:<not specified> vimId:Identifier
VnfExtCpConfig	cpInstanceId:Identifier linkPortId:Identifier cpProtocolData:<not specified>
VnfExtCpData	cpdId:Identifier cpConfig:**VnfExtCpConfig**
VnfLinkPortData	vnfLinkPortId:Identifier resourceHandle:**ResourceHandle**

A.7.2 NS domain

Table A.7.2-1 shows the information elements related to NS, together with their attribute:type.

Table A.7.2-1: NS information elements

Information element	Attribute:Type
NetworkService	nsInstanceId:Identifier nsName:String description:String nsd:**Nsd** nf:***NetworkFunction*** nsVirtuaLlink:**NsVirtualLink** vnffg:**Vnffg** sap:**Sap** nestedNs:**NetworkService**
NsVirtualLink	nsVirtualLinkDesc:**NsVirtualLinkDesc** (see Table A.7.2-2) virtualNetwork:**VirtualNetwork** (see Table A.7.3-2) linkPort:***LinkPort***
Pnf	pnfd:**Pnfd** (see Table A.7.2-2) pnfExternalCp:**PnfExtCp**
PnfExtCp	cpd:**Cpd** (see Table A.7.1-1) cpId:Identifier
Sap	sapd:**Sapd** cpId:Identifier
Vnffg	vnffgId:Identifier vnffgd:**Vnffgd** (see Table A.7.2-2) nfId:Identifier virtualLinkId:Identifier cpId:Identifier nfp:**Nfp** (see Table A.7.3-1)

Table A.7.2-2 shows the information elements related to NS template, together with their attribute:type.

Table A.7.2-2: NS template information elements

Information element	Attribute:Type
CpProfile	cpProfileId:Identifier constituentProfileElements:<not specified>
Dependencies	primaryId:Identifier secondaryId:Identifier
NfpPositionDesc	nfpPositionDescId:Identifier forwardingBehaviour:*ForwardingBehaviourType* forwardingBehaviourInputParameters:<not specified> nfpPositionElementId:Identifier
NfpPositionElement	nfpPositionElementId:Identifier nfpPositionElementDesc:**CpdInConstituentElement**
Nfpd	nfpdId:Identifier nfpRule:**NfpRule** (see Table A.4.2-2) nfpPositionDesc:**NfpPositionDesc**
NsDf	nsDfId:Identifier flavourKey:String vnfProfile:**VnfProfile** pnfProfile:**PnfProfile** virtualLinkProfile:**VirtualLinkProfile** scalingAspect:**NsScalingAspect** affinityOrAntiAffinityGroup:**AffinityOrAntiAffinityGroup** (see Table A.7.1-1) nsInstantiationLevel:**NsLevel** defaultNsInstantiationLevelId:Identifier nsProfile:**NsProfile** dependencies:**Dependencies** monitoredInfo:**MonitoredData** priority:Integer serviceAvailabilityLevel:*ServiceAvailabilityLevel* nsLcmAdditionalParams:**NsLcmAdditionalParams**

Information element	Attribute:Type
NsLevel	nsLevelId:Identifier description:String vnfToLevelMapping:**VnfToLevelMapping** virtualLinkToLevelMapping:**VirtualLinkToLevelMapping** nsToLevelMapping:**NsToLevelMapping**
NsProfile	nsProfileId:Identifier nsdId:Identifier nsDfId:Identifier nsInstantiationLevelId:Identifier minNumberOfInstances:Integer maxNumberOfInstances:Integer affinityOrAntiAffinityGroupId:Identifier nsVirtualLinkConnectivity:**NsVirtualLinkConnectivity**
NsScalingAspect	nsScalingAspectId:Identifier name:String description:String nsScaleLevel:**NsLevel**
NsVirtualLinkDesc	virtualLinkDf:**VirtualLinkDf** virtualLinkDescId:Identifier connectivityType:**ConnectivityType** (see Table A.7.1-1) testAccess:String description:String
Nsd	nsdIdentifier:Identifier designer:String version:Version nsdName:String nsdInvariantId:Identifier nestedNsdId:Identifier vnfdId:Identifier pnfdId:Identifier sapd:**Sapd** virtualLinkDesc:**NsVirtualLinkDesc** vnffgd:Vnffgd autoScalingRule:<not specified> lifeCycleManagementScript:**LifeCycleManagementScript** nsDf:**NsDf** security:**SecurityParameters** (see Table A.7.1-1)
PnfExtCpd	cpdId:Identifier layerProtocol:*LayerProtocol* cpRole:String description:String trunkMode:Boolean securityGroupRuleId:Identifier cpProtocol:**CpProtocolData** (see Table A.7.1-1)
PnfProfile	pnfProfileId:Identifier pnfdId:Identifier pnfVirtualLinkConnectivity:**NsVirtualLinkConnectivity**
Pnfd	pnfdId:Identifier functionDescription:String provider:String version:Version pnfdInvariantId:Identifier name:String pnfExtCp:**PnfExtCpd** geographicalLocationInfo:<not specified>
Sapd	nsVirtualLinkDescId:Identifier associatedCpd:**CpdInConstituentElement** cpdId:Identifier layerProtocol:*LayerProtocol* cpRole:String description:String trunkMode:Boolean securityGroupRuleId:Identifier cpProtocol:**CpProtocolData** (see Table A.7.1-1)

Information element	Attribute:Type
VirtualLinkProfile	virtualLinkProfileId:Identifier virtualLinkDescId:Identifier flavourId:Identifier localAffinityOrAntiAffinityRule:**LocalAffinityOrAntiAffinityRule** (see Table A.7.1-1) affinityOrAntiAffinityGroupId:Identifier maxBitrateRequirements:**LinkBitrateRequirements** (see Table A.7.1-1) minBitrateRequirements:**LinkBitrateRequirements** (see Table A.7.1-1) virtualLinkProtocolData:**VirtualLinkProtocolData** (see Table A.7.4-2)
VnfProfile	vnfProfileId:Identifier vnfdId:Identifier flavourId:Identifier instantiationLevel:Identifier minNumberOfInstances:Integer maxNumberOfInstances:Integer localAffinityOrAntiAffinityRule:**LocalAffinityOrAntiAffinityRule** (see Table A.7.1-1) nsVirtualLinkConnectivity:**NsVirtualLinkConnectivity** serviceAvailabilityLevel:*ServiceAvailabilityLevel* affinityOrAntiAffinityGroupId:Identifier
Vnffgd	vnffgdId:Identifier vnfProfileId:Identifier pnfProfileId:Identifier virtualLinkProfileId:Identifier nfpd:**Nfpd** nfpPositionElement:**NfpPositionElement** nestedNsProfileId:Identifier
CpPoolManagement	cpdId:Identifier forwardingBehaviour:*ForwardingBehaviourType* forwardingBehaviourInputParameters:<not specified>
CpdInConstituentElement	constituentBaseElementId:Identifier constituentCpdId:Identifier
LifeCycleManagementScript	event:String script:<not specified>
MonitoredData	vnfIndicatorInfo:**VnfIndicatorData** monitoringParameter:**MonitoringParameter** (see Table A.7.1-1)
NsQoS	priority:Integer latency:Number packetDelayVariation:Number packetLossRatio:Number
NsToLevelMapping	nsProfileId:Identifier numberOfInstances:Integer
NsVirtualLinkConnectivity	virtualLinkProfileId:Identifier constituentCpdId:Identifier
VirtualLinkDf	flavourId:Identifier qos:**NsQoS** serviceAvailabilityLevel:*ServiceAvailabilityLevel*
VirtualLinkToLevelMapping	virtualLinkProfileId:Identifier bitrateRequirements:**LinkBitrateRequirements** (see Table A.7.1-1)
VnfIndicatorData	vnfdId:Identifier vnfIndicator:**VnfIndicator** (see Table A.7.4-2)
VnfToLevelMapping	vnfProfileId:Identifier numberOfInstances:Integer
NsLcmAdditionalParams	instantiateNsAdditionalParams:**InstantiateNsAdditionalParams** scaleNsAdditionalParams:**ScaleNsAdditionalParams** healNSAdditionalParams:**HealNsAdditionalParams**
InstantiateNsAdditionalParams	nsAdditionalParam:<not specified>
ScaleNsAdditionalParams	nsAdditionalParam:<not specified>
HealNsAdditionalParams	nsAdditionalParam:<not specified>

A.7.3 Resource domain

Table A.7.3-1 shows the information elements related to network forwarding path, together with their attribute:type.

