ETSI TS 128 526 V15.0.0 (2018-07)

LTE;
Telecommunication management;
Life Cycle Management (LCM) for mobile networks that include virtualized network functions;
Procedures
(3GPP TS 28.526 version 15.0.0 Release 15)

Reference

RTS/TSGS-0528526vf00

Keywords

LTE

Intellectual Property Rights

Foreword

This Technical Specification (TS) has been produced by ETSI 3rd Generation Partnership Project (3GPP).

The present document may refer to technical specifications or reports using their 3GPP identities, UMTS identities or GSM identities. These should be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between GSM, UMTS, 3GPP and ETSI identities can be found under http://webapp.etsi.org/key/queryform.asp.

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**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.

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

x the first digit:

1 presented to TSG for information;

2 presented to TSG for approval;

3 or greater indicates TSG approved document under change control.

y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.

z the third digit is incremented when editorial only changes have been incorporated in the document.

Introduction

The present document is part of a TS-family covering the 3rd Generation Partnership Project Technical Specification Group Services and System Aspects, Telecommunication Management; as identified below:

TS 28.525: Telecommunication management; Life Cycle Management (LCM) for mobile networks that include virtualized network functions; Requirements.

TS 28.526: Telecommunication management; Life Cycle Management (LCM) for mobile networks that include virtualized network functions; Procedures.

TS 28.527: Telecommunication management; Life Cycle Management (LCM) for mobile networks that include virtualized network functions; Stage 2.

TS 28.528: Telecommunication management; Life Cycle Management (LCM) for mobile networks that include virtualized network functions; Stage 3.

1 Scope

The present document specifies the Life Cycle Management (LCM) procedures for mobile networks that include virtualized network functions.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] 3GPP TR 21.905: "Vocabulary for 3GPP Specifications".

[2] 3GPP TS 28.500: "Management concept, architecture and requirements for mobile networks that include virtualized network functions".

[3] 3GPP TS 28.525: "Life Cycle Management (LCM) for mobile networks that include virtualized network functions; Requirements".

[4] ETSI GS NFV-IFA008 V2.1.1 (2016-10) "Network Function Virtualization (NFV); Management and Orchestration; Ve-Vnfm Reference Point - Interface and Information Model Specification".

[5] ETSI GS NFV-IFA013 V2.1.1 (2016-10) "Network Function Virtualization (NFV); Management and Orchestration; Os-Ma-nfvo Reference Point - Interface and Information Model Specification".

[6] 3GPP TS 32.508: "Telecommunication management; Procedure flows for multi-vendor plug-and-play eNode B connection to the network".

[7] 3GPP TS 32.532: "Telecommunication management; Software management (SwM); Integration Reference Point (IRP); Information Service (IS)".

[8] ETSI GS NFV-IFA011 V2.1.1 (2016-10) "Network Function Virtualization (NFV); Management and Orchestration; VNF Packaging Specification".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in 3GPP TR 21.905 [1], 3GPP TS 28.500 [2] and the following apply. A term defined in the present document takes precedence over the definition of the same term, if any, in 3GPP TR 21.905 [1] or in 3GPP TS 28.500 [2].

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in 3GPP TR 21.905 [1], 3GPP TS 28.500 [2] and the following apply. An abbreviation defined in the present document takes precedence over the definition of the same abbreviation, if any, in 3GPP TR 21.905 [1] or in 3GPP TS 28.500 [2].

4 Lifecycle Management procedures

4.1 Introduction

The procedures listed in clause 4, as some of all the possibilities, are not exhaustive.

4.2 VNF Instance procedures

4.2.1 VNF Identifier creation

Figure 4.2.1-1 depicts the procedure of VNF instance identifier creation initiated through Ve-Vnfm-em reference point (see clause 7.2.2 of GS NFV-IFA008 [4])

1. EM sends to VNFM a *CreateVnfIdentifierRequest* with parameter `vnfdId`, `vnfInstanceName` and `vnfInstanceDescription` to create a VNF instance identifier (`vnfInstanceId`) and an associated instance of an `NsInfo` information element (see clause 7.2.2.2 of GS NFV-IFA008 [4]).

2. VNFM sends to EM a *CreateVnfIdentifierResponse* with parameter `vnfInstanceId` identifying the instance identifier of the VNF has been created (see clause 7.2.2.3 of GS NFV-IFA008 [4]).

3. VNFM sends to EM a *Notify* (see clause 7.3.3 of GS NFV-IFA008 [4]) carrying `VnfIdentifierCreationNotification` information element with attribute `vnfInstanceId` to indicate the VNF instance identifier creation (see clause 9.5.7 of GS NFV-IFA008 [4]).

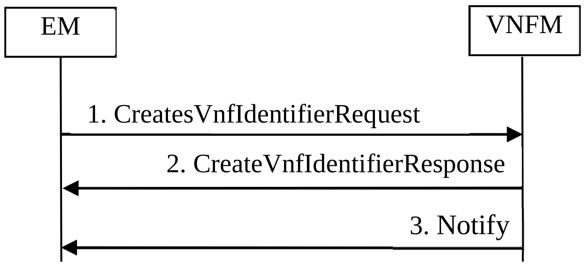
EM
VNFM
1. CreatesVnfIdentifierRequest
2. CreateVnfIdentifierResponse
3. Notify

Figure 4.2.1-1 Procedures of VNF identifier creation

4.2.2 VNF instantiation

4.2.2.1 VNF instantiation by EM request

Figure 4.2.2.1-1 depicts a procedure of VNF instantiation by EM request. It is assumed that EM has subscribed to receive the VNF lifecycle change notification from VNFM. As a result of this procedure, the new VNF instance is not associated to any NS (see NOTE in clause 7.2.3.4 in [4]).

1. EM sends *CreateVnfRequest* to VNFM with `vnfDescriptorId`, `vnfInstanceName`, and `vnfInstanceDescription` to create the VNF identifier (see clause 7.2.2 [4]).

2. VNFM sends *CreateVnfResponse* to EM with `vnfInstanceId` to indicate the creation of a new instance of a VNF information element (see clause 7.2.2.3 [4]).

3. EM sends *InstantiateVnfRequest* to VNFM with input parameters, listed in clause 7.2.3.2 [4] to instantiate a VNF (see clause 7.2.3 [4]).

4. VNFM sends *InstantiateVnfResponse* with `lifecycleOperationOccurrenceId` to EM (see clause 7.2.3.3 [4]).

5. VNFM send a *Notify* (see clause 7.5.3 [4]), carrying `VnfLifecycleChangeNotification` information element to EM with attributes `vnfInstanceId`, `status` = “start”, `operation` = “instantiation”, `lifeycleOperationOccurrenceId, affectedVnfc, affectedVl,` and `affectedVirtualStorage` to indicate the start of VNF instantiation (see clause 9.5.1 [4]).

6. VNFM send a *Notify* (see clause 7.5.3 [4]), carrying `VnfLifecycleChangeNotification` information element to EM with attributes `vnfInstanceId`, `status` = "result", `operation` = "instantiation", `lifeycleOperationOccurrenceId`, `affectedVnfc`, `affectedVl`, and `affectedVirtualStorage` to indicate the result of VNF instantiation, when the VNF instantiation operation is completed (see clause 9.5.1 [4]).

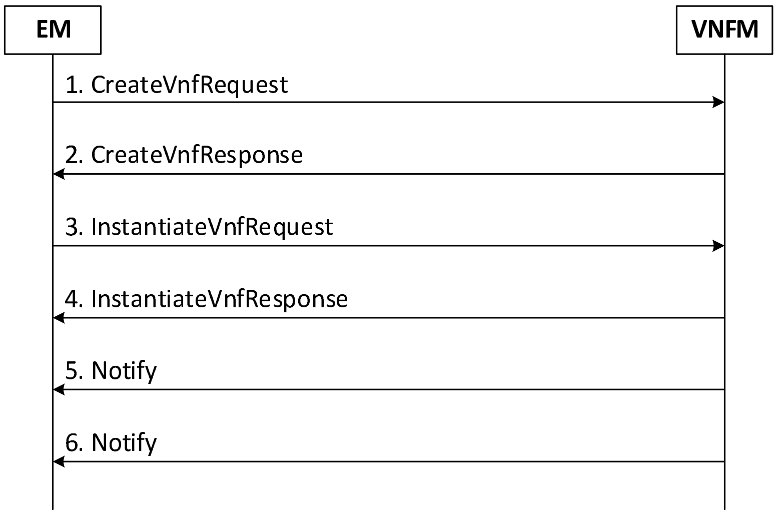

Figure 4.2.2.1-1: VNF instantiation procedure

4.2.2.2 VNF instantiation as part of NS update initiated through Os-Ma-nfvo

Figure 4.2.2.2-1 depicts the procedure of instantiating a VNF instance as part of an NS update through the Os-Ma-nfvo reference point (see clause 7.3.5 [5]).

1. NM sends to NFVO an *UpdateNsRequest* with parameters `nsInstanceId`, `updateType` = "InstantiateVnf", `instantiateVnfData`, and `updateTime` to instantiate the VNF instance (see clause 7.3.5.2 [5]). The `instantiateVnfData` contains the parameters that are needed for VNF instantiation, including `vnfdId`, `flavourId`, and can include in addition the parameters `instantiationLevelId`, `vnfInstanceName`, etc (see clause 8.3.4.12 [5]).

2. NFVO sends to NM an *UpdateNsResponse* with parameter `lifecycleOperationOccurrenceId` providing the identifier of the NS lifecycle operation occurrence (see clause 7.3.5.3 [5]).

3. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]), carrying `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "UpdateNs", and `notificationType` = "start" to indicate the start of the NS update that includes the VNF instantiation (see clause 8.3.2.2 [5]).

4. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]), carrying `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "UpdateNs", `notificationType` = "result" to indicate the end result of the NS update that includes the VNF instantiation, and `affectedVnf` providing information about the added VNF instance. The `affectedVnf` includes parameters `vnfInstanceId`, `vnfdId`, `vnfProfileId`, `vnfName` and `changeType` = "instantiated" (see clauses 8.3.2.2 and 8.3.2.3 [5]).

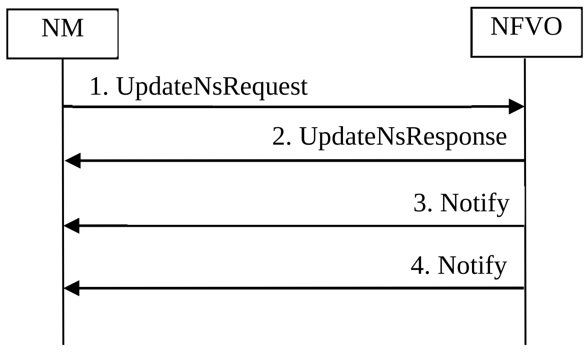

Figure 4.2.2.2-1: VNF instantiation as part of NS update initiated through Os-Ma-nfvo

4.2.2.3 Provide IP address of the managing EM in VNF instantiation

Figure 4.2.2.3-1 depicts a procedure of providing the IP address of the managing EM to the VNF in instantiation. As a result of this procedure, the new VNF instance is not associated to any NS (see NOTE in clause 7.2.3.4 in [4]). The figure uses UML notation to show multiple options available.

1. EM sends *CreateVnfIdentifierRequest* with parameters `vnfdId`, and optionally `vnfInstanceName` and `vnfInstanceDescription` to VNFM (see clause 7.2.2 [4]).

2. VNFM creates a new VnfInfo object

3. VNFM sends *CreateVnfIdentifierResponse* with the new VNF identifier to EM (see clause 7.2.2 [4]).

If the Multi-vendor Plug and Play connection to the network method is not used and managing EM IP address is provided as VNF configuration data, the steps 4.1.1 through 4.1.4 are executed.

4.1.1. EM sends *ModifyVnfConfigurationRequest* with parameters `vnfInstanceId`, `vnfConfigurationData`, `extVirtualLink` and `vnfcConfigurationData` to VNFM (see clause 7.6.2 [4]). The managing EM IP address value used in parameter `vnfConfigurationData`.

4.1.2. VNFM sets the `vnfConfigurationData` in the `vnfInfo` object.

4.1.3. VNFM sends *ModifyVnfConfigurationResponse* to EM (see clause 7.6.2 [4]).

4.1.4. EM sends *InstantiateVnfRequest* with parameters `vnfInstanceId`, `flavourId`, `instantiationLevelId`, `extVirtualLink`, `extManagedVirtualLink`, `localizationLanguage` and `additionalParam` to VNFM (see clause 7.2.3 [4]).
Note: The `extVirtualLink` may be known to the EM (e.g. provided by another entity).