Table A.7.3-1: Network forwarding path information elements

Information element	Attribute:Type
Nfp	nfpId:Identifier virtualNetworkPortGroup:**VirtualNetworkPortGroup** totalVnp:Integer nfpRule:**NfpRule** (see Table A.4.2-2) nfpState:*OperationalState*
VirtualNetworkPortGroup	virtualNetworkPortPair:**VirtualNetworkPortPair** forwardingBehaviour:*ForwardingBehaviourRule* forwardingBehaviourInputParameters:<not specified>
VirtualNetworkPortPair	ingressVnp:**VirtualNetworkPort** egressVnp:**VirtualNetworkPort**

Table A.7.3-2 shows the information elements related to virtualised resource, together with their attribute:type.

Table A.7.3-2: Virtualised resource information elements

Information element	Attribute:Type
ConnectivityServiceEndpoint	connectivityServiceEndpointId:Identifier associatedResourceId:Identifier connectivityServiceEndpoint:<not specified>
NetworkQoS	qosName:String qosValue:Value
NetworkSubnet	resourceId:Identifier networkId:Identifier ipVersion:*IpVersion* gatewayIp:IpAddress cidr:<not specified> isDhcpEnabled:Boolean addressPool:<not specified> metadata:KeyValuePair
NfviPop	nfviPopId:Identifier vimId:Identifier geographicalLocationInfo:Location networkConnectivityEndpoint:**ConnectivityServiceEndpoint**
ResourceZone	zoneId:Identifier zoneName:String zoneState:String nfviPopId:Identifier zoneProperty:<not specified> metadata:KeyValuePair
VirtualCompute	computeId:Identifier computeName:String flavourId:Identifier accelerationCapability:<not specified> virtualCpu:**VirtualCpu** virtualMemory:**VirtualMemory** virtualNetworkInterface:**VirtualNetworkInterface** virtualDisks:**VirtualStorage** vcImageId:Identifier zoneId:Identifier hostId:Identifier operationalState:*OperationalState* metadata:KeyValuePair
VirtualCpu	cpuArchitecture:String numVirtualCpu:Integer cpuClock:Number virtualCpuOversubscriptionPolicy:<not specified> virtualCpuPinning:**VirtualCpuPinning**
VirtualMachine	virtualCompute:**VirtualCompute**
VirtualMemory	virtualMemSize:Number virtualMemOversubscriptionPolicy:<not specified> numaEnabled:Boolean

Information element	Attribute:Type
VirtualNetwork	networkResourceId:Identifier networkResourceName:String subnetId:Identifier networkPort:**VirtualNetworkPort** bandwidth:Number networkType:String segmentType:String networkQoS:**NetworkQoS** isShared:Boolean sharingCriteria:<not specified> zoneId:Identifier operationalState:*OperationalState* metadata:KeyValuePair
VirtualNetworkInterface	resourceId:Identifier ownerId:Identifier networkId:Identifier networkPortId:Identifier ipAddress:IpAddress typeVirtualNic:<not specified> typeConfiguration:<not specified> macAddress:MacAddress bandwidth:Number accelerationCapability:<not specified> operationalState:*OperationalState* metadata:KeyValuePair
VirtualNetworkPort	resourceId:Identifier networkId:Identifier portType:String segmentId:Identifier bandwidth:Number attachedResourceId:Identifier operationalState:*OperationalState* metadata:KeyValuePair
VirtualStorage	storageId:Identifier storageName:String flavourId:Identifier typeOfStorage:String sizeOfStorage:Number rdmaEnabled:Boolean ownerId:Identifier zoneId:Identifier hostId:Identifier operationalState:*OperationalState* metadata:KeyValuePair
VirtualCpuPinning	cpuPinningPolicy:*CpuPinningPolicy* cpuPinningRules:<not specified> cpuMap:<not specified>

A.7.4 VNF domain

Table A.7.4-1 shows the information elements related to VNF, together with their attribute:type.

Table A.7.4-1: VNF information elements

Information element	Attribute:Type
Vnf	vnfInstanceId:Identifier vnfd:**Vnfd** vnfc:**Vnfc** vnfExtCp:**VnfExtCp** vnfVirtualLink:**VnfVirtualLink** virtualStorage:**VirtualStorage** (see Table A.7.3-2)
VnfExtCp	vnfExtCpd:**VnfExtCpd** cp:**VnfcCp** linkPort:***LinkPort*** cpId:Identifier

Information element	Attribute:Type
VnfVirtualLink	vnfVirtualLinkDesc:**VnfVirtualLinkDesc** virtualNetwork:**VirtualNetwork** (see Table A.7.3-2) linkPort:***LinkPort***
Vnfc	vnfcInstanceId:Identifier vdu:**Vdu** cp:**VnfcCp** virtualCompute:**VirtualCompute** (see Table A.7.3-2) virtualStorage:**VirtualStorage** (see Table A.7.3-2) virtualisationContainer:***VirtualisationContainer***
VnfcCp	vduCpd:**VduCpd** cpId:Identifier

Table A.7.4-2 shows the information elements related to VNF template, together with their attribute:type.

Table A.7.4-2: VNF template information elements

Information element	Attribute:Type
ComponentMapping	componentType:<not specified> sourceDescId:Identifier dstDescId:Identifier description:String
InstantiationLevel	levelId:Identifier description:String vduLevel:**VduLevel** virtualLinkBitRateLevel:**VirtualLinkBitRateLevel** scaleInfo:**ScaleInfo** (see Table A.6.5-2)
LogicalNodeRequirements	id:Identifier logicalNodeRequirementDetail:<not specified>
MaxNumberOfImpactedInstances	groupSize:Integer maxNumberOfImpactedInstances:Integer
NfviMaintenanceInfo	impactNotificationLeadTime:Number isImpactMitigationRequested:Boolean supportedMigrationType:MigrationType maxUndetectableInterruptionTime:Number minRecoveryTimeBetweenImpacts:Number maxNumberOfImpactedInstances:**MaxNumberOfImpactedInstances**
ScalingAspect	id:Identifier name:String description:String maxScaleLevel:PositiveInteger aspectDeltaDetails:**AspectDeltaDetails**
Subport	subportCpd:**VduCpd** segmentationId:Identifier
SwImageDesc	id:Identifier name:String version:Version checksum:**ChecksumData** (see Table A.7.1-1) containerFormat:String diskFormat:String minDisk:Number minRam:Number size:Number swImage:SwImage operatingSystem:String supportedVirtualisationEnvironment:String
TrunkPortTopology	parentPortCpd:**VduCpd** subportList:**Subport** segmentationType:*SegmentationType*
Vdu	vduId:Identifier name:String description:String intCpd:**VduCpd** virtualComputeDesc:**VirtualComputeDesc** virtualStorageDesc:**VirtualStorageDesc** bootOrder:KeyValuePair

Information element	Attribute:Type
	swImageDesc:**SwImageDesc** nfviConstraint:String monitoringParameter:**MonitoringParameter** (see Table A.7.1-1) configurableProperties:**VnfcConfigurableProperties** bootData:<not specified> trunkPort:**TrunkPortTopology**
VduCpd	intVirtualLinkDesc:**VnfVirtualLinkDesc** bitrateRequirement:Number virtualNetworkInterfaceRequirements:**VirtualNetworkInterfaceRequirements** order:Integer vnicType:*VnicType* cpdId:Identifier layerProtocol:*LayerProtocol* cpRole:String description:String trunkMode:Boolean securityGroupRuleId:Identifier cpProtocol:**CpProtocolData** (see Table A.7.1-1)
VduLevel	vduId:Identifier numberOfInstances:Integer
VduProfile	vduId:Identifier minNumberOfInstances:Integer maxNumberOfInstances:Integer localAffinityOrAntiAffinityRule:**LocalAffinityOrAntiAffinityRule** (see Table A.7.1-1) nfviMaintenanceInfo:**NfviMaintenanceInfo** affinityOrAntiAffinityGroupId:Identifier
VersionSelector	srcVnfdId:Identifier dstVnfdId:Identifier srcFlavourId:Identifier
VipCpd	vnfExtCpd:**VnfExtCpd** intCpd:**VduCpd** vipFunction:*VipFunction* cpdId:Identifier layerProtocol:*LayerProtocol* cpRole:String description:String trunkMode:Boolean securityGroupRuleId:Identifier cpProtocol:**CpProtocolData** (see Table A.7.1-1)
VirtualComputeDesc	virtualComputeDescId:Identifier logicalNode:**LogicalNodeRequirements** requestAdditionalCapabilities:**RequestedAdditionalCapabilityData** computeRequirements:<not specified> virtualMemory:**VirtualMemoryData** virtualCpu:**VirtualCpuData** virtualDisk:**BlockStorageData**
VirtualLinkDescFlavour	flavourId:Identifier qos:**QoS** (see Table A.7.1-1)
VirtualLinkProfile	vnfVirtualLinkDescId:Identifier flavourId:Identifier localAffinityOrAntiAffinityRule:**LocalAffinityOrAntiAffinityRule** (see Table A.7.1-1) maxBitRateRequirements:**LinkBitrateRequirements** (see Table A.7.1-1) minBitRateRequirements:**LinkBitrateRequirements** (see Table A.7.1-1) virtualLinkProtocolData:**VirtualLinkProtocolData** affinityOrAntiAffinityGroupId:Identifier
VirtualNetworkInterfaceRequirements	name:String description:String supportMandatory:Boolean networkInterfaceRequirements:<not specified> nicIoRequirements:**LogicalNodeRequirements**

Information element	Attribute:Type
VirtualStorageDesc	id:Identifier typeOfStorage:*StorageType* blockStorageData:**BlockStorageData** objectStorageData:**ObjectStorageData** fileStorageData:**FileStorageData** nfviMaintenanceInfo:**NfviMaintenanceInfo**
VnfDf	flavourId:Identifier description:String vduProfile:**VduProfile** virtualLinkProfile:**VirtualLinkProfile** instantiationLevel:**InstantiationLevel** supportedOperation:*SupportedOperations* vnfLcmOperationsConfiguration:**VnfLcmOperationsConfiguration** affinityOrAntiAffinityGroup:**AffinityOrAntiAffinityGroup** (see Table A.7.1-1) vnfIndicator:**VnfIndicator** monitoringParameter:**MonitoringParameter** (see Table A.7.1-1) scalingAspect:**ScalingAspect** initialDelta:**ScalingDelta** supportedVnfInterface:**VnfInterfaceDetails** defaultInstantiationLevelId:Identifier
VnfExtCpd	intVirtualLinkDesc:**VnfVirtualLinkDesc** intCpd:**VduCpd** virtualNetworkInterfaceRequirements:**VirtualNetworkInterfaceRequirements** vipCpd:**VipCpd** cpdId:Identifier layerProtocol:*LayerProtocol* cpRole:String description:String trunkMode:Boolean securityGroupRuleId:Identifier cpProtocol:**CpProtocolData** (see Table A.7.1-1)
VnfIndicator	id:Identifier name:String indicatorValue:String source:*VnfIndicatorSource*
VnfInterfaceDetails	interfaceName:*InterfaceNames* cpdId:Identifier interfaceDetails:<not specified>
VnfLcmOperationCoordination	vnfLcmOpCoordinationId:Identifier description:String endpointType:*EndpointType* coordinationStage:<not specified> coordinationParams:<not specified>
VnfPackageChangeInfo	selector:**VersionSelector** additionalParamsId:Identifier modificationQualifier:*ModificationQualifier* additionalModificationDescription:String componentMapping:**ComponentMapping** lcmScriptId:Identifier coordinationId:Identifier dstFlavourId:Identifier
VnfVirtualLinkDesc	monitoringParameter:**MonitoringParameter** (see Table A.7.1-1) virtualLinkDescFlavour:**VirtualLinkDescFlavour** nfviMaintenanceInfo:**NfviMaintenanceInfo** virtualLinkDescId:Identifier connectivityType:**ConnectivityType** (see Table A.7.1-1) testAccess:String description:String

Information element	Attribute:Type
Vnfd	vnfdId:Identifier vnfProvider:String vnfProductName:String vnfSoftwareVersion:Version vnfdVersion:Version vnfProductInfoName:String vnfProductInfoDescription:String vnfmInfo:String localizationLanguage:<not specified> defaultLocalizationLanguage:<not specified> vdu:**Vdu** virtualComputeDesc:**VirtualComputeDesc** virtualStorageDesc:**VirtualStorageDesc** swImageDesc:**SwImageDesc** intVirtualLinkDesc:**VnfVirtualLinkDesc** vnfExtCpd:**VnfExtCpd** deploymentFlavour:**VnfDf** configurableProperties:**VnfConfigurableProperties** modifiableAttributes:**VnfInfoModifiableAttributes** lifeCycleManagementScript:**LifeCycleManagementScript** vnfIndicator:**VnfIndicator** autoScale:<not specified> securityGroupRule:**SecurityGroupRule** (see Table A.7.1-1) vipCpd:**VipCpd** lcmOperationCoordination:**VnfLcmOperationCoordination** vnfPackageChangeInfo:**VnfPackageChangeInfo**
AspectDeltaDetails	deltas:**ScalingDelta** stepDeltas:**ScalingDelta**
BlockStorageData	sizeOfStorage:Number vduStorageRequirements:<not specified> rdmaEnabled:Boolean swImageDesc:**SwImageDesc**
ChangeCurrentVnfPackageOpConfig	parameter:<not specified> opConfigId:Identifier
ChangeExtVnfConnectivityOpConfig	parameter:<not specified>
ChangeVnfFlavourOpConfig	parameter:<not specified>
CreateSnapshotVnfOpConfig	parameter:<not specified>
FileStorageData	sizeOfStorage:Number fileSystemProtocol:String intVirtualLinkDesc:**VnfVirtualLinkDesc**
HealVnfOpConfig	parameter:<not specified> cause:String
InstantiateVnfOpConfig	parameter:<not specified>
L2ProtocolData	name:String networkType:*L2NetworkType* vlanTransparent:Boolean mtu:Integer segmentationId:Identifier
L3ProtocolData	name:String ipVersion:*IpVersion* cidr:<not specified> ipAllocationPools:<not specified> gatewayIp:IpAddress dhcpEnabled:Boolean ipv6AddressMode:*Ipv6AddressMode*
LifeCycleManagementScript	lcmScriptId:Identifier event:*LcmScriptEventType* lcmTransitionEvent:String script:<not specified> scriptDsl:String scriptInput:<not specified>
ObjectStorageData	maxSizeOfStorage:Number
OperateVnfOpConfig	minGracefulStopTimeout:Number maxRecommendedGracefulStopTimeout:Number parameter:<not specified>

Information element	Attribute:Type
RequestedAdditionalCapabilityData	requestedAdditionalCapabilityName:String supportMandatory:Boolean minRequestedAdditionalCapabilityVersion:Version preferredRequestedAdditionalCapabilityVersion:Version targetPerformanceParameters:KeyValuePair
RevertToSnapshotVnfOpConfig	parameter:<not specified>
ScaleVnfOpConfig	parameter:<not specified> scalingByMoreThanOneStepSupported:Boolean
ScaleVnfToLevelOpConfig	parameter:<not specified> arbitraryTargetLevelsSupported:Boolean
ScalingDelta	vduDelta:**VduLevel** virtualLinkBitRateDelta:**VirtualLinkBitRateLevel** scalingDeltaId:Identifier
TerminateVnfOpConfig	minGracefulTerminationTimeout:Number maxRecommendedGracefulTerminationTimeout:Number parameter:<not specified>
VirtualCpuData	cpuArchitecture:String numVirtualCpu:Integer virtualCpuClock:Number virtualCpuOversubscriptionPolicy:<not specified> vduCpuRequirements:<not specified> virtualCpuPinning:**VirtualCpuPinningData** (see Table A.5.2.5-2)
VirtualLinkBitRateLevel	vnfVirtualLinkDescId:Identifier bitrateRequirements:**LinkBitrateRequirements** (see Table A.7.1-1)
VirtualLinkProtocolData	associatedLayerProtocol:*LayerProtocol* l2ProtocolData:**L2ProtocolData** l3ProtocolData:**L3ProtocolData**
VirtualMemoryData	virtualMemSize:Number virtualMemOversubscriptionPolicy:<not specified> vduMemRequirements:<not specified> numaEnabled:Boolean
VnfConfigurableProperties	isAutoscaleEnabled:Boolean isAutohealEnabled:Boolean additionalConfigurableProperty:<not specified> vnfmInterfaceInfo:<not specified> vnfmOauthServerInfo:<not specified> vnfOauthServerInfo:<not specified>
VnfInfoModifiableAttributes	extension:<not specified> metadata:<not specified>
VnfLcmOperationsConfiguration	instantiateVnfOpConfig:**InstantiateVnfOpConfig** scaleVnfOpConfig:**ScaleVnfOpConfig** scaleVnfToLevelOpConfig:**ScaleVnfToLevelOpConfig** changeVnfFlavourOpConfig:**ChangeVnfFlavourOpConfig** healVnfOpConfig:**HealVnfOpConfig** terminateVnfOpConfig:**TerminateVnfOpConfig** operateVnfOpConfig:**OperateVnfOpConfig** changeExtVnfConnectivityOpConfig:**ChangeExtVnfConnectivityOpConfig** createSnapshotVnfOpConfig:**CreateSnapshotVnfOpConfig** revertToSnapshotVnfOpConfig:**RevertToSnapshotVnfOpConfig** changeCurrentVnfPackageOpConfig:**ChangeCurrentVnfPackageOpConfig**
VnfcConfigurableProperties	additionalVnfcConfigurableProperty:<not specified>

A.8 Definition of enumerations

Table A.8-1 shows the Enum values used for all interfaces. If applicable, the column "Expandable" indicates that additional values are possible for the related Enum.