If the Multi-vendor Plug and Play connection to the network method is not used and managing EM IP address is provided as additional parameter for instantiation, the steps 4.2.1 through 4.2.3 are executed.

4.2.1. EM sends *InstantiateVnfRequest* with parameters `vnfInstanceId`, `flavourId`, `instantiationLevelId`, `extVirtualLink`, `extManagedVirtualLink`, `localizationLanguage` and `additionalParam` to VNFM (see clause 7.2.3 [4]). The managing EM IP address value used in parameter `additionalParam`.

4.2.2. VNFM maps the managing EM IP address value received in parameter `additionalParam` of *InstantiateVnfRequest* to `vnfConfigurationData` (see Note 1).

NOTE 1: the specific mechanism for this mapping (e.g. vendor specific LCM script or specific VNFM) is out of scope of 3GPP

4.2.3. VNFM sets the `vnfConfigurationData` in the `vnfInfo` object.

If the Multi-vendor Plug and Play connection to the network method is used to provide the managing EM IP address to VNF, step 4.3.1 is executed.

4.3.1. EM sends *InstantiateVnfRequest* with parameters `vnfInstanceId`, `flavourId`, `instantiationLevelId`, `extVirtualLink`, `extManagedVirtualLink`, `localizationLanguage` and `additionalParam` to VNFM (see clause 7.2.3 [4]).

5. VNFM initiates the VNF instantiation process.

6. VNFM sends *InstantiateVnfResponse* with the new `lifecycleOperationOccurrenceId` to EM (see clause 7.2.3 [4]).

7. VNFM sends *SetInitialConfigurationRequest* with parameters `vnfInstanceId`, `vnfConfigurationData` and `vnfcConfigurationData` to VNF (see clause 6.2.2 [4]).

8. VNF sends *SetInitialConfigurationResponse* with parameters `vnfConfigurationData` and `vnfcConfigurationData` to VNFM (see clause 6.2.2 [4]).

9. If Multi-vendor Plug and Play connection to the network method is used to provide the managing EM IP address to VNF, VNF performs the EM discovery (see "Establishing connection to Element Manager" procedure in clause 5.5 of TS 32.508 [6]).

10. VNF connects to the managing EM.

11. "Normal" NE management by the EM over Type-1 interface (e.g. s/w update, configuration) begins.

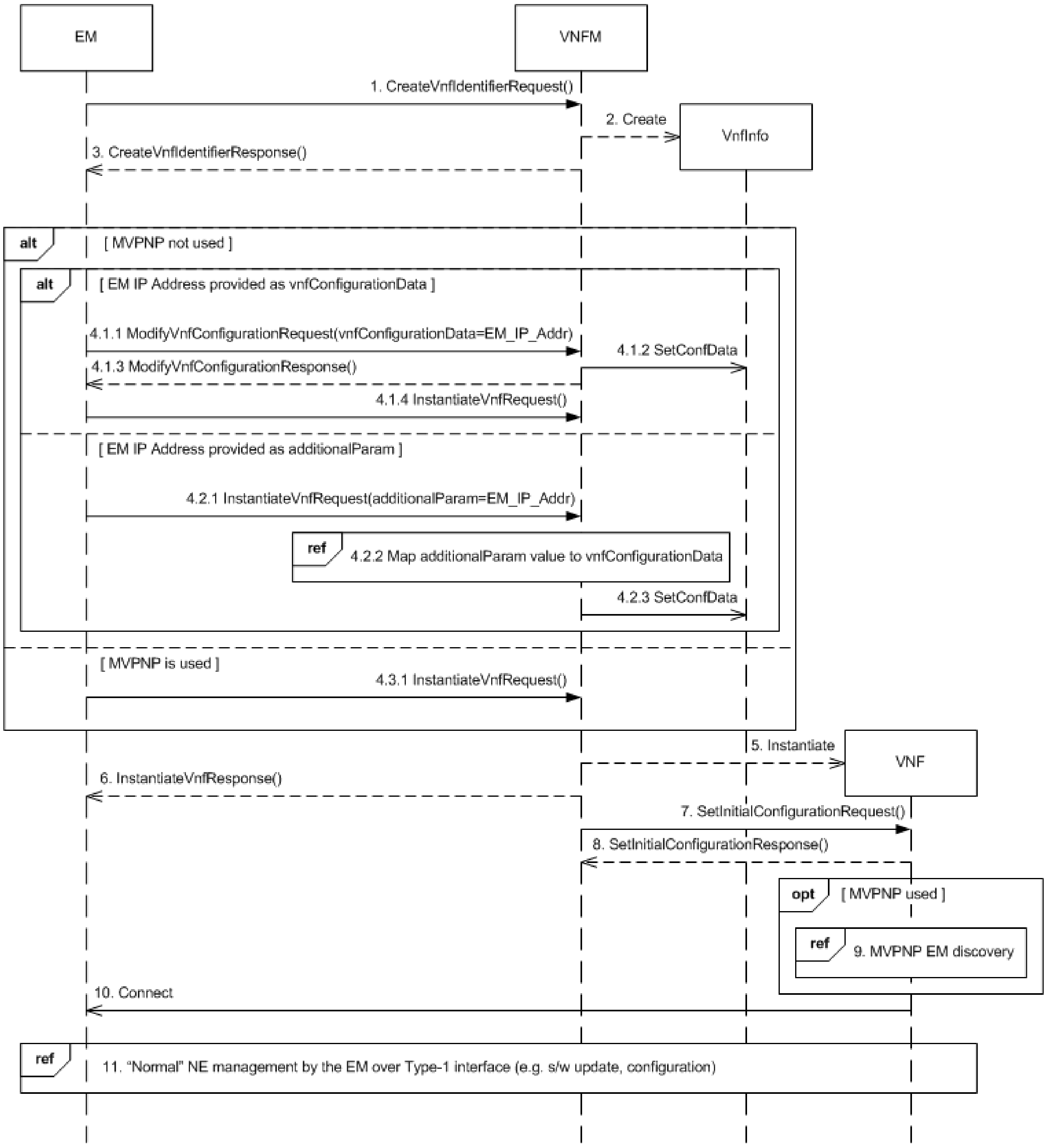
EM
VNFM
1. CreateVnfIdentifierRequest()
2. Create
VnfInfo
3. CreateVnfIdentifierResponse()
alt
[MVPNP not used]
alt
[EM IP Address provided as vnfConfigurationData]
4.1.1 ModifyVnfConfigurationRequest(vnfConfigurationData=EM_IP_Addr)
4.1.2 SetConfData
4.1.3 ModifyVnfConfigurationResponse()
4.1.4 InstantiateVnfRequest()
[EM IP Address provided as additionalParam]
4.2.1 InstantiateVnfRequest(additionalParam=EM_IP_Addr)
ref
4.2.2 Map additionalParam value to vnfConfigurationData
4.2.3 SetConfData
[MVPNP is used]
4.3.1 InstantiateVnfRequest()
5. Instantiate
VNF
6. InstantiateVnfResponse()
7. SetInitialConfigurationRequest()
8. SetInitialConfigurationResponse()
opt
[MVPNP used]
ref
9. MVPNP EM discovery
10. Connect
ref
11. "Normal" NE management by the EM over Type-1 interface (e.g. s/w update, configuration)

Figure 4.2.2.3-1: Provide IP address of the managing EM in VNF instantiation procedure

4.2.2.4 Query VNF instance information through Ve-Vnfm-em

Figure 4.2.2.4-1 depicts the procedure of querying VNF instance information through the Ve-Vnfm-em reference point (see clause 7.2.9 [4]).

1. EM sends to VNFM a *QueryVnfRequest* with parameters `filter` and `attributeSelector` used to filter the VNF instance(s) and select the information attributes that are requested to be returned (see clause 7.2.9.2 [4]).

2. VNFM sends to EM a *QueryVnfResponse* with parameter `vnfInfo` providing the information that is selected according to parameters `filter` and `attributeSelector` (see clause 7.2.9.3 [4]).

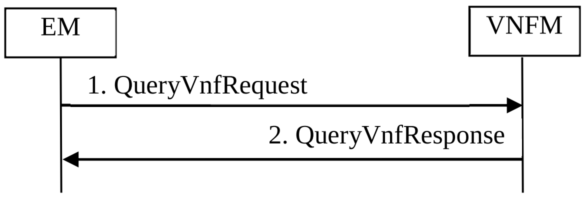

Figure 4.2.2.4-1: Query VNF instance information through Ve-Vnfm-em

4.2.3 VNF scaling

4.2.3.1 Scale VNF instance initiated by EM

Figure 4.2.3.1-1 depicts a procedure of scaling VNF instance (see clause 7.2.8 [4]).

1. EM sends *ScaleVnfRequest* with parameters `vnfInstanceId`, `type`, `aspectId`, and `numberOfSteps` to scale the VNF instance (see clause 7.2.8 [4]).

2. VNFM sends *ScaleVnfResponse*with the identifier of the VNF lifecycle operation occurrence `lifecycleOperationOccurrenceId` to EM (see clause 7.2.8.

3. VNFM send a *Notify* (see clause 7.5.3 [4]), carrying `VnfLifecycleChangeNotification` information element, to EM with attributes `vnfInstanceId`, `status` = "start", `operation` to indicate the start of VNF scaling (see clause 9.5.1 [4]).

4. VNFM send a *Notify* (see clause 7.5.3 [4]), carrying `VnfLifecycleChangeNotification` information element, to EM with attributes `vnfInstanceId`, `status` = "result", `operation` to indicate the result of VNF scaling, when the VNF scaling operation is completed (see clause 9.5.1 [4]).

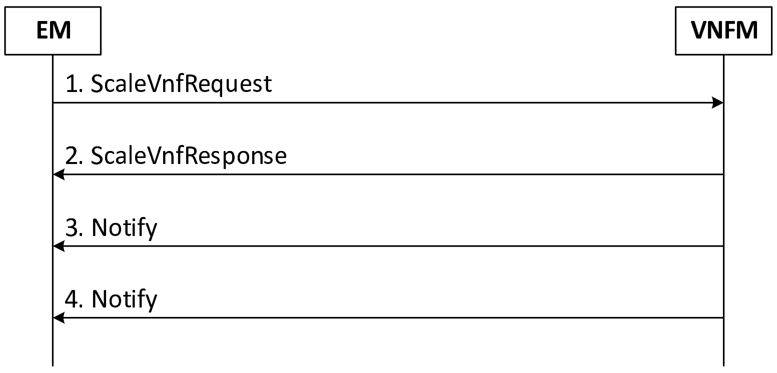

Figure 4.2.3.1-1: Scale VNF instance procedure

4.2.3.2. Scale VNF instance to a level initiated by EM

Figure 4.2.3.2-1 depicts a procedure of scaling VNF instance to a level (see clause 7.2.9 [4]).

1. EM sends *ScaleVnfToLevelRequest* with parameters `vnfInstanceId`, and (`instantiationLevelId` or `scaleInfo`) to scale the VNF instance to a level defined by `instantiationLevelId` or `scaleInfo` (see clause 7.2.9 [4]).

2. VNFM sends *ScaleVnfToLevelResponse* with the identifier of the VNF lifecycle operation occurrence `lifecycleOperationOccurrenceId` to EM (see clause 7.2.9 [4]).

3. VNFM send a *Notify* (see clause 7.5.3 [4]), carrying `VnfLifecycleChangeNotification` information element, to EM with attributes `vnfInstanceId`, `status` = "start", `operation` to indicate the start of VNF scaling (see clause 9.5.1 [4]).

4. VNFM send a *Notify* (see clause 7.5.3 [4]), carrying `VnfLifecycleChangeNotification` information element, to EM with attributes `vnfInstanceId`, `status` = "result", `operation` to indicate the result of VNF scaling, when the VNF scaling operation is completed (see clause 9.5.1 [4]).

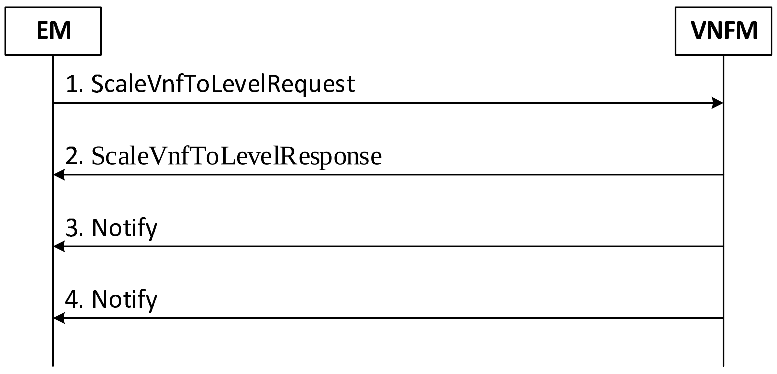

Figure 4.2.3.2-1: Scale VNF instance to a level procedure

4.2.3.3 VNF scaling as part of NS scaling initiated through Os-Ma-nfvo

Figure 4.2.3.3-1 depicts the procedure of scaling VNF instance for which the NM has selected the NS instance within which the subject VNF instance is intended to be scaled (see clause 7.3.4 [5]).

1. NM sends to NFVO a *ScaleNsRequest* with parameters `nsInstanceId`, `scaleType =` "SCALE_VNF", `scaleVnfData`, and `scaleTime` to scale the VNF instance(s) (see clause 7.3.5.2 [5]). The `scaleVnfData` contains the parameters that are needed for scaling a specific VNF instance, including among others `vnfInstanceId`, `type`, and either `scaleToLevelData` or `scaleByStepData` depending on the type of scaling (see clauses 8.3.4.9.2, 8.3.4.10 and 8.3.4.11 [5]).

2. NFVO sends to NM a *ScaleNsResponse* with parameter `lifecycleOperationOccurrenceId` providing the identifier of the NS lifecycle operation occurrence (see clause 7.3.5.3 [5]).

3. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "ScaleNs", and `notificationType` = "start" to indicate the start of the NS scaling that is being performed through specific VNF scaling (see clause 8.3.2.2 [5]).

4. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "ScaleNs", `notificationType` = "result" to indicate the end result of the NS scaling performed through specific VNF scaling, and `affectedVnf` providing information about the scaled VNF instance(s), including `vnfInstanceId`, `vnfdId`, `vnfProfileId`, `vnfName` and `changeType` = "scaled" (see clauses 8.3.2.2 and 8.3.2.3 [5]).

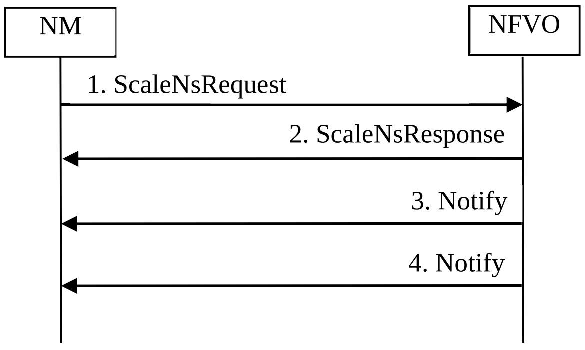

Figure 4.2.3.3-1: VNF scaling as part of NS scaling initiated through Os-Ma-nfvo

4.2.4 VNF instance termination

4.2.4.1 VNF termination by EM request

Figure 4.2.4.1-1 depicts a procedure of VNF termination by EM request, when this VNF instance is not needed. The VNF instance has been instantiated (see NOTE in clause 7.2.3.4 in [8]) by the EM. It is assumed that EM has subscribed to receive the VNF lifecycle change notification from VNFM. The VNF instance identifier will be deleted after the VNF termination.

1. EM sends *TerminateVnfRequest* to VNFM with `vnfInstanceId` to terminate the VNF instance (see clause 7.2.7 [4]).
2. VNFM sends *TerminateVnfResponse* with `lifecycleOperationOccurrenceId` to EM (see clause 7.2.7.3 [4]).
3. VNFM sends a *Notify* (see clause 7.5.3 [4]), carrying `VnfLifecycleChangeNotification` to EM with attributes `vnfInstanceId`, `status` = "start", `operation` = "termination" to indicate the start of VNF termination (see clause 9.5.1 [4]).
4. VNFM sends a *Notify* (see clause 7.5.3 [4]), carrying `VnfLifecycleChangeNotification` to EM with attributes `vnfInstanceId`, `affectedVnfc`, `affectedVirtualLink`, `affectedVirtualStorage`, `status` = "result", `operation` = "termination" to indicate the result of VNF termination, when the VNF termination operation is completed (see clause 9.5.1 [4]).
5. EM sends *DeleteVnfIdentifierRequest* to VNFM with `vnfInstanceId` to delete the VNF instance identifier (see clause 7.2.8 [4]).
6. VNFM sends a *Notify* (see clause 7.5.3 [4]), carrying `VnfIdentifierDeletionNotification` information element to EM with attributes `vnfInstanceId` (see clause 9.5.8 [4]).

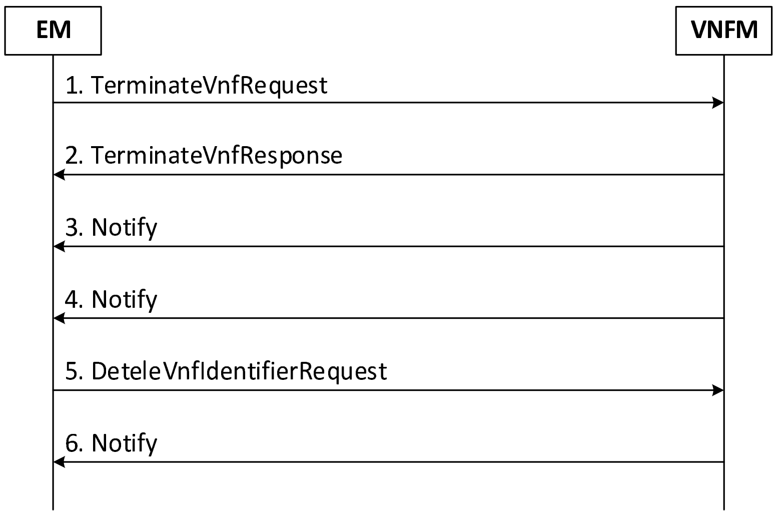

Figure 4.2.4.1-1: VNF termination procedure

4.2.4.2 VNF termination by removing VNF instance from NS

Figure 4.2.4.2-1 depicts the procedure of VNF termination by removing VNF instance from NS through Os-Ma-Nfvo reference point (see clause 7.3.5 of GS NFV-IFA013 [5])

1. NM sends to NFVO an *UpdateNsRequest* with parameter `nsInstanceId`, `updateType =` "RemoveVnf", `RemoveVnfInstanceId` and `updateTime` to remove the VNF instance(s) from NS (see clause 7.2.2.2 of GS NFV-IFA013 [5]). The `RemoveVnfInstanceId` identifies the VNF instance to be removed from the NS.

2. NFVO remove the target VNF instance and sends to NM an *UpdateNsResponse* with parameter `vnfInstanceId` and `lifecycleOperationOccurrenceId` providing the identifier of the NS lifecycle operation occurrence (see clause 7.3.5.3 of GS NFV-IFA013 [5]).

3. NFVO sends to NM a *Notify* (see clause 7.4.3 of GS NFV-IFA013 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "UpdateNs" and `notificationType` = "start" to indicate the start of the NS updating that is being performed through removing specific VNF instance(s) (see clause 8.3.2.2 of GS NFV-IFA013[5]).

4. If the NFVO decides to terminate the NS instance, it sends to NM a *Notify* (see clause 7.4.3 of GS NFV-IFA013 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "UpdateNs" and `notificationType` = "result" to indicate the end result of the NS updating performed through removing specific VNF(s) removal, and `affectedVnf` providing information about the removed VNF instance(s), including `vnfInstanceId`, `vnfdId`, `vnfProfileId`, `vnfName` and `changeType` = "terminated" (see clause 8.3.2.2 and 8.3.2.3 of GS NFV-IFA013 [5]).

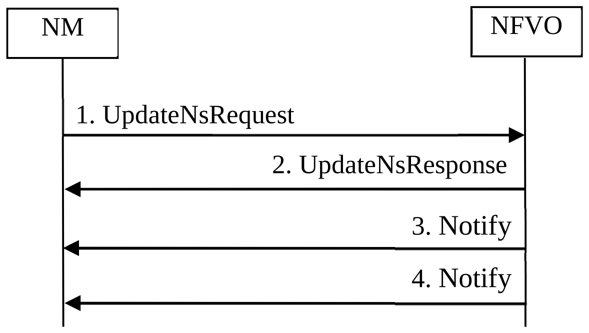

Figure 4.2.4.2-1 Procedures of VNF termination by removing VNF from NS instance

4.2.5 Notifications about VNF lifecycle changes

Figure 4.2.5-1 depicts the procedure of notification on VNF instance lifecycle change notifications through the Ve-Vnfm-em reference point (see clause 7.3.3 [4]).

1. VNFM sends to EM a *Notify* according to clause 7.3.3 in [4]. This operation sends to EM the notifications supported on the VNF lifecycle change notification interface.

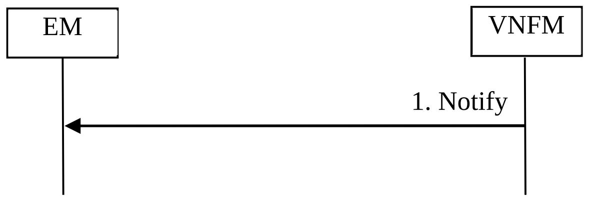

Figure 4.2.5-1: Notifications about VNF lifecycle changes

The following notifications can be notified/sent by this operation:

- VnfLifecycleChangeNotification. See clause 9.5.2 [4] with the list of attributes in the Table 9.5.2.3-1 [4].
- VnfInfoAttributeValueChangeNotification. See clause 9.5.6 [4] with the list of attributes in the Table 9.5.6.3-1 [4].
- VnfIdentifierCreationNotification. See clause 9.5.7 [4] with the list of attributes in the Table 9.5.7.3-1 [4].
- VnfIdentifierDeletionNotification. See clause 9.5.8 [4] with the list of attributes in the Table 9.5.8.3-1 [4].

4.2.6 Autoscaling enabling/disabling

4.2.6.1 Autoscaling enabling/disabling initiated through Ve-Vnfm-em

Figure 4.2.6.1-1 depicts the procedure of enabling/disabling the autoscaling of a VNF instance through the Ve-Vnfm-em reference point.

1. EM sends to VNFM a *ModifyVnfConfigurationRequest* (see clause 7.6.2 [4]) with input parameter `vnfInstanceId`, `vnfConfigurationData.autoScalable = TRUE/FALSE` (see clause 7.1.12 [8]) for enabling/disabling the autoscaling.

2. VNFM sends to EM a *ModifyVnfConfigurationResponse*(see clause 7.6.2 [4]).

3. VNFM send a *Notify* (see clause 7.6.4 [4]), carrying `VnfConfigAttributeValueChangeNotification` information element, to EM with attributes `vnfInstanceId`, `changedConfig` to inform about the changed VNF configuration. (see clause 9.2.7 [4]).

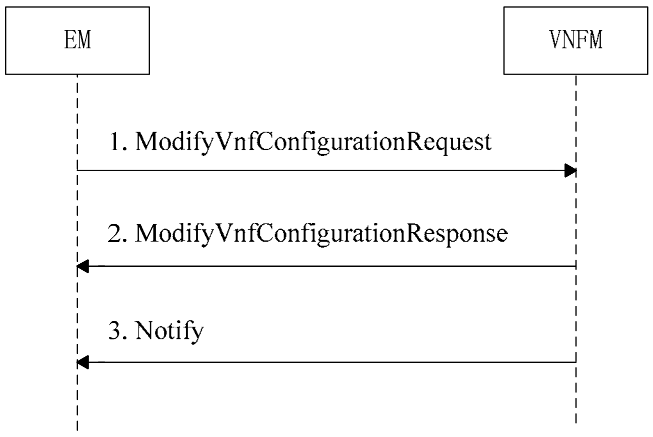
EM
VNFM
1. ModifyVnfConfigurationRequest
2. ModifyVnfConfigurationResponse
3. Notify

Figure 4.2.6.1-1: Autoscaling enabling/disabling initiated through Ve-Vnfm-em

4.2.6.2 Autoscaling enabling/disabling through Os-Ma-nfvo

For enabling/disabling the autoscaling initiated through Os-Ma-nfvo, the procedure in clause 4.4.5.3 of the present document about "Modifying VNF instance configuration through Os-Ma-nfvo" applies. The `vnfConfigurationData` contained in the `modifyVnfConfigData` parameter that is carried in the *UpdateNsRequest* of step 1 of the referred procedure provides the information for enabling/disabling the autoscaling of the corresponding VNF instance (refer to the autoScalable attribute of VnfConfigurableProperties in clause 7.1.12 in [8]).