Table A.8-1: Enum values for all interfaces

Enum	Values	Expandable
AckState	ACKNOWLEDGED, UNACKNOWLEDGED	
ActivationStatus	ACTIVATED, DEACTIVATED	
AddressType	MAC address, IP address	X
AffinityOrAntiAffinity	AFFINITY, ANTI_AFFINITY	
AffinityOrAntiAffinityConstraintType	AFFINITY_CONSTRAINT, ANTI_AFFINITY_CONSTRAINT	
AffinityOrAntiAffinityScope	NFVI_NODE, NFVI-PoP, Zone, ZoneGroup, NFVI-node, network-link-and-node, container-namespace, NIC, VIRTUAL_SWITCH_OR_ROUTER, PHYSICAL_NIC, PHYSICAL_NETWORK	X
AlarmState	FIRED, UPDATED, CLEARED	
ChangeResultType	ADD, REMOVE, INSTANTIATE, TERMINATE, SCALE, HEAL, UPDATE	
ConnectivityMode	P2P, MP	
ConstraintResourceReferenceType	RES_MGMT, GRANT	
CpuPinningPolicy	STATIC, DYNAMIC	
Direction	INGRESS, EGRESS	
Directionality	INBOUND, OUTBOUND, BOTH	
EndpointType	RECEIPT_OF_REQUEST_MESSAGE_OF_INSTANTIATION, RECEIPT_OF_REQUEST_MESSAGE_OF_SCALING, RECEIPT_OF_REQUEST_MESSAGE_OF_HEALING, RECEIPT_OF_REQUEST_MESSAGE_OF_TERMINATION, RECEIPT_OF_REQUEST_MESSAGE_OF_CHANGE _VNF_FLAVOUR, RECEIPT_OF_REQUEST_MESSAGE_OF_OPERATE_VNF, RECEIPT_OF_REQUEST_MESSAGE_OF_CHANGE _VNF_EXT_CONN, RECEIPT_OF_REQUEST_MESSAGE_OF_VNFINFO_MODIFICATION, RECEIPT_OF_VNF_INDICATOR_VALUE_CHANGE_NOTIFICATION	
EtherType	IPV4, IPV6	
EventType	COMMUNICATION_ALARM, PROCESSING_ALARM, ENVIRONMENT_ALARM, QOS_ALARM, EQUIPMENT_ALARM	
FaultyResourceType	COMPUTE, STORAGE, NETWORK	
ForwardingBehaviourRule	ALL, LB	
ForwardingBehaviourType	ALL, LB	X
InformationChangeType	ADDITION, REMOVAL, UPDATE	
InstantiationState	NOT_INSTANTIATED, INSTANTIATED	
InterfaceNames	VNF_CONFIGURATION, VNF_INDICATOR, VNF_LCM_COORDINATION	
IpAddressType	IPV4, IPV6	
Ipv6AddressMode	SLAAC, DHCPV6-STATEFUL, DHCPV6-STATELESS	
IpVersion	IPV4, IPV6	
L2NetworkType	FLAT, VLAN, VXLAN, GRE	
LayerProtocol	Ethernet, MPLS, ODU2, IPV4, IPV6, Pseudo-Wire	X
LmcOperationStatus	STARTING, PROCESSING, COMPLETED, FAILED_TEMP, FAILED ROLLING_BACK, ROLLED_BACK	X

Enum	Values	Expandable
LcmScriptEventType	EVENT_START_INSTANTIATION, EVENT_END_INSTANTIATION, EVENT_START_SCALING, EVENT_END_SCALING, EVENT_START_SCALING_TO_LEVEL, EVENT_END_SCALING_TO_LEVEL, EVENT_START_HEALING, EVENT_END_HEALING, EVENT_START_TERMINATION, EVENT_END_TERMINATION, EVENT_START_VNF_FLAVOR_CHANGE, EVENT_END_VNF_FLAVOR_CHANGE, EVENT_START_VNF_OPERATION_CHANGE, EVENT_END_VNF_OPERATION_CHANGE, EVENT_START_VNF_EXT_CONN_CHANGE, EVENT_END_VNF_EXT_CONN_CHANGE, EVENT_START_VNFINFO_MODIFICATION, EVENT_END_VNFINFO_MODIFICATION, EVENT_START_VNF_SNAPSHOT_CREATION, EVENT_END_VNF_SNAPSHOT_CREATION, EVENT_START_VNF_SNAPSHOT_REVERTINGTO, EVENT_END_VNF_SNAPSHOT_REVERTINGTO, EVENT_START_CHANGE_CURRENT_VNF_PACKAGE, EVENT_END_CHANGE_CURRENT_VNF_PACKAGE xor receipt of request message of instantiation/scaling/healing/termination, change of VNF flavour, change of the operation state of the VNF, change of external VNF connectivity, creation of and reverting to VNF snapshot, change of current VNF Package, modification of VNF information, receipt of a notification regarding the change of a VNF indicator value	
ManoConsumerInterfaceType (see note)	when consuming from VIM: Sim, Vcrm, Vcrim, Vcrcm, Vcrcn, Vcfm, Vnrm, Vnrim, Vnrcm, Vnrcn, Nfpm, Vsrm, Vsrim, Vsrcm, Vsrcn, Vrpm, Vrfm, Vcrmm, Vnrmm, Vsrmm, Vrrcn, Vcrqm, Vnrqm, Vsrqm, Vrqcn	
	when consuming from VNFM: Vnflcm, Vnfpm, Vnffm, Vnfind	
	when consuming from NFVO: Vnfpkgm, Vnflcog, Vrim, Vrm, Vrrm, Vrrcn, Vrcn, Vrpm, Vrfm, Vrqm, Vrqan	
ManoEntityInterfaceType (see note)	for VIM: Sim, Vcrm, Vcrim, Vcrcam, Vcrcn, Vcfm, Vnrm, Vnrim, Vnrcam, Vnrcn, Nfpm, Vsrm, Vsrim, Vsrcam, Vsrcn, Vrpm, Vrfm, Vcrmm, Vnrmm, Vsrmm, Vrrcn, Vcrqm, Vnrqm, Vsrqm, Vrqcn, Chrm, Chcam, Pom	
	for VNFM: Vnflcm, Vnfpm, Vnffm, Vnfind, Pom, Vnfspm	
	for NFVO: Nsd, Vnfpkgm, Nslcm, Nspm, Nsfm, Vnflcog, Vrim, Vrm, Vrrm, Vrrcn, Vrcn, Vrpm, Vrfm, Vrqm, Vrqan, Pom	
ManoEntityType	NFVO, VNFM, VIM	
MigrationType	NO_MIGRATION, OFFLINE_MIGRATION, LIVE_MIGRATION	
ModificationQualifier	UP, DOWN	
MSNCDirectionality	UNIDIRECTIONAL, BIDIRECTIONAL	
NestedNsChangeType	ADD, REMOVE, INSTANTIATE, TERMINATE, SCALE, HEAL, UPDATE	
NetworkResourceType	NETWORK, SUBNET, NETWORK_PORT	
NsComponentType	VNF, nestedNS, PNF	
NsDegreeHealing	COMPLETE_HEALING_OF_THE_NS_RESTORING_THE_STATE_OF_THE_NS_BEFORE_THE_FAILURE_OCCURRED, COMPLETE_HEALING_BASED_ON_THE_NEWEST_QOS_VALUES, COMPLETE_HEALING_RESETTING_TO_THE_ORIGINAL_INSTANTIATION_STATE_OF_THE_NS, PARTIAL_HEALING	
NsInstanceUsageStatus	START, END	
NsLifecycleOperation	SCALE_NS, TERMINATE_NS, HEAL_NS	
NsScaleDirection	SCALE_IN, SCALE_OUT	
ObjectType	LINK, NODE, TOPOLOGY, NETWORK	
OnboardingState	CREATED, UPLOADING, PROCESSING, ONBOARDED	
OperationAction	ABORT, CONTINUE, CONTINUE_AFTER_DELAY, RETRY_AFTER_DELAY	X
OperationalState	ENABLED, DISABLED	
OperationStatus	START, RESULT	
PerceivedSeverity	CRITICAL, MAJOR, MINOR, WARNING, INDETERMINATE, CLEARED	
PnfChangeType	ADD, MODIFY, REMOVE	
PolicyChangeOperations	TRANSFER_POLICY, DELETE_POLICY, ACTIVATE_POLICY, DEACTIVATE_POLICY	

Enum	Values	Expandable
ProtectionScheme	UNPROTECTED, <0:1>, <1:1>, <1+1>, <1:N>, <M:N>	
Protocol	TCP, UDP, ICMP	X
ReservationStatus	RESERVATION_BEING_USED, RESERVATION_NOT_USED	
ResourceDefinitionType	COMPUTE, VL, LINKPORT, STORAGE	X
ResourceMgmtModeSupport	DIRECT, INDIRECT, BOTH	
SapChangeType	ADD; REMOVE, MODIFY	
SegmentationType	VLAN, INHERIT	
ServiceAvailabilityLevel	LEVEL_1, LEVEL_2, LEVEL_3	
SnapshotPkgState	CREATED, BUILDING, UPLOADING, AVAILABLE	
StorageType	BLOCK, OBJECT, FILE	
SupportedOperations	Scale VNF, Scale VNF to Level, Heal VNF, Operate VNF	X
ThresholdCrossing	UP, DOWN	
ThresholdType	SIMPLE	X
UsageState	IN_USE, NOT_IN_USE	
UserDataTransportationMethod	CONFIG-DRIVE	
VipFunction	high availability, load balancing	
VirtualLinkChangeType	ADD, DELETE, MODIFY, ADD_LINK_PORT, REMOVE_LINK_PORT	
VirtualStorageChangeType	ADDED, REMOVED, MODIFIED, TEMPORARY	
Visibility	PRIVATE, PUBLIC	
VnfcChangeType	ADDED, MODIFIED, REMOVED, TEMPORARY	
VnfChangeType	ADD, REMOVE, INSTANTIATE, TERMINATE, SCALE, HEAL, OPERATE, MODIFY_INFORMATION, CHANGE_FLAVOUR, CHANGE_EXT_VNF_CONNECTIVITY, REVERT_TO_VNF_SNAPSHOT, CHANGE_CURRENT_VNF_PKG, ASSOCIATE_WITH_VNF_PROFILE	
VnfcState	STARTED, STOPPED	
VnffgChangeType	ADD, REMOVE, MODIFY	
VnfIndicatorSource	VNF, EM, Both	
VnfLifecycleOperation	InstantiateVnf, ScaleVnf, ScaleVnfToLevel, ChangeVnfFlavour, TerminateVnf, HealVnf, OperateVnf, ChangeExtVnfConnectivity, CreateSnapshot, RevertToSnapshot, ChangeCurrentVnfPackage	
VnfPackageChangeType	CHANGE_OF_OPERATIONAL_STATE, DELETION_OF_A_VNF_PACKAGE	
VnfScaleType	SCALE_OUT, SCALE_IN, SCALE_UP, SCALE_DOWN, SCALE_TO_INSTANTIATION_LEVEL, SCALE_TO_SCALE_LEVEL	
VnfState	STARTED, STOPPED	
VnfStopType	FORCEFUL, GRACEFUL	
VnicType	NORMAL, MACVTAP, DIRECT, BAREMETAL, VIRTIO-FORWARDER, DIRECT-PHYSICAL, SMART-NIC, BRIDGE, IPVLAN, LOOPBACK, MACVLAN, PTP, VLAN, HOST-DEVICE	

NOTE: The list of abbreviations for the interfaces used in the attributes 'ManoConsumerInterfaceType' or 'ManoEntityInterfaceType' are as follows:

- For VIM: Sim (Software Image Management), Vrpm (Virtualised Resources Performance Management), Vrfm (Virtualised Resources Fault Management), Vrqcn (Virtualised Resources Quota Change Notification), Chrm (Compute Host Reservation Management), Chcam (Compute Host Capacity Management), Pom (Policy Management), Vrrcn: Virtualised Resources Reservation Change Notification:
    - virtualised compute interfaces: Vcrm (Virtualised Compute Resources Management), Vcrcm (Virtualised Compute Resources Capacity Management), Vcrim (Virtualised Compute Resources Information Management), Vcrcam (Virtualised Compute Resources Capacity Management), Vcrcn (Virtualised Compute Resources Change Notification), Vcfm (Virtualised Compute Flavour Management).
    - virtualised network interfaces: Vnrm (Virtualised Network Resources Management), Vnrcm (Virtualised Network Resources Capacity Management), Vnrim: Virtualised Network Resources Information Management), Vnrcam (Virtualised Network Resources Capacity Management), Vnrcn (Virtualised Network Resources Change Notification), Nfpm (Network Forwarding Path Management).

    - virtualised storage interfaces: Vsrm (Virtualised Storage Resources Management), Vsrcm (Virtualised Storage Resources Capacity Management), Vsrim (Virtualised Storage Resources Information Management), Vsrcam (Virtualised Storage Resources Capacity Management), Vsrcn (Virtualised Storage Resources Change Notification).

    - virtualised resource reservation interfaces: Vcrmm (Virtualised Compute Resources Reservation Management), Vnrmm (Virtualised Network Resources Reservation Management), Vsrmm (Virtualised Storage Resources Reservation Management).

    - virtualised resource quota interfaces: Vcrqm (Virtualised Compute Resources Quota Management), Vnrqm (Virtualised Network Resources Quota Management), Vsrqm (Virtualised Storage Resources Quota Management).

- For VNFM: Vnflcm (VNF Lifecycle Management), Vnfpm (VNF Performance Management), Vnffm (VNF Fault Management), Vnfind (VNF Indicator), Pom (Policy Management), Vnfspm (VNF Snapshot Package Management).

- For NFVO: Nsd (NSD Management), Vnfpkgm (VNF Package Management), Nslcm (NS Lifecycle Management), Nspm (NS Performance Management), Nsfm (NS Fault Management),Vnflcog (VNF Lifecycle Operation Granting), Vrim (Virtualised Resources Information Management), Vrm (Virtualised Resources Management),Vrrm (Virtualised Resources Reservation Management), Vrrcn (Virtualised Resources Reservation Change Notification),Vrcn (Virtualised Resource Change Notification),Vrpm (Virtualised Resources Performance Management),Vrfm (Virtualised Resources Fault Management),Vrqm (Virtualised Resources Quota Management),Vrqan (Virtualised Resources Quota Available Notification), Pom (Policy Management.

Table A.8-2 shows other Enum values found in ETSI GR NFV-IFA 015 [i.9], but which are not related to any information element or type listed in the present annex.

Table A.8-2: Other Enum values

Enum	Values
ConnectionPointType	LAYER_1, LAYER_2, LAYER_3
ExternalStimulusEventType	RECEIPT_OF_REQUEST_MESSAGE_OF_INSTANTIATION, RECEIPT_OF_REQUEST_MESSAGE_OF_SCALING, RECEIPT_OF_REQUEST_MESSAGE_OF_HEALING, RECEIPT_OF_REQUEST_MESSAGE_OF_TERMINATION, RECEIPT_OF_REQUEST_MESSAGE_OF_CHANGE _VNF_FLAVOUR, RECEIPT_OF_REQUEST_MESSAGE_OF_OPERATE_VNF, RECEIPT_OF_REQUEST_MESSAGE_OF_CHANGE _VNF_EXT_CONN, RECEIPT_OF_REQUEST_MESSAGE_OF_VNFINFO_MODIFICATION, RECEIPT_OF_VNF_INDICATOR_VALUE_CHANGE_NOTIFICATION
NsdInfoChangeType	CHANGE_OF_OPERATIONAL_STATE_OF_AN_ON-BOARDED_NSD, NSD_IN_DELETION_PENDING, DELETION_OF_AN_NSD
VirtualCpuPolicy	dedicated, shared
VnfLcmEventType	EVENT_START_INSTANTIATION, EVENT_END_INSTANTIATION, EVENT_START_SCALING_IN, EVENT_END_SCALING_IN, EVENT_START_SCALING_OUT, EVENT_END_SCALING_OUT, EVENT_START_SCALING, EVENT_END_SCALING, EVENT_START_SCALING_TO_LEVEL, EVENT_END_SCALING_TO_LEVEL, EVENT_START_HEALING, EVENT_END_HEALING, EVENT_START_TERMINATION, EVENT_END_TERMINATION, EVENT_START_VNF_FLAVOR_CHANGE, EVENT_END_VNF_FLAVOR_CHANGE, EVENT_START_VNF_OPERATION_CHANGE, EVENT_END_VNF_OPERATION_CHANGE, EVENT_START_VNF_EXT_CONN_CHANGE, EVENT_END_VNF_EXT_CONN_CHANGE, EVENT_START_VNFINFO_MODIFICATION, EVENT_END_VNFINFO_MODIFICATION, EVENT_START_VNF_SNAPSHOT_CREATION, EVENT_END_VNF_SNAPSHOT_CREATION, EVENT_START_VNF_SNAPSHOT_REVERTINGTO, EVENT_END_VNF_SNAPSHOT_REVERTINGTO
VnfScaleByStepType	SCALE_OUT, SCALE_IN