4.2.7 Subscribing to VNF lifecycle change notifications through Ve-Vnfm-em

Figure 4.2.7-1 depicts the procedure of subscribing to VNF lifecycle change notifications through the Ve-Vnfm-em reference point (see clause 7.3.2 [4]).

1. EM sends to VNFM a *SubscribeRequest* with input parameter `filter` for selecting the notifications, which can be on the VNF instance(s) of interest or other attributes of the notification (see clause 7.3.2.2 [4]).

2. VNFM sends to EM a *SubscribeResponse* with parameter `subscriptionId` providing the identifier of the subscription realized (see clause 7.3.2.3 [4]).

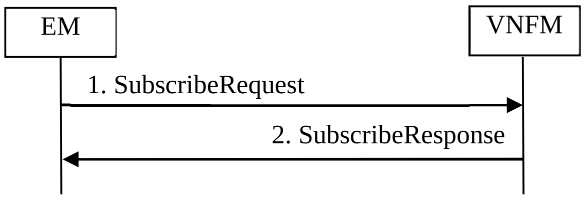

Figure 4.2.7-1: Subscribing to VNF lifecycle change notifications through Ve-Vnfm-em

4.2.8 3GPP network function software update when application software is part of VNF Package and synchronization of VNF instance information is performed through Os-Ma-nfvo

Figure 4.2.8-1 depicts the procedure of updating the application software of the virtualized 3GPP network function when the software is part of the VNF Package and the software update does not require a change of the VNF's underlying virtualised resources or internal VNFC(s) topology/composition.

1. NM requests to EM to update the software, EM acts to update the software, and notifies to NM about the completion of the software update using the Software Management IRP [7].
2. Once the NM confirms the normality of service of the 3GPP network function whose software has been updated, NM requests to NFVO to update the VNF instance information (see procedure in clause 4.4.5.2) capturing the information of the new VNF Package for the 3GPP network function whose software has been updated.

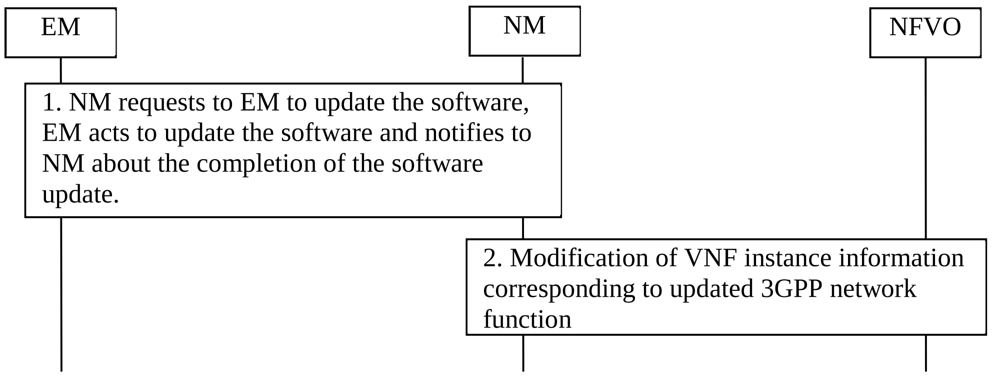
EM
NM
NFVO
1. NM requests to EM to update the software, EM acts to update the software and notifies to NM about the completion of the software update.
2. Modification of VNF instance information corresponding to updated 3GPP network function

Figure 4.2.8-1: 3GPP network function application software update when application software is part of VNF Package and synchronization of VNF instance information is performed through Os-Ma-nfvo

4.3 VNF Package procedures

4.3.1 VNF package on-boarding

Figure 4.4-1 depicts a procedure of VNF package on-boarding,

1. NM sends *OnboardVnfPackageRequest* to NFVO with input parameters listed in clause 7.7.2.2 [5] to on-board a VNF package (see clause 7.7.2 [5]).
2. NFVO sends *OnboardVnfPackageResponse* to NM with with `vnfPackageId` to indicate a VNF package has been on-boarded (see clause 7.7.2.3 [5]).

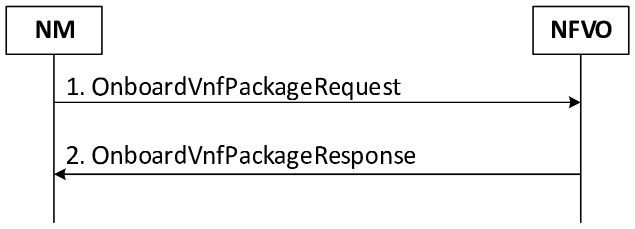

Figure 4.4-1: VNF package on-boarding procedure

4.3.2 VNF Package enabling

Figure 4.3.2-1 depicts the procedure of VNF package enabling.

1. NM sends *EnableVnfPackageRequest* to NFVO with parameter `onboardedVnfPkgInfoId` to enable a VNF package (see clause 7.7.3.2 of GS NFV-IFA013 [5]).

2. NFVO sends *EnableVnfPackageResponse* to NM (see clause 7.7.3.3 of GS NFV-IFA013 [5]).

3. NFVO sends to NFVO a *Notify* (see clause 7.7.8 of GS NFV-IFA013 [5]) carrying `VnfPackageChangeNotification`, information element with attribute `onboardedVnfPkgInfoId`, `vnfdId` and `operationalState` = "Enabled" to indicate the VNF package enabling (see clause 8.6.9 of GS NFV-IFA013 [5])

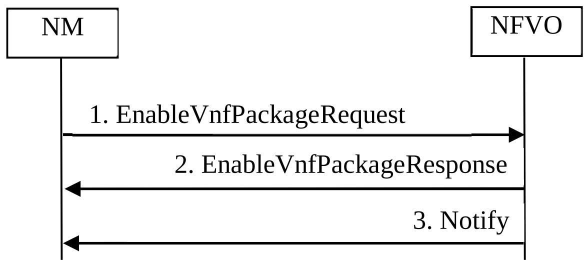

Figure 4.3.2-1 Procedures of VNF package enabling

4.3.3 VNF Package disabling

Figure 4.3.3-1 depicts the procedure of VNF Package disabling.

1. NM sends *DisableVnfPackageRequest* to NFVO with `onboardedVnfPkgInfoId` to disable a previously enabled VNF Package instance, preventing further use for instantiation of new VNF instances with this package (unless and until the VNF Package is re-enabled) (see clause 7.7.4 [5]).

2. NFVO sends *DisableVnfPackageResponse* to NM to indicate the result of the operation (see clause 7.7.4.4 [5]).

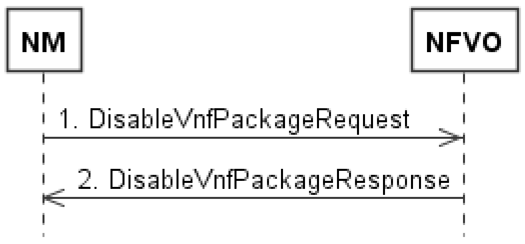

Figure 4.3.3-1: VNF Package disabling procedure

4.3.4 VNF Package deleting

Figure 4.3.4-1 depicts the procedure of VNF Package deleting.

1. NM sends *DeleteVnfPackageRequest* to NFVO with `onboardedVnfPkgInfoId` to delete a VNF Package (see clause 7.7.5 [5]).

2. NFVO sends *DeleteVnfPackageResponse* to NM to indicate the result of the operation (see clause 7.7.5.4 [5]).

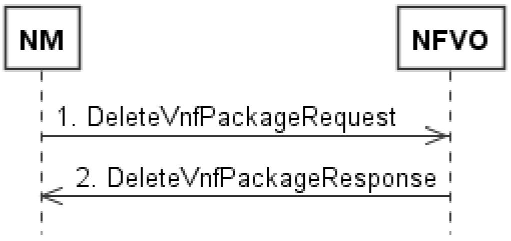

Figure 4.3.4-1: VNF Package deleting procedure

4.3.5 Abort VNF package deletion

Figure 4.3.5-1 depicts the procedure to abort deletion of a VNF Package that is in deletion pending state.

1. NM sends *AbortPackageDeletionRequest* to NFVO with `onboardedVnfPkgInfoId` to delete a VNF Package (see clause 7.7.12 [5]).

2. NFVO sends *AbortPackageDeletionResponse* to NM to indicate the result of the operation (see clause 7.7.12.4 [5]).

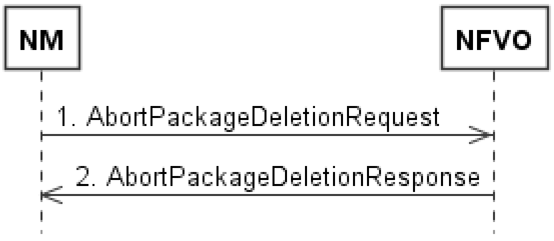
NM
NFVO
1. AbortPackageDeletionRequest
2. AbortPackageDeletionResponse

Figure 4.3.5-1: Abort VNF Package deletion procedure

4.3.6 VNF Package querying

Figure 4.3.6-1 depicts a procedure to query from the NFVO for information it has stored about one or more VNF Packages.

1. NM sends *QueryOnboardedVnfPkgInfoRequest* to NFVO with `filter` and `attributeSelector` used to filter the VNF Packages on which the query applies, based on the attributes of `OnboardedVnfPkgInfo` and select the information attributes of `OnboardedVnfPkgInfo` that are requested to be returned (see clause 7.7.6 [5]).

2. NFVO sends *QueryOnboardedVnfPkgInfoResponse* to NM with parameter `queryResult` providing the information that is selected according to parameters `filter` and `attributeSelector` (see clause 7.7.6.4 [5]).

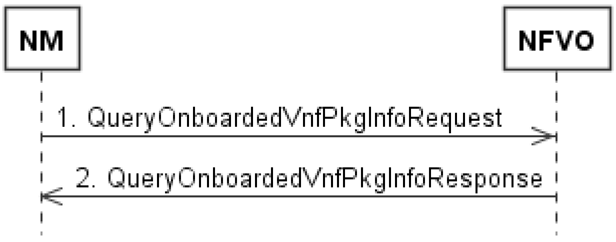
NM
NFVO
1. QueryOnboardedVnfPkgInfoRequest
2. QueryOnboardedVnfPkgInfoResponse

Figure 4.3.6-1: VNF Package querying procedure

4.3.7 Fetch VNF Package

Figure 4.3.7-1 depicts a procedure to fetch from the NFVO a whole VNF Package based on the VNFD identifier that has been assigned by the VNF Provider.

NOTE 1: In cases where NM already knows the value of `onboardedVnfPkgInfoId` for the VNF Package it wishes to fetch, it may skip the steps 1 and 2 below (execution of procedure begins at step 3).

1. NM sends *QueryOnboardedVnfPkgInfoRequest* to NFVO with `filter` parameter set to VNFD identifier that has been assigned by the VNF Provider and `attributeSelector` set to `OnboardedVnfPkgId` (see clause 7.7.6 [5]).

2. NFVO sends *QueryOnboardedVnfPkgInfoResponse* to NM with parameter `queryResult` providing the `OnboardedVnfPkgId` identifier allocated by NFVO to the corresponding VNF package (see clause 7.7.6.4 [5]).

3. NM sends *FetchOnboardedVnfPackageRequest* to NFVO with `onboardedVnfPkgInfoId` parameter identifying the VNF Package to fetch (see clause 7.7.10 [5]).

4. NFVO sends *FetchOnboardedVnfPackageResponse* to NM with the `vnfPackage` requested (see clause 7.7.10.4 [5]).

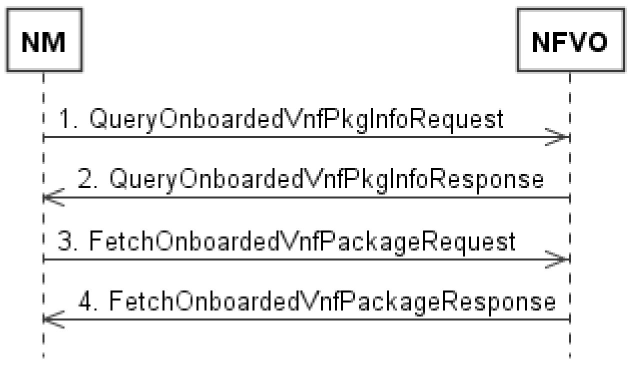
NM
NFVO
1. QueryOnboardedVnfPkgInfoRequest
2. QueryOnboardedVnfPkgInfoResponse
3. FetchOnboardedVnfPackageRequest
4. FetchOnboardedVnfPackageResponse

Figure 4.3.7-1: Fetch VNF Package procedure

4.3.8 Notify operation on VNF Package management interface

Figure 4.3.8-1 depicts the procedure of notification on VNF package management interface.