Annex B: NFV interfaces and operations

Using the domain classification of ETSI GR NFV-IFA 015 [i.9], Tables B-1, B-2 and B-3 list the operations found at the different interfaces classified through the list shown in clause 5.1.

Table B-1: Operations at the different interfaces

Domain	Interface	Operation	Establishment/Removal: create, activate, associate, upload, fetch, instantiate, add, allocate, attach, build, transfer, extract / delete, stop, deactivate, disassociate, terminate, detach	Modification: modify, change, update, scale, migrate	Questioning: query, get (alarm list, operation status, indicator value)	Incidents: escalate severity, ACK alarms, heal, revert-to snapshot	Other LCM operations: grant (NS/VNF lifecycle), coordinate, operate, set (configuration)	Subscription: initialization, termination, information query	Notification
NFV common	Fault management	Get Alarm List			X				
		Escalate Perceived Severity				X			
		Acknowledge Alarms				X			
		Subscribe						X	
		Terminate Subscription						X	
		Query Subscription Info						X	
		Notify							X
	Performance management	Create PM Job	X						
		Delete PM Jobs	X						
		Query PM Job			X				
		Create Threshold		X					
		Delete Thresholds		X					
		Query Threshold			X				
		Subscribe						X	
		Terminate Subscription						X	
		Query Subscription						X	
		Notify							X
	Policy management	Transfer Policy	X						
		Delete Policy	X						
		Query Policy			X				
		Activate Policy	X						
		Deactivate Policy	X						
		Associate Policy	X						
		Disassociate Policy	X						
		Subscribe						X	
		Terminate Subscription						X	
		Query Subscription Info						X	
		Notify							X
NFV-MANO OAM	NFV-MANO configuration and information management	Modify Config		X					
		Query Config Info			X				
		Change State		X					
		Subscribe						X	
		Terminate Subscription						X	
		Query Subscription Information						X	
		Notify							X
	NFV-MANO log management	Create Logging Job	X						
		Stop Logging Job	X						
		Query Logging Job			X				
		Subscribe						X	
		Terminate Subscription						X	
		Query Subscription Info						X	
		Notify							X
NS	NSD management	Upload NSD	X						
		Update NSD Info							
		Delete NSD	X						
		Create NSD Info	X						
		Query NSD Info	X						
		Fetch NSD	X						
		Fetch NSD Archive Artifacts	X						
		Upload PNFD	X						
		Update PNFD Info		X					
		Delete PNFD	X						
		Create PNFD Info	X						
		Query PNFD Info			X				
		Fetch PNFD	X						
		Fetch PNFD Archive Artifacts	X						
		Subscribe						X	
		Terminate Subscription						X	
		Query Subscription Info						X	
		Notify							X
	NS lifecycle management	Create NS Identifier	X						
		Delete NS Identifier	X						
		Instantiate NS	X						
		Scale NS		X					
		Update NS		X					
		Heal NS				X			
		Terminate NS	X						
		Query NS			X				
		Get Operation Status							
		Subscribe						X	
		Terminate Subscription						X	
		Query Subscription Info						X	
		Notify							X
	NS instance usage notification	Subscribe						X	
		Terminate Subscription						X	
		Query Subscription						X	
		Notify							X
	NS lifecycle operation granting	Grant NS Lifecycle					X		
	LCM coordination	CoordinateLcmOperation					X		

Table B-2: Operations at the different interfaces (cont.)

Resource	Software image management	Add Image	X						
		Query Images			X				
		Query Image			X				
		Update Image		X					
		Delete Image	X						
	Virtualised c/n/s resources management	Allocate Virtualised C/N/S Resource	X						
		Query Virtualised C/N/S Resource			X				
		Update Virtualised C/N/S Resource		X					
		Terminate Virtualised C/N/S Resource	X						
		Operate Virtualised C/S Resource					X		
		Scale Virtualised C/S Resource		X					
		Migrate Virtualised C/S Resource		X					
		Create Virtualised C/N/S Resource A Or AA Constraints Group	X						
		Attach Virtualised Storage Resource	X						
		Detach Virtualised Storage Resource	X						
	Virtualised c/n/s resources change notification	Subscribe						X	
		Notify							X
	Virtualised c/n/s resources information management	Query Virtualised C/N/S Resource Information			X				
		Subscribe						X	
		Notify							X
	Virtualised c/n/s resources capacity management	Query C/N/S Capacity			X				
		Query Compute/Storage Resource Zone			X				
		Query NFVI-PoP C/N/S Information			X				
		Subscribe						X	
		Notify							X
	Virtualised compute flavour management	Create Compute Flavour	X						
		Query Compute Flavour			X				
		Delete Compute Flavour	X						
	Network forwarding path management	Create NFP	X						
		Query NFP			X				
		Delete NFP	X						
		Change NFP State		X					
		Update NFP		X					
	Virtualised c/n/s resources reservation management	Create C/N/S Resource Reservation	X						
		Query C/N/S Resource Reservation			X				
		Update C/N/S Resource Reservation		X					
		Terminate C/N/S Resource Reservation	X						
	Virtualised resources reservation change notification	Subscribe						X	
		Notify							X
	Virtualised c/n/s resources quota management	Create C/N/S Resource Quota	X						
		Query C/N/S Resource Quota			X				
		Update C/N/S Resource Quota		X					
		Terminate C/N/S Resource Quota	X						
	Virtualised resources quota change notification	Subscribe						X	
		Notify							X
	NFVI capacity information	Query NFVI capacity			X				
		Create Capacity Threshold	X						
		Delete Capacity Thresholds	X						
		Query Capacity Threshold			X				
		Subscribe						X	
		Terminate Subscription						X	
		Query Subscription Info						X	
		Notify							X
	MSCS management	Create MSCS	X						
		Query MSCS			X				
		Update MSCS		X					
		Terminate MSCS	X						
		Create MSCS Reservation	X						
		Query MSCS Reservation			X				
		Update MSCS Reservation		X					
		Terminate MSCS Reservation	X						
		Subscribe						X	
		Query Subscription Info						X	
		Terminate Subscription						X	
		Notify							X
	MSCS capacity management	Query Capacity			X				
		Create Capacity Threshold	X						
		Delete Capacity Thresholds	X						
		Query Capacity Threshold			X				
		Query Topology Information			X				
		Query Node Information			X				
		Query Link Information			X				
		Query Network Edge Point Information			X				
		Subscribe						X	
		Terminate Subscription						X	
		Query Subscription						X	
		Notify							X
	Compute host reservation management	Create Compute Host Reservation	X						
		Query Compute Host Reservation			X				
		Update Compute Host Reservation		X					
		Terminate Compute Host Reservation operation	X						
	Compute host capacity management	Query Compute Host Capacity			X				
		Subscribe						X	
		Notify							X
	Virtualised resources quota available notification	Subscribe						X	
		Terminate Subscription						X	
		Query Subscription Info						X	
		Notify							X

Table B-3: Operations at the different interfaces (cont.)

VNF	**VNF configuration**	Set Configuration					X		
	VNF lifecycle operation granting	Grant VNF Lifecycle Operation					X		
	VNF indicator	Get Indicator Value			X				
		Subscribe						X	
		Terminate Subscription						X	
		Query Subscription Info						X	
		Notify							X
	VNF lifecycle management	Create VNF Identifier	X						
		Instantiate VNF	X						
		Scale VNF		X					
		Scale VNF to Level		X					
		Change VNF Flavour		X					
		Terminate VNF	X						
		Delete VNF Identifier	X						
		Query VNF			X				
		Heal VNF				X			
		Operate VNF					X		
		Modify VNF Information		X					
		Get Operation Status			X				
		Change External VNF Connectivity		X					
		Query Snapshot Information			X				
		Create Snapshot	X						
		Revert-to Snapshot			X				
		Delete Snapshot Information	X						
		Change current VNF package		X					
		Fetch VNF state snapshot	X						
		Subscribe						X	
		Terminate Subscription						X	
		Query Subscription Info						X	
		Notify							X
	VNF package management	Upload VNF Package	X						
		Delete VNF Package	X						
		Create VNF Package Info	X						
		Update VNF Package Info		X					
		Query VNF Package Info			X				
		Fetch VNF Package	X						
		Fetch VNF Package Artifacts	X						
		Subscribe						X	
		Terminate Subscription						X	
		Query Subscription Info						X	
		Notify							X
	VNF snapshot package management	Fetch VNF Snapshot Package	X						
		Fetch VNF Snapshot Package Artifacts	X						
		Query VNF Snapshot Package Information			X				
		Create VNF Snapshot Package Info	X						
		Build VNF Snapshot Package	X						
		Upload VNF Snapshot Package	X						
		Extract VNF Snapshot Package	X						
		Delete VNF Snapshot Package	X						
		Update VNF Snapshot Package		X					
	LCM coordination	CoordinateLcmOperation					X		

Annex C: ANN model creation experiment for the service availability assurance use case

To demonstrate the creation of ANN models for the VNFs of the sample NS (see clause 7.1.2.2.1.1), an experiment was conducted using the analytical models described in [i.14] to generate labels with the label structures shown in Figure 7.1.2.2.2.1-1:

- for a single ANN model using label structure A; and also
- for two chained ANN models using respectively label structures B and C.

To verify the approach, the ANN models were implemented in Python™ using the TensorFlow library. For both cases, 76 000 labels were generated using the analytical models. 90 % of the generated labels were used to train the ANN models and 10 % for validation.

To determine suitable hyperparameters, ANN models were created with different hyperparameter configurations and trained using the training set in short training sessions (between 2 000 and 10 000 epochs). During each session the loss function was observed if the configuration was converging as desired. If this was not the case the ANN model configuration was discarded. After some trials, the number of hidden layers was set to 19 and the number of nodes for each hidden layer was set to 35. These values were set regardless whether a single-ANN model or the chained-ANN models were used. It is expected that the same values can be used to model other NSs as well for similar requirements.

The ANN models were trained in all cases for 20 000 epochs (i.e. training cycles).

In case of the single ANN, the training took 1 hour and 57 minutes, while for the chained ANNs, it took 4 hours and 21 minutes using the same hardware.

The trained prototypes were also checked using the validation portion of the generated labels.

Table C-1 shows the standard deviation for each output parameter (i.e. number of StandBys (SB), Health-check Interval (HI), Checkpointing Interval (CpI)) predicted by the chained ANN models and the respective output value in the validation set, that is, the optimal value determined by the analytical models.

Table C-1: Validation results for the chained ANN models for predicting new configuration values

Parameter	Average	Standard deviation
VNF1_HI	1 071,332	63,486
VNF1_CpI	882,395	26,925
VNF2_HI	698,098	35,447
VNF2_CpI	50,000	0,000
VNF3_HI	2 564,516	119,640
VNF3_CpI	487,007	10,394
VNF1_SB	2 072	0,111
VNF2_SB	3 040	0,170
VNF3_SB	1 000	0,000

To compare the two solutions, i.e. single-ANN model and chained-ANN models, the number of StandBys (SB) generated by the two solutions were compared in the validation results. As discussed in clause 7.1.2.2.2.1, chaining the ANN models decouples this output parameter from others, namely the Health-check Interval (HI).

In Table C-2, the average values represent the values expected based on the analytical models and are the same for the two solutions. The standard deviation shows the deviation of the predicted values compared to these average values. As shown in Table C-2 the standard deviations for the number of standbys of the different VNFs have decreased as a result of chaining two models. I.e. decoupling the output parameters resulted in a better learning at the cost of the increase of the training time, which has more than doubled.

Table C-2: Validation results for the number of standbys

	Average value	Standard deviation	
		Single ANN	Chained ANNs
VNF1	2 072	0,356	0,111
VNF2	3 040	0,411	0,170
VNF3	1 000	0,137	0,000

Considering that the number of standbys is an integer, this difference might not justify the extra efforts, however, the case would be similar for other cases of coupled output parameters.

Annex D: Illustrations for the fault localization use case

D.1 Data collection, pre-processing and model building

To validate the fault localization method using 2-layered SOMs, [i.26] performed different experiments in a testbed modeling a video streaming cloud service. This testbed consisted of three main parts:

- Server Side, where the X traces were collected with the device statistics and could be equated to NFVI KPIs. The device statistics X_i of each server S_i (where i = 1…n) were collected at the operating system level on server S_i for both physical and virtual components. These metrics included, but were not limited to, CPU, memory, I/O operations, and network statistics. The device statistics *X* of the cluster formed by all servers was the union of the server statistics: $X = (X_1 \cup X_2 \cup ... X_n)$.

- Client Side, where the Y traces were collected with service level metrics such as displayed video frames, audio buffer rate and video frame rate. These metrics can correspond to NS and/or VNF level metrics collected on the corresponding performance and/or indicator APIs.

- Load Generator, which was used to create the different contexts for the measurements including the generation of specific traffic load patterns (e.g. constant load, periodic load) and injection of faults (i.e. CPU hog, memory hog and I/O hog) according to a specific probability distribution and for a specific duration.

The metrics *X* and *Y* evolved over time and were influenced by the load generated towards the servers, the operating system dynamics, and the injected faults. Assuming a global clock, these series of metrics could be represented as time series $\{X_t\}_t$, $\{Y_t\}_t$, and $\{(X_t, Y_t)\}_t$, which were considered as the training and validation data.

To enable supervised learning for the second layer of the SOM, a service level agreement (SLA) was defined for the client-side service level metrics, which, considering a threshold, could be evaluated as being either in a 'violated' or 'fulfilled' state at any given moment in time. Accordingly, for each service metric Y_t at time instance *t* an SLA_t value was computed and added to Y_t.

Data traces were collected for different scenarios at the server side. These included constant load with a specific type (CPU, memory and I/O) of faults injected; constant load with all types of faults injected, periodic load with CPU faults injected, and periodic load with all types of faults injected. Each trace lasted for a total of 10 hours resulting in approximately 36 000 samples per trace. Faults were injected every 30 seconds with a certain probability, and they persisted for a specified duration. For cases where all types of faults were injected, the type of the fault was selected randomly.

The original traces contained data for a large number of features (i.e. 648) out of which 15 were selected through feature engineering to be used for training. These included CPU utilization at host/container levels, used/committed memory, used/cached swap, read/written blocks, received/transmitted packets/data.

Since the collected data varied widely, the pre-processing steps of normalization and data smoothing were applied. Depending on the characteristics of the collected data, other pre-processing steps may also be performed.

Using the collected and pre-processed traces, different maps were trained. I.e. from traces that contained all types of faults, generalized maps were trained, while based on those that contained only a specific fault type, specialized maps were trained and validated. In each case, the K-fold cross-validation process was applied.

K-fold cross-validation is particularly useful when the available data set is limited. It produces less biased results than just splitting the data into training and validation sets. In case of K-fold cross-validation, the available data is split into K equal data sets. Then K models are trained using K-1 data sets while setting aside the remaining one data set for validation. This way each of the K data sets is used for validation exactly once. The prediction errors calculated by the validations (e.g. mean squared error) help to assess the prediction error of the model. If the training model has a hyperparameter, then the average prediction error is a function of this parameter, which then can be minimized to construct an optimal model. For more details on cross-validation, see [i.25].

In [i.26], K = 3 was selected and, accordingly, the data were split into three equal parts and three maps were trained each with a different 2/3 of the data and validated with the remaining 1/3. The validation results were used to select the best performing map.