1. NFVO sends Notify message to NM per 7.7.8 [5]. This operation sends to NM the notifications supported on the VNF Package management interface. To receive notifications, the NM performs an explicit Subscribe operation beforehand (see clause 7.7.11 [5]).

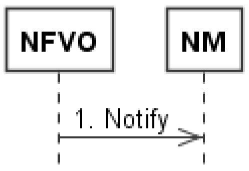

Figure 4.3.8-1: Notify operation on VNF Package management interface procedure

The following notification can be notified/sent by this operation:

- VnfPackageOnBoardingNotification. See clause 8.6.8 [5] with the list of attributes in the Table 8.6.8.3-1 [5].

- VnfPackageChangeNotification. See clause 8.6.9 [5] with the list of attributes in the Table 8.6.9.3-1 [5].

4.3.9 Subscribe operation on VNF Package management interface

Figure 4.3.9-1 depicts a procedure to subscribe for notifications on VNF package management interface.

1. NM sends *SubscribeRequest* to NFVO with `inputFilter` selecting the VNF Package(s) and the related change notifications to subscribe to. This filter can contain information about specific types of changes to subscribe to, or attributes of the VNF Package (see clause 7.7.7 [5]).

2. NFVO sends *SubscribeResponse* to NM with the list of `subscriptionId` identifier of the subscription realized (see clause 7.7.7.4 [5]).

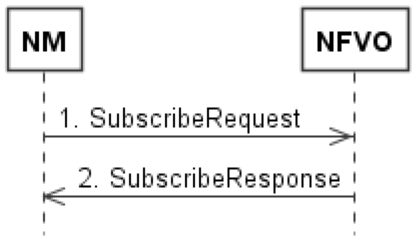

Figure 4.3.9-1: Subscribe operation on VNF Package management interface procedure

4.3.10 Fetch on-boarded VNF Package artifacts

Figure 4.3.10-1 depicts a procedure to fetch from the NFVO selected artifacts contained in an on-boarded VNF package.

1. NM sends *FetchOnboardedVnfPackageArtifactsRequest* to NFVO with `onboardedVnfPkgInfoId` parameter identifying specific on-boarded VNF Package and `artifactAccessInformation` parameter listing individual package artifacts to fetch (see clause 7.7.11 [5]).

2. NFVO sends *FetchOnboardedVnfPackageArtifactsResponse* to NM with the list of `vnfPackageArtifact` (e.g. files) requested (see clause 7.7.11.4 [5]).

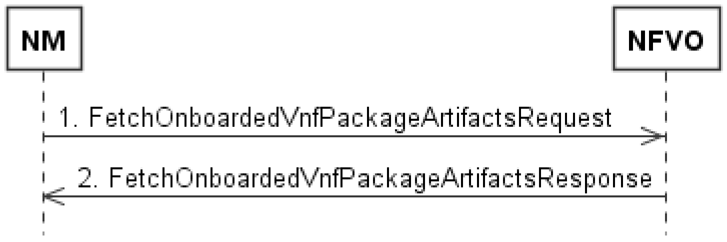
NM
NFVO
1. FetchOnboardedVnfPackageArtifactsRequest
2. FetchOnboardedVnfPackageArtifactsResponse

Figure 4.3.10-1: Fetch on-boarded VNF Package artifacts procedure

4.4 NS Instance procedures

4.4.1 NS Instance instantiation

Figure 4.4.1-1 depicts the procedure of an NS instantiation initiated through the Os-Ma-nfvo reference point (see clause 7.3.3 [5]). The procedure includes the steps of creating first the corresponding NS instance identifier.

1. NM sends to NFVO a *CreateNsIdentifierRequest* with parameters `nsdId`, `nsName`, and `nsDescription` to create an NS instance identifier (`nsInstanceId`) and an associated instance of an `NsInfo` information element (see clause 7.3.2.2 [5]).

2. NFVO sends to NM a *CreateNsIdentifierResponse* with parameter `nsInstanceId` identifying the instance of the NS that has been created (see clause 7.3.2.3 [5]).

3. NFVO sends to subscribers a *Notify* (see clause 7.4.3 [5]) carrying `NsIdentifierCreationNotification` information element with attribute `nsInstanceId` to indicate the NS instance idenfier creation (see clause 8.3.2.9 [5]).

4. NM sends to NFVO an *InstantiateNsRequest* with parameters `nsInstanceId` and `flavourId`. Additional parameters can be provided including `sapData`, `pnfInfo`, `locationConstraints`, `additionalParamsForNs`, `additionalParamForVnf`, `startTime`,

`nsInstantiationLevelId`, and `additionalAffinityOrAntiAffinityRule`. In addition, if the NS instantiation includes reusing existing VNF instances and/or NS instances, parameters `vnfInstanceData` and `nestedNsInstanceId` are provided, respectively. See clause 7.3.3.2 [5].

5. NFVO sends to NM an *InstantiateNsResponse* with parameter `lifecycleOperationOccurrenceId` providing the identifier of the NS lifecycle operation occurrence (see clause 7.3.3.3 [5]).

6. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "NsInstantiation", and `notificationType` = "start" to indicate the start of the NS instantiation (see clause 8.3.2.2 [5]).

7. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "NsInstantiation", and `notificationType` = "result" to indicate the end result of the NS instantiation. According to the results of the NS instantiation, additional information is provided in the notification with parameters `affectedVnf`, `affectedPnf`, `affectedVl`, `affectedVnffg`, `affectedNs` and `affectedSap` (see clause 8.3.2.2 [5]).

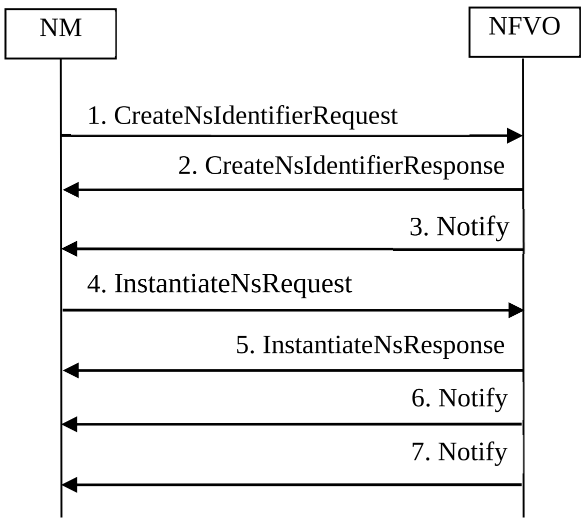

Figure 4.4.1-1: NS instantiation

4.4.2 NS Instance termination

Figure 4.4.2-1 depicts the procedure of an NS instance termination initiated through the Os-Ma-Nfvo reference point (see clause 7.3.7 [5]).

1. NM sends to NFVO a *TerminateNsRequest* with parameter `nsInstanceId` and `terminateTime` (see clause 7.3.7 [5]).

2. NFVO sends to NM a *TerminateNsResponse* with parameter `lifecycleOperationOccurrenceId` providing the identifier of the NS lifecycle operation occurrence (see clause 7.3.7.3 [5]).

3. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carriying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId, lifecycleOperationoccurrenceId`, `operation` = "TerminationNs", and "`notificationType`" = "start' to indicate the start of the NS termination (see clause 8.3.2.2 [5]).

4. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`,

`operation` = "TerminationNs", and `notificationType` = "result" to indicate the end result of the NS termination (see clause 8.3.2.2 [5]).

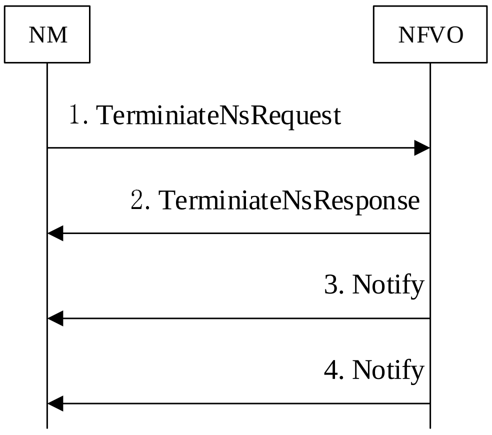

Figure 4.4.2-1: NS instance termination

4.4.3 NS Instance querying

4.4.3.1 Query NS instance information through Os-Ma-nfvo

Figure 4.4.3.1-1 depicts the procedure of NS querying through the Os-Ma-nfvo reference point (see clause 7.3.6 in [5]) used also to query information about the constituents of the NS instance, including VNF instances.

1. NM sends to NFVO a *QueryNsRequest* with parameters `filter` and `attributeSelector` used to filter the NS instances on which the query applies, based on attributes of the NS instance and select the information attributes that are requested to be returned (see clause 7.3.6.2 [5]).

   In particular, the `filter` can be set to select certain VNF instances(s) that are part of an NS, by providing their identifiers with the `attributeSelector` set to select certain attributes of the VNF instances.

2. NFVO sends to NM a *QueryNsResponse* with parameter `queryNsResult` providing the information that is selected according to parameters `filter` and `attributeSelector` (see clause 7.3.6.3 [5]).

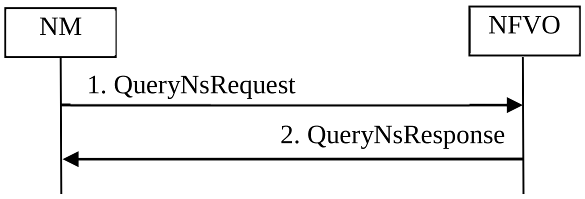

Figure 4.4.3.1-1: Query NS instance information through Os-Ma-nfvo

4.4.4 NS Instance scaling

Figure 4.4.4-1 depicts a procedure for NS Instance scaling. Scaling an NS instance can be performed by explicitly adding/removing existing VNF instances to/from the NS instance, by leveraging on the abstraction mechanism provided by the NS scaling aspects and NS levels information elements declared in the NSD or by scaling individual VNF instances (as described by the Use Case 6.4.3.4 in [3]) that are part of the NS itself. When adding VNFs and nested NSs - already existing or not - to the NS to be scaled, the NFVO shall follow the indications provided by the dependencies attribute, as specified in the corresponding NSD.

1. NM sends *ScaleNsRequest* to NFVO with `nsInstanceId` parameter identifying the NS Instance being scaled, `scaleType` parameter with possible values `[SCALE_NS, SCALE_VNF]`indicating the type of scaling to be performed, list of `scaleNsData` providing necessary scale information when `scaleType = SCALE_NS` or list of `scaleVnfData` providing necessary scale information when `scaleType = SCALE_VNF` and `scaleTime` indicating the scale time of the NS if it's not expected to start immediately (see clause 7.3.4 [5]).

2. NFVO sends *ScaleNsResponse* to NM with the `lifecycleOperationOccurrenceId` identifier of the NS lifecycle operation occurrence (see clause 7.3.4.4 [5]).

3. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "ScaleNs", and `notificationType` = "start" to indicate the start of the NS scale (see clause 8.3.2.2 [5]).

4. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "ScaleNs", `notificationType` = "result" to indicate the result of NS scale. Per the changes in the NS instance performed because of the NS scale, additional information is provided in the notification with parameters `affectedVnf`, `affectedPnf`, `affectedVl`, `affectedVnffg`, `affectedNs` and `affectedSap` (see clause 8.3.2.2 [5]).

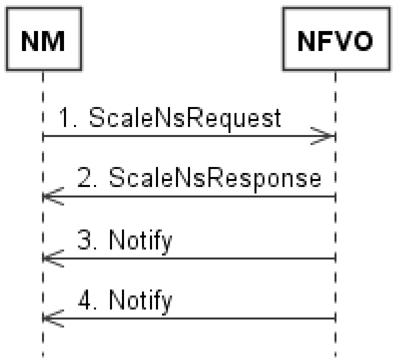

Figure 4.4.4-1: NS Instance scaling procedure

4.4.5 NS Instance updating

4.4.5.1 NS update to associate an NS instance to a different NSD version

Figure 4.4.5.1-1 depicts the procedure of NS update to associate an NS instance to a different NSD version.