D.2 Fault localization performance

To evaluate the fault localization performance of the trained generalized and specialized maps, two different performance criteria were used.

a) First, it was calculated how many times each fault type was correctly considered as the primary cause of a service degradation. The localization result was considered good if the actual injected fault type was correctly detected by showing most fault occurrences.

b) Secondly, a fault localization accuracy was calculated as:

$$Localization\ Accuracy = \frac{Number\ of\ correct\ localizations}{Total\ number\ of\ localizations\ performed}$$

where $Number\ of\ correct\ localizations$ was equal to the number of localizations where the injected fault was ranked as one of the *top three* possible faults.

When comparing generalized and specialized maps using the above criteria regarding memory faults, the observation was that when considering the top three possible faults as in localization accuracy b), the maps performed comparably (i.e. 0,963 for specialized map and 0,989 for the generalized maps). However, when considering criterion a) - identifying the primary fault type, the specialized map identified incorrectly the CPU fault as the primary fault type 6 011 times out of 6 300, while the generalized map correctly identified the memory fault as the primary cause 4 102 times out of 6 937.

Similar "confusion" of the specialized map could be observed with respect to I/O faults. That is, according to b), the localization accuracies were 0,969 and 0,998 for the specialized and generalized maps, respectively. According to a), however, the specialized map failed to identify the I/O fault as primary. Instead, 1 149 times it has identified the memory fault incorrectly as primary, and only 955 times it identified correctly the I/O fault (i.e. it has appeared as secondary) out of 2 452 cases. The generalized map identified the I/O fault 1 612 times correctly out of 1 889 cases.

[i.26] suggests that the reason behind the better capability of differentiation of the generalized map was that it was trained on data that varied more on different faults. It is thus recommended to use generalized maps as the optimal choice for fault localization.

Annex E: Illustrations for the anomaly prediction use case

E.1 Anomaly prediction using syslogs

[i.16] has used syslogs produced in a real-world deployment of virtualised Provider Edge routers (vPE) in order to design and validate a LSTM-based anomaly detection system which identifies conditions that correlate with network trouble tickets. 18 consecutive months of data were used in order to train and validate the model. A preliminary analysis of network trouble tickets showed that their root cause is related to:

- maintenance, i.e. expected or scheduled network actions or changes;
- circuit, i.e. the loss of connection between two devices on specific interfaces;
- cable, i.e. cable disconnection due to environmental or human artifacts;
- hardware, i.e. failures of cards that constitute the chassis system and components that constitute a card;
- software, i.e. software issues;
- duplicate, i.e. follow-up failures when the original issue was not resolved.

Following the initial training of the LSTM model using one subset of the data (month 1), another subset (month 2) was exploited for the validation of the model through monthly detection of anomalies, followed by the mapping of the detected anomalies to relevant trouble tickets. The model was then repeatedly updated and validated with subsequent subsets of monthly data.

Concerning the types of network trouble tickets showing early signs in the syslog, the collected data show, that VNF syslog messages (i.e. anomalies called early warning signals or ticket-triggering signatures in clause 7.3.2.2.3) appear for multiple trouble ticket types: 74 % for circuit, 55 % for software, 40 % for cable and 28 % for hardware.

According to [i.16], *"The majority of detected syslog anomalies are 5 min ahead of the ticket generation. For circuit, 36 % of syslog anomalies are 15 min ahead, and the ratio is even higher for cable (39 %) and hardware (38 %) categories. Although more in-depth investigation is required, these results indicate the possibility that operators may be able to leverage these syslog anomalies to either improve their ticketing process, or identify predictive or early conditions indicative of network failures*".

Finally, some failures do not display syslog anomalies before ticket generation. Analysing these failures shows that for the majority of their tickets (80 %), syslogs will display anomalous patterns within 15 min after the ticket generation, i.e. as reported in [i.16], "*patterns of failures become visible at the NFV layer after a small delay, which can be leveraged by NFV for trouble ticket analysis, diagnosis and management*".

E.2 Comparison of prediction models using KPIs

To evaluate the anomaly prediction models, [i.17] has used four KPIs computed from LTE operations data: with a step size of 15 min for an observation window of 24 hours, 96 values were provided for each KPI. As data were collected during 68 days, the total number of observations was 6 573 per KPI.

The considered KPIs were:

- Average Latency (inside the cell)
- Average Active User (within the cell)
- Downlink Traffic Volume
- Downlink Load (utilization rate of physical resource block in downlink)

These KPIs are typically used for detecting and analysing congestions. Because of the random distribution of users over the cells and the variety of sessions, some cells are more loaded than others: when the load is unbalanced between cells, a congestion can happen, and the victim cell can no longer serve the users located in its coverage area. If one of the KPIs indicated above shows degradation, i.e. exceeding a certain threshold, it is usually followed by a congestion.

[i.17] has studied four prediction models: Linear Regression (LiReg) and Distributed-Log Regression (DLR) based on discrete dataset, Logistic Regression (LoReg) and Random Forest (RF) using functional dataset. To measure the quality of the prediction models, four indicators were used:

- *Accuracy* (A = $\frac{TP+TN}{TP+TN+FP+FN}$) defines the proportion of true predictions;
- *Recall* (R = $\frac{TP}{TP+FN}$) is defined by the ratio of the detected anomalies to the total number of anomalies - a low value of R means that the method does not predict anomalies well;
- *Precision* (P = $\frac{TP}{TP+FP}$) measures the reliability of anomaly prediction - when P is low, the number of false alarms (false alert of the presence of an anomaly) is high;
- *F-measure* (F = $\frac{2\,P*R}{P+R}$) is the weighted harmonic mean of the precision and the recall - a high value of F-measure means that both precision and recall are high.

Where TP stands for True Positive, TN for True Negative, FP for False Positive and FN for False Negative.

The performance of congestion prediction using discrete data and based on linear regression was studied first: m = 96 measurements were considered per KPI and used for the DLR model while for the functional dataset-based models each KPI observation (96 measurements) was represented by M = 25 coefficients after passing through the smoothing block.

The study was realized with a prediction horizon h varying between 15 min and 4 hours since it was expected that with the increase of h, the performance of the model would degrade. Indeed, it was observed that, for example, the performance of the LiReg model degraded because the correlation between the future congestion and the actual values of the KPIs fell especially when the temporary shift was important. For h > 1 hour, the correlation became weak, i.e. the model was not able to reliably predict future congestions one hour (or more) in advance since R ~ 55 %.
With h = 2 hours, the model detected only 45 % of the anomalies and 32 % of them were false alarms (P = 68 %).

[i.17] has computed the accuracy and F-measure for the four prediction models. The value of h has been varied between 0 (detection) and 4 hours, to understand whether congestion prediction might allow to perform, e.g. proactive load balancing. Different interpretations were derived from the results of this study:

- When h < 15 min, LiReg was as successful as other robust methods that perform data processing before setting up the prediction model, i.e. LoReg and RF - meaning that there was a strong correlation in such cases between future anomalies and current KPI measurements.
- DLR performance was lower compared to LiReg when h was close - actually, the use of several previous measurements per KPI weakened the correlation between the forecasted variable and the predictors for such horizon. When h was far, the correlation between future congestion and KPIs has weakened: thus trend-based prediction of anomalies gained significance, as shown by the improved performance of DLR compared to LiReg.
- As DLR suffered from the noisy KPI measurements, the transition to functional data improved the prediction performance. When h = 30 min, the accuracy was 74 % for LoReg (vs. 65 % for DLR) and the F-measure was 73 % for LoReg (vs. 63 % for DLR) - actually, KPI values collected by the network corresponded to an average of 15 min and this average did not reflect the KPI fluctuation which could be fast because of the random behaviour of the users leading to measurement errors.
- LoReg and RF had the potential to effectively prevent future congestions even 4 hours in advance, unlike LiReg that failed to predict the anomaly 1 hour in advance. This was due to the functional data processing based on the smoothing and the Principal Components Analysis (PCA) which managed to extract a maximum of correlation between the variables and reduce the amount of noise associated with the data.
- LoReg performances were close to the RF ones, e.g. when h = 1 hour, the accuracy and F-measure were respectively 71 % and 70 % for LoReg, and 74 % and 75 % for RF.

Finally, the complexity of different prediction models can be apprehended through the construction/training time of the models. The study showed that models based on functional data (with construction/training time resp. 8 min for LoReg and 15 min for RF), were more complex than the ones based on discrete data (resp. 30 sec for LiReg and 1,3 min for DLR), since the collected KPI measurements had to be projected on a function basis with a smoothing method and a matrix calculation was required using the PCA technique. RF appeared to be the most complex technique, as it performed parallel learning on multiple decision trees randomly constructed and trained on different subsets of data.

History

Document history		
V5.1.1	February 2023	Publication