1. NM sends to NFVO an *UpdateNsRequest* with parameters `nsInstanceId`, `updateType =` "AssociateNewNsdVersion", `assocNewNsdVersionData`, and `updateTime` to associate a new NSD version to the NS instance (see clause 7.3.5.2 [5]). The `assocNewNsdVersionData` contains the parameter `newNsdId`, and can include in addition the parameter `sync` to indicate whether the NS instance should be automatically synchronized to the new NSD by the NFVO.

2. NFVO sends to NM an *UpdateNsResponse* with parameter `lifecycleOperationOccurrenceId` providing the identifier of the NS lifecycle operation occurrence (see clause 7.3.5.3 [5]).

3. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "UpdateNs", and `notificationType` = "start" to indicate the start of the NS update (see clause 8.3.2.2 [5]).

4. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "UpdateNs", `notificationType` = "result" to indicate the end result of NS update. According to the changes in the NS instance performed as a result of the NS update, additional information is

provided in the notification with parameters `affectedVnf`, `affectedPnf`, `affectedVl`, `affectedVnffg`, `affectedNs` and `affectedSap` (see clause 8.3.2.2 [5]).

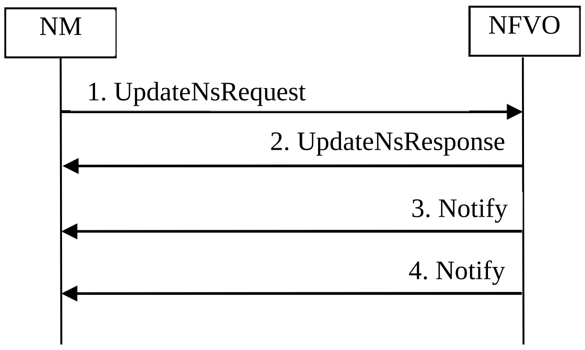

Figure 4.4.5.1-1: NS update to associate an NS instance to a different NSD version

4.4.5.2 Modifying VNF instance information through Os-Ma-nfvo

Figure 4.4.5.2-1 depicts the procedure of modifying VNF instance information through the Os-Ma-nfvo reference point (see clause 7.3.5 [5]).

1. NM sends to NFVO an *UpdateNsRequest* with parameters `nsInstanceId`, `updateType =` "ModifyVnfInformation", `modifyVnfInfoData`, and `updateTime` to modify the VNF instance information (see clause 7.3.5.2 [5]). The `modifyVnfInfoData` contains the parameters that are needed for VNF instance information modification, namely `vnfInstanceId`, and list of `newValues` (see clause 8.3.4.17 [5]).

2. NFVO sends to NM an *UpdateNsResponse* with parameter `lifecycleOperationOccurrenceId` providing the identifier of the NS lifecycle operation occurrence (see clause 7.3.5.3 [5]).

3. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "UpdateNs", and `notificationType` = "start" to indicate the start of the NS update that includes the VNF instance information modification (see clause 8.3.2.2 [5]).

4. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "UpdateNs", `notificationType` = "result" to indicate the end result of NS update that includes the VNF instance information modification, and `affectedVnf` providing information about the VNF instance whose information has been modified, including `vnfInstanceId`, `vnfdId`, `vnfProfileId`, `vnfName` and `changeType` = "information modified" (see clauses 8.3.2.2 and 8.3.2.3 [5]).

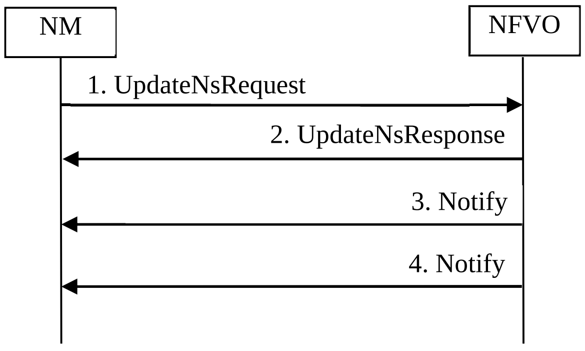

Figure 4.4.5.2-1: Modifying VNF instance information through Os-Ma-nfvo

4.4.5.3 Modifying VNF instance configuration through Os-Ma-nfvo

Figure 4.4.5.3-1 depicts the procedure of modifying VNF instance configuration through the Os-Ma-nfvo reference point (see clause 7.3.5 [5]).

1. NM sends to NFVO an *UpdateNsRequest* with parameters `nsInstanceId`, `updateType =` “ModifyVnfConfig”, `modifyVnfConfigData`, and `updateTime` to modify the VNF instance configuration (see clause 7.3.5.2 [5]). The `modifyVnfConfigData` contains the parameters that are needed for VNF instance configuration modification, namely `vnfInstanceId`, list of `vnfConfigurationData` and list of `extVirtualLink` (see clause 8.3.4.18 [5]).
2. NFVO sends to NM an *UpdateNsResponse* with parameter `lifecycleOperationOccurrenceId` providing the identifier of the NS lifecycle operation occurrence (see clause 7.3.5.3 [5]).
3. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "UpdateNs", and `notificationType` = "start" to indicate the start of the NS update that includes the VNF instance configuration modification (see clause 8.3.2.2 [5]).
4. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "UpdateNs", `notificationType` = "result" to indicate the end result of the NS update that includes the VNF instance configuration modification, and `affectedVnf` providing information about the modified VNF instance whose configuration has been modified, including `vnfInstanceId`, `vnfdId`, `vnfProfileId`, `vnfName` and `changeType` = "configuration modified" (see clauses 8.3.2.2 and 8.3.2.3 [5]).

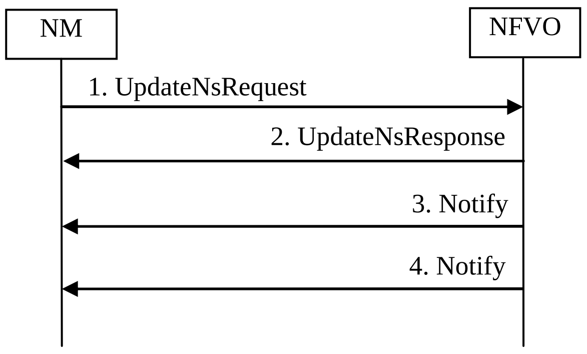

Figure 4.4.5.3-1: Modifying VNF instance configuration through Os-Ma-nfvo

4.4.5.4 NS update to add a PNF to the NS instance

Figure 4.4.5.4-1 depicts the procedure of NS update to add a PNF to the NS instance (see clause 7.3.5 [5]). It is assumed that the PNFD of the PNF to be added has been on-boarded, and the NS where the PNF instances are to be added has been instantiated.

1. NM sends to NFVO an *UpdateNsRequest* (see clause 7.3.5 of [5]) with the following parameters (see clause 7.3.5.2 of [5]) to on-board the NSD:

   - `nsInstanceId`: the identifier of the NS instance being updated.

   - `updateType` = “AddPnf” to indicate the type of update operation.

   - `addPnfData`: the PNF information that contain the following attributes (see clause 8.3.4.32.2 of [5]):

   > `pnfId`: the identifier of the PNF;

   > `pnfName`: human readable name of the PNF;

> `pnfdId`: identifier (reference to) the PNFD related to this PNF;

> `pnfProfileId`: identifier (reference to) the PNF Profile to be used for this PNF;

> `cpData`: information of the external CP of the PNF.

2. NFVO sends to NM an *UpdateNsResponse* (see clause 7.3.5 of [5]) with the attribute `lifecycleOperationOccurrenceId` that is the identifier of the NS lifecycle operation occurrence.

3. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "NsUpdate", and `notificationType` = "start" to indicate the start of the NS instantiation (see clause 8.3.2.2 [5]).

4. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "NsUpdate ", and `notificationType` = "result" to indicate the end result of the NS instantiation (see clause 8.3.2.2 [5]).

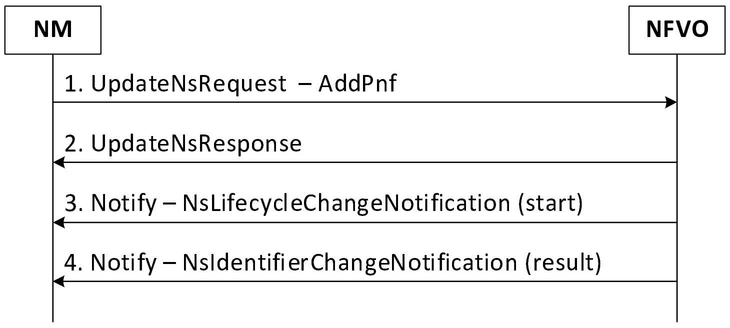
NM
NFVO
1. UpdateNsRequest – AddPnf
2. UpdateNsResponse
3. Notify – NsLifecycleChangeNotification (start)
4. Notify – NsIdentifierChangeNotification (result)

Figure 4.4.5.4-1: NS update to add a PNF to the NS instance

4.4.5.5 NS update to modify a PNF in the NS instance

Figure 4.4.5.5-1 depicts the procedure of NS update to modify the PNF to the NS instance (see clause 7.3.5 [5]). It is assumed that the NS where the PNF instances are to be modified has been instantiated.

1. NM sends to NFVO an *UpdateNsRequest* (see clause 7.3.5 of [5]) with the following parameters (see clause 7.3.5.2 of [5]) to on-board the NSD:

   - `nsInstanceId`: the identifier of the NS instance being updated.

   - `updateType` = "ModifyPnf" to indicate the type of update operation.

   - `modifyPnfData`: the PNF information that contain the following attributes (see clause 8.3.4.33.2 of [5]):

     > `pnfId`: the identifier of the PNF;

     > `pnfName`: human readable name of the PNF;

     > `cpData`: information of the external CP of the PNF.

2. NFVO sends to NM an *UpdateNsResponse* (see clause 7.3.5 of [5]) with the attribute `lifecycleOperationOccurrenceId` that is the identifier of the NS lifecycle operation occurrence.

3. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`, `operation` = "NsUpdate", and `notificationType` = "start" to indicate the start of the NS instantiation (see clause 8.3.2.2 [5]).

4. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an `NsLifecycleChangeNotification` information element with attributes `nsInstanceId`, `lifecycleOperationOccurrenceId`,

operation = "NsUpdate ", and notificationType = "result" to indicate the end result of the NS instantiation (see clause 8.3.2.2 [5]).

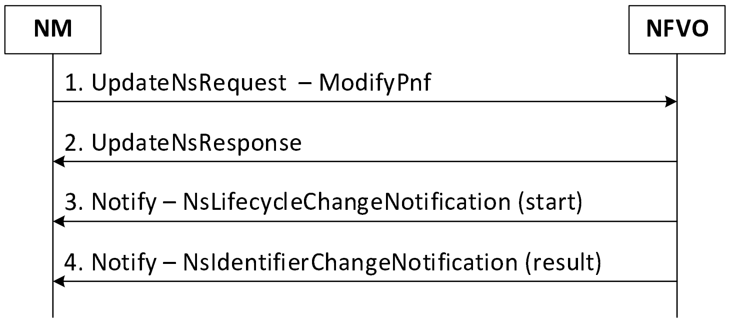
NM
NFVO
1. UpdateNsRequest – ModifyPnf
2. UpdateNsResponse
3. Notify – NsLifecycleChangeNotification (start)
4. Notify – NsIdentifierChangeNotification (result)

Figure 4.4.5.5-1: NS update to modify a PNF in the NS instance

4.4.5.6 NS update to remove a PNF from the NS instance

Figure 4.4.5.6-1 depicts the procedure of NS update to remove the PNF from the NS instance (see clause 7.3.5 [5]). It is assumed that the NS where the PNF instances are to be removed has been instantiated.

1. NM sends to NFVO an *UpdateNsRequest* (see clause 7.3.5 of [5]) with the following parameters (see clause 7.3.5.2 of [5]) to on-board the NSD:

   - nsInstanceId: the identifier of the NS instance being updated.

   - updateType = “RemovePnf” to indicate the type of update operation.

   - removePnfId: the identifier of the PNF(s) in the NS instance that are to be removed.

2. NFVO sends to NM an *UpdateNsResponse* (see clause 7.3.5 of [5]) with the attribute lifecycleOperationOccurrenceId that is the identifier of the NS lifecycle operation occurrence.

3. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an NsLifecycleChangeNotification information element with attributes nsInstanceId, lifecycleOperationOccurrenceId, operation = "NsUpdate", and notificationType = "start" to indicate the start of the NS instantiation (see clause 8.3.2.2 [5]).

4. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying an NsLifecycleChangeNotification information element with attributes nsInstanceId, lifecycleOperationOccurrenceId, operation = "NsUpdate ", and notificationType = "result" to indicate the end result of the NS instantiation (see clause 8.3.2.2 [5]).

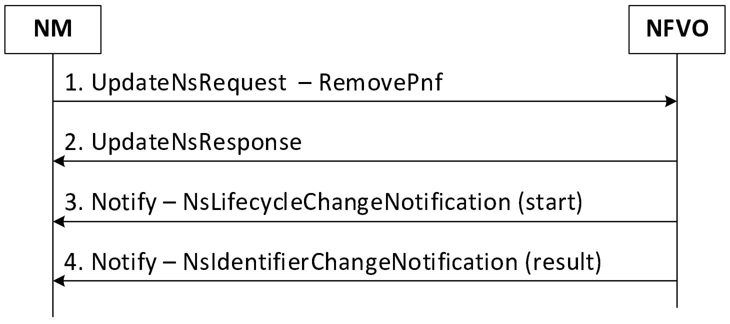
NM
NFVO
1. UpdateNsRequest – RemovePnf
2. UpdateNsResponse
3. Notify – NsLifecycleChangeNotification (start)
4. Notify – NsIdentifierChangeNotification (result)

Figure 4.4.5.6-1: NS update to remove a PNF from the NS instance

4.4.6 Subscription regarding NS Instance lifecycle changes

The Figure 4.4.6-1 depicts a procedure of subscription to NS instance lifecycle change notifications

1. NM sends *SubscribeRequest* message to NFVO with input parameter *filter* described in 7.4.2.2 [5] used for selecting the notifications, which can be on the NS instance(s) of interest or other attributes of the notification, to subscribe for the notifications sent by the NFVO supported on the NS lifecycle change notification interface.

2. NFVO sends *SubscribeResponse* message to NM to indicate if the subscription has been successful or not with a standard success/error result. Output parameter *subscriptionId* is specified in 7.4.2.3 [5] to provide the identifier of the subscription realized. After successful subscription, the consumer (NM) is registered to receive notifications supported on the NS lifecycle change notification interface. For a particular subscription, only notifications matching the filter will be delivered to the consumer (see clause 7.4.2.4 [5]).

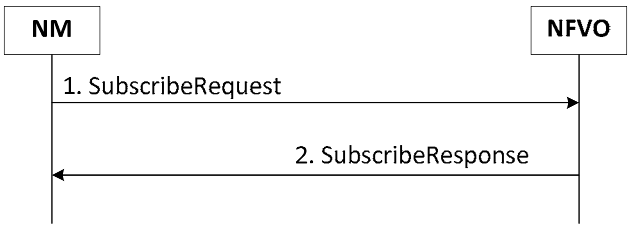

Figure 4.4.6-1: Subscribing to NS lifecycle change notifications through Os-Ma-nfvo

4.4.7 Create NS Instance identifier

Figure 4.4.7-1 depicts a procedure for creation of an NS instance identifier, and an associated instance of an NsInfo information element, identified by that identifier, in the NOT_INSTANTIATED state without instantiating the NS or doing any additional lifecycle operation(s). It allows the immediate return of an NS instance identifier that can be used in subsequent lifecycle operations, such as the Instantiate NS operation (see NS state model in clause D.3 [5]).

1. NM sends *CreateNsIdentifierRequest* to NFVO with `nsdId` parameter referencing the NSD used to create this NS instance, `nsName` parameter providing human readable name of the NS instance and `nsDescription` parameter providing human readable description of the NS instance (see clause 7.3.2 [5]).

2. NFVO sends *CreateNsIdentifierResponse* to NM with the `nsInstanceId` identifier of the instance of a NS that has been created (see clause 7.3.2.4 [5]).

3. NFVO sends to subscribers a *Notify* (see clause 7.4.3 [5]) carrying `NsIdentifierCreationNotification` information element with attribute `nsInstanceId` to indicate the NS instance idenfier creation (see clause 8.3.2.9 [5]).

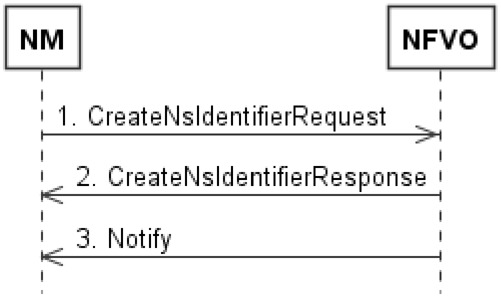

Figure 4.4.7-1: Create NS Instance identifier procedure

4.4.8 Delete NS Instance identifier

Figure 4.4.8-1 depicts the procedure of an NS instance identifier deletion initiated through the Os-Ma-Nfvo reference point (see clause 7.3.8 [5]).

1. NM sends to NFVO a *DeleteNsRequest* with parameter `nsInstanceId` to delete the NS instance identifier (see clause 7.3.8.2 [5]).

2. NFVO sends to NM a *DeleteNsResponse* (see clause 7.3.8.2 [5]).

3. NFVO sends to NM a *Notify* (see clause 7.4.3 [5]) carrying `NsIdentifierObjectDeletionNotification` information element with attributes `nsInstanceId` to indicate the result of the NS instance identifier deletion (see clause 8.3.2.10 [5]).

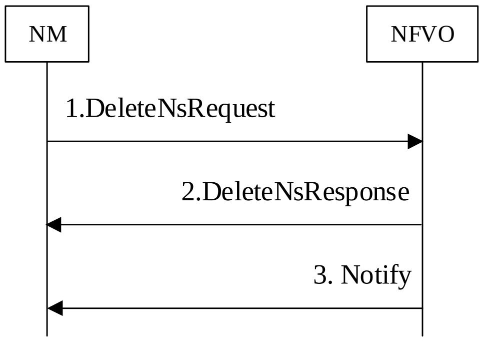

Figure 4.4.8-1: NS instance identifier deletion

4.4.9 Procedure for the Notify operation for notifications to NM

The Figure 4.4.9-1 depicts a procedure of notification on NS instance lifecycle change notifications

1. NFVO sends *Notify* message to NM according to 7.4.3 [5]. This operation distributes to subscribers the notifications supported on the NS lifecycle changes notification interface. In order to receive notifications, the consumer (NM) has to perform an explicit Subscribe operation beforehand.

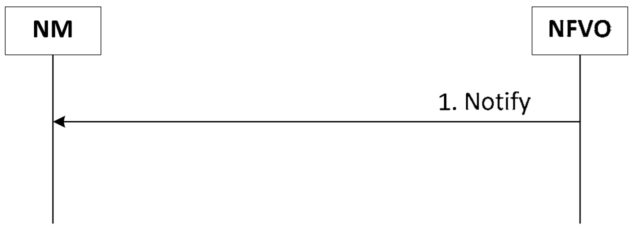

Figure 4.4.9-1: Procedure for the Notify operation

The following notifications can be notified/sent by this operation:

- NsLifecycleChangeNotification. See clause 8.3.2.2 [5] with the list of attributes in the Table 8.3.2.2.3-1 [5].
- NsIdentifierCreationNotification. See clause 8.3.2.9 [5] with the attribute nsInstanceId per Table 8.3.2.9.3-1 [5].
- NsIdentifierDeletionNotification. See clause 8.3.2.10 [5] with the attribute nsInstanceId per Table 8.3.2.10.3-1.

4.5 NS Descriptor (NSD) procedures

4.5.1 NSD on-boarding

Figure 4.5.1-1 depicts a procedure for on-boarding an NSD in the NFVO. Associated descriptors (VLD and VNFFGD), that are part of the NSD, are on-boarded at the same time. All descriptors needed by the NSD: VNFD, PNFD and NSD for nested NSs shall be on-boarded before being able to successfully on-board the NSD.

1. NM sends *OnboardNsdRequest* to NFVO with `nsd` parameter representing the NSD to be on-boarded and list of `userDefinedData` key-value pairs providing user defined data for the NSD (see clause 7.2.2 [5]).
2. NFVO sends *OnboardNsdResponse* to NM with the `nsdInfoId` identifier of the on-boarded instance of the NSD (see clause 7.2.2.4 [5]).

3. NFVO sends to subscribers a *Notify* (see clause 7.2.13 [5]) carrying `NsdOnBoardingNotification` information element with attributes `nsdInfoId` and `nsdId` to indicate on-boarding of NSD (see clause 8.2.6 [5]).

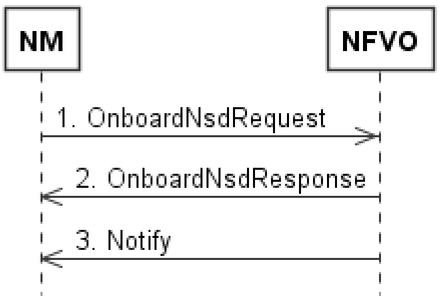

Figure 4.5.1-1: NSD on-boarding procedure

4.5.2 NSD enabling

Figure 4.5.2-1 depicts a procedure for enabling a previously disabled NSD instance, allowing again its use for instantiation of new network service with this descriptor. The "In use/Not in use" sub-state does not change as a result of the operation.

1. NM sends *EnableNsdRequest* to NFVO with `nsdInfoId` parameter representing the identifier of on-boarded NSD to be enabled (see clause 7.2.3 [5]).

2. NFVO sends *EnableNsdResponse* to NM to indicate the result of the operation (see clause 7.2.3.4 [5]).

3. NFVO sends to subscribers a *Notify* (see clause 7.2.13 [5]) carrying `NsdChangeNotification` information element with attributes `nsdInfoId`, `changeType`, `operationalState` and `deletionPending` to indicate a change of status of NSD (see clause 8.2.7 [5]).

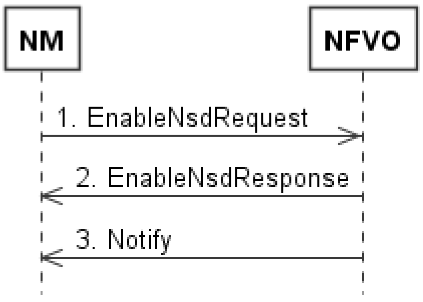

Figure 4.5.2-1: NSD enabling procedure

4.5.3 NSD disabling

Figure 4.5.3-1 depicts a procedure for disabling a previously enabled NSD instance, preventing any further use for instantiation of new network service with this descriptor. The "In use/Not in use" sub-state does not change because of the operation.

1. NM sends *DisableNsdRequest* to NFVO with `nsdInfoId` parameter representing the identifier of on-boarded NSD to be disabled (see clause 7.2.4 [5]).

2. NFVO sends *DisableNsdResponse* to NM to indicate the result of the operation (see clause 7.2.4.4 [5]).

3. NFVO sends to subscribers a *Notify* (see clause 7.2.13 [5]) carrying `NsdChangeNotification` information element with attributes `nsdInfoId`, `changeType`, `operationalState` and `deletionPending` to indicate a change of status of NSD (see clause 8.2.7 [5]).

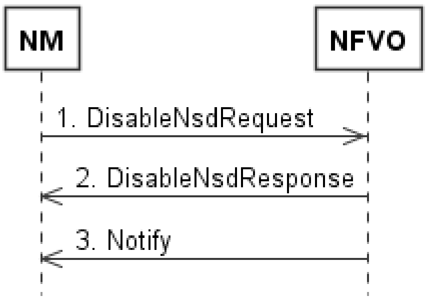

Figure 4.5.3-1: NSD disabling procedure

4.5.4 NSD querying

Figure 4.5.4-1 depicts the procedure of querying NSD information through the Os-Ma-nfvo reference point (see clause 7.2.7 [5]).

1. NM sends to NFVO a *QueryNsdRequest* with parameters `filter` and `attributeSelector` used to filter the NSDs on which the query applies and the attributes that will be returned for the instances of NSD(s) matching the filter (see clause 7.2.7.2 [5]).

2. NFVO sends to NM a *QueryNsdResponse* with parameter `queryResult` providing the information that is selected according to parameters `filter` and `attributeSelector` (see clause 7.2.7.3 [5]). The result of the operation indicates if it has been successful or not with a standard success/error result.

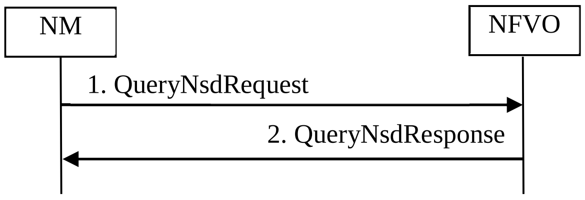

Figure 4.5.4-1: Query NSD information through Os-Ma-nfvo

4.5.5 NSD deletion

Figure 4.5.5-1 depicts a procedure for deletion of one or more NSD(s). It is possible to delete only a single version of an NSD or all versions. An NSD can only be deleted when there is no instantiated NS using it. An NSD in the deletion pending state can no longer be enabled, disabled or updated. It is not possible to instantiate NS(s) using an NSD in the deletion pending state.

1. NM sends *DeleteNsdRequest* to NFVO with list of `nsdInfoId` parameter representing the identifier(s) of on-boarded NSD to be deleted and an optional `applyOnAllVersions` parameter indicating if the delete operation is to be applied on all versions of this NSD. By default, if `applyOnAllVersions` parameter is not present, the request applies only on the current NSD version. (see clause 7.2.6 [5]).

2. NFVO sends *DeleteNsdResponse* to NM with list of `deletedNsdInfoId` parameter representing the identifier(s) of deleted NSD(s) to indicate the result of the operation (see clause 7.2.6.4 [5]).

3. NFVO sends to subscribers a *Notify* (see clause 7.2.13 [5]) carrying `NsdChangeNotification` information element with attributes `nsdInfoId`, `changeType`, `operationalState` and `deletionPending` to indicate a change of status of NSD (see clause 8.2.7 [5]).

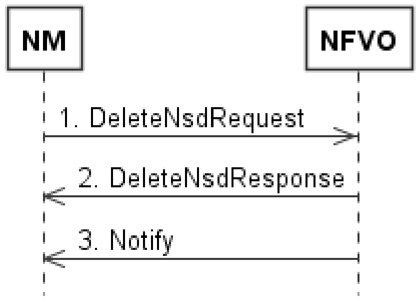

Figure 4.5.5-1: NSD deletion procedure

4.5.6 NSD updating

Figure 4.5.6-1 depicts a procedure for updating an already on-boarded NSD, creating a new version of the NSD. The procedure can also be used to update the userDefinedData of an existing NsdInfo information element without creating a new version of the NSD. The previous versions of the NSDs are not modified. It is possible to add (remove) constituent descriptors (i.e. VNFDs, PNFDs, nested NSDs, VLDs, VNFFGDs and Service Access Point Descriptors (SAPDs)) to (from) an NSD via the Update NSD operation. This is done by changing the various descriptor references in the new NSD. For example, to add VNFDs to an NSD, the NM adds corresponding VNFD identifiers to the list of vnfdIds in the new NSD. To remove VNFDs, the NM simply does not include the vnfdIds (of the VNFDs to be removed) in the new NSD.

1. NM sends *UpdateNsdRequest* to NFVO with `nsdInfoId` parameter representing the identifier of on-boarded NSD to be updated, an optional (only present if the NSD itself is updated) `nsd` parameter representing the new NSD to be created and list of `userDefinedData` key-value pairs optional parameter representing the user defined data to be updated (for existing keys, the value is replaced). At least one of the two parameters (`nsd` or `userDefinedData`) needs to be present. If `nsd` is not present, the operation is used only to update existing or add additional user defined data using the userDefinedData parameter (see clause 7.2.5 [5]).

2. NFVO sends *UpdateNsdResponse* to NM with `nsdInfoId` parameter representing the identifier of the updated NSD to indicate the result of the operation (see clause 7.2.5.4 [5]).

3. NFVO sends to subscribers a *Notify* (see clause 7.2.13 [5]) carrying `NsdChangeNotification` information element with attributes `nsdInfoId`, `changeType`, `operationalState` and `deletionPending` to indicate a change of status of NSD (see clause 8.2.7 [5]).

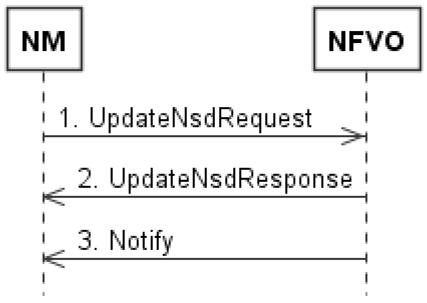

Figure 4.5.6-1: NSD updating procedure

4.5.7 Subscribe to NSD change notifications

The Figure 4.5.7-1 depicts a procedure of subscription to NSD change notifications.

1. NM sends *SubscribeRequest* message to NFVO with input parameter `filter` for selecting the NSD(s) and the related change notifications to subscribe to, as described in 7.2.12 [5]. The filter can contain information about specific types of changes to subscribe to, or attributes of the NsdInfo.

2. NFVO sends *SubscribeResponse* message to the NM to indicate if the subscription has been successful or not with a standard success/error result. Output parameter `subscriptionId` is specified in 7.2.12.3 [5] to provide the identifier of the subscription realized. After successful subscription, the consumer (NM) is registered to

receive notifications supported on the NSD Management interface. For a particular subscription, only notifications matching the filter will be delivered to the consumer (see clause 7.2.13.1 [5] for full list of possible notifications).

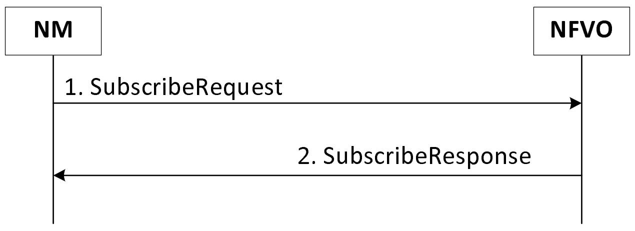

Figure 4.5.7-1: Subscribing to NSD change notifications through Os-Ma-nfvo

4.5.8 Notify operation for NSD management changes

The Figure 4.5.8-1 depicts a procedure of delivery of notifications related to NSD management changes.

1. NFVO sends *Notify* message to the NM according to the clause 7.2.13 in [5]. This operation delivers to the NM the notifications related to NSD Management changes. In order to receive notifications, the NM shall perform an explicit Subscribe operation beforehand.

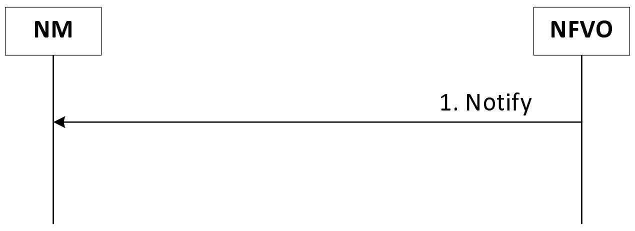

Figure 4.5.8-1: Procedure for the Notify operation for NSD management changes

The following notifications can be sent by this operation:

- `NsdOnBoardingNotification`. See clause 8.2.6 in [5].

The notification contains the following parameters:

- `nsdInfoId`: Identifier of the on-boarded instance of the NSD
- `nsdId`: Identifies the NSD being on-boarded.
- `NsdChangeNotification`. See clause 8.2.7 in [5].

The notification contains the following parameters:

- `nsdInfoId`:Identifier of the on-boarded instance of the NSD.
- `changeType`: It categorizes the type of change. Possible values can be "change of operational state of an on-boarded NSD", "NSD in deletion pending", and "deletion of an NSD".
- `operationalState`:New operational state of the NSD. Only present when changeType is "change of operational state".
- `deletionPending`: Indicates if the deletion of the NSD instance has been requested but the NSD still being used by instantiated NSs. Only present when changeType is "NSD in deletion pending".

4.6 PNFD procedures

4.6.1 PNFD on-boarding

Figure 4.6.1-1 depicts a procedure for on-boarding a PNFD in the NFVO, making it available to be used by NSDs.

1. NM sends *OnboardPnfdRequest* to NFVO with `pnfd` parameter representing the PNFD to be on-boarded and list of `userDefinedData` key-value pairs providing user defined data for the PNFD (see clause 7.2.8 [5]).

2. NFVO sends *OnboardPnfdResponse* to NM with the `pnfdInfoId` identifier of the on-boarded instance of the PNFD (see clause 7.2.8.4 [5]).

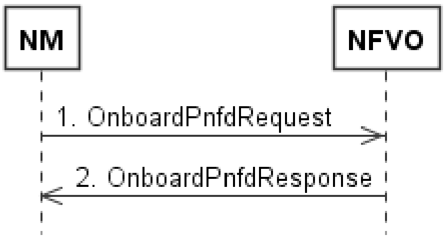

Figure 4.6.1-1: PNFD on-boarding procedure

4.6.2 PNFD updating

Figure 4.6.2-1 depicts a procedure for updating a PNFD, creating a new version of already on-boarded PNFD. The procedure can also be used to update the userDefinedData of an existing PnfInfo information element without creating a new version of the PNFD. The previous versions of the PNFD are not modified.

1. NM sends *UpdatePnfdRequest* to NFVO with `pnfdInfoId` parameter representing the identifier of on-boarded PNFD to be updated, an optional (only present if the PNFD itself is updated) `pnfd` parameter representing the new PNFD to be created and list of `userDefinedData` key-value pairs optional parameter representing the user defined data to be updated (for existing keys, the value is replaced). At least one of the two parameters (`pnfd` or `userDefinedData`) needs to be present. If `pnfd` is not present, the operation is used only to update existing or add additional user defined data using the userDefinedData parameter (see clause 7.2.9 [5]).

2. NFVO sends *UpdatePnfdResponse* to NM with `pnfdInfoId` parameter representing the identifier of the updated PNFD to indicate the result of the operation (see clause 7.2.9.4 [5]).

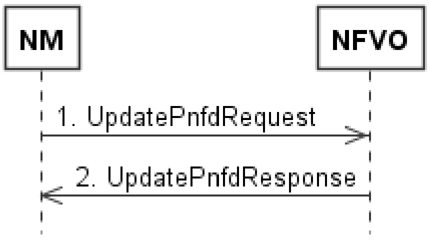

Figure 4.6.2-1: PNFD updating procedure

4.6.3 PNFD deletion

Figure 4.6.3-1 depicts a procedure for deletion of one or more PNFD(s). A PNFD can only be deleted when there is no NS (in the active or NOT_INSTANTIATED state) using it. It is not possible to instantiate NSs that include a PNFD in deletion pending state.

1. NM sends *DeletePnfdRequest* to NFVO with list of `pnfdInfoId` parameter representing the identifier(s) of on-boarded PNFD(s) to be deleted and an optional `applyOnAllVersions` parameter indicating if the delete operation is to be applied on all versions of this PNFD. By default, if `applyOnAllVersions` parameter is not present, the request applies only on the current PNFD version. (see clause 7.2.10 [5]).

2. NFVO sends *DeletePnfdResponse* to NM with list of `deletedPnfdInfoId` parameter representing the identifier(s) of deleted PNFD(s) to indicate the result of the operation (see clause 7.2.10.4 [5]).

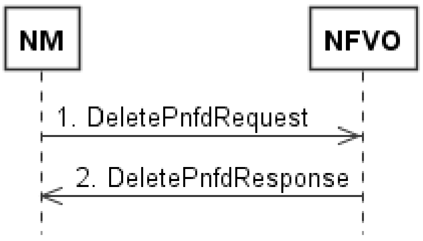

Figure 4.6.3-1: PNFD deletion procedure

4.6.4 PNFD querying

Figure 4.6.4-1 depicts a procedure to query the NFVO concerning details of one or more PNFDs.

1. NM sends *QueryPnfdRequest* to NFVO with `filter` and `attributeSelector` used to filter the PNFD(s) on which the query applies, based on the attributes of `PnfdInfo` and select the information attributes of `PnfdInfo` that are requested to be returned (see clause 7.2.11 [5]).

2. NFVO sends *QueryPnfdResponse* to NM with parameter `queryResult` providing the information of the on-boarded PNFD matching the input `filter` that is selected according to `attributeSelector` (see clause 7.2.11.4 [5]).

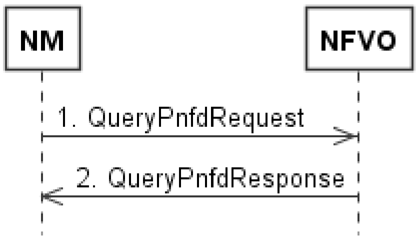

Figure 4.6.4-1: PNFD querying procedure

Annex A (informative): Change history

Change history							
Date	**Meeting**	**TDoc**	**CR**	**Rev**	**Cat**	**Subject/Comment**	**New version**
2018-03	SA#79	SP-180059	0002	1	F	Scope extension	14.1.0
2018-03	SA#79	SP-180059	0004	1	F	Procedures of PNF addition, removal, and modification	14.1.0
2018-06	-	-	-	-	-	Update to Rel-15 version (MCC)	**15.0.0**

History

Document history		
V15.0.0	July 2018	Publication