|  |
| --- |
| webwxgetmsgimg (7).jpeg |

|  |
| --- |
| Technical Specification |
|  |

|  |
| --- |
| Non-RT RIC: Architecture |

|  |
| --- |
|  |
| Copyright © 2024 by the O-RAN ALLIANCE e.V.  The copying or incorporation into any other work of part or all of the material available in this specification in any form without the prior written permission of O-RAN ALLIANCE e.V. is prohibited, save that you may print or download extracts of the material of this specification for your personal use, or copy the material of this specification for the purpose of sending to individual third parties for their information provided that you acknowledge O-RAN ALLIANCE as the source of the material and that you inform the third party that these conditions apply to them and that they must comply with them.  O-RAN ALLIANCE e.V., Buschkauler Weg 27, 53347 Alfter, Germany  Register of Associations, Bonn VR 11238, VAT ID DE321720189 |

Contents

Foreword 4

Modal verbs terminology 4

1 Scope 5

2 References 5

2.1 Normative references 5

2.2 Informative references 5

3 Definition of terms, symbols and abbreviations 6

3.1 Terms 6

3.2 Symbols 6

3.3 Abbreviations 6

4 Overview 7

4.1 Non-RT RIC in O-RAN overall architecture 7

4.2 Void 7

5 Requirements 7

5.1 Requirements for the Non-RT RIC framework 7

5.2 Requirements for R1 10

5.3 Requirements for rApps 10

6 Non-RT RIC Architecture 10

6.1 General 10

6.2 R1 interface 12

7 R1 Service Definitions 12

7.1 General description 12

7.2 Service management and exposure services 13

7.3 Data management and exposure services 13

7.4 A1-related Services 14

7.4.1 A1 policy management services 14

7.4.2 A1 enrichment information related services 15

7.5 RAN OAM-related Services 15

7.6 O2-related services 16

7.7 AI/ML workflow services 16

7.8 rApp management services 17

7.9 TE&IV-related services 17

8 Non-RT RIC Function Definitions 17

8.1 General description 17

8.2 Service management and exposure functions 18

8.2.1 Service registration and discovery functions 18

8.2.2 Authentication and authorisation functions 18

8.3 Data management and exposure functions 18

8.4 A1-related functions 19

8.4.1 A1 policy functions 19

8.4.2 A1 enrichment information functions 20

8.5 RAN OAM-related functions 20

8.6 O2-related functions 20

8.7 AI/ML workflow functions 21

9 External Interfaces 21

9.1 A1 Interface 21

9.2 Void 21

10 Procedures 21

10.1 Data management and exposure procedures 21

10.1.0 Overview 21

10.1.1 Subscribe data procedures in data consumption and collection 22

10.1.2 Request data procedure in data consumption and data collection 27

10.1.3 O1 related data collection procedures based on data subscription 29

10.1.4 O2 related data collection procedures based on data subscription 31

10.1.5 Data offer procedures 33

10.2 A1-related procedures 36

10.2.1 EI results collection from rApps 36

10.2.2 rApp creating a policy with a Near-RT RIC identifier 39

10.3 RAN OAM-related procedures 40

10.3.1 Alarm query procedure 40

10.3.2 Change alarm acknowledgement state procedure 41

10.3.3 Read configuration data procedure 43

10.3.4 Write configuration changes procedure 45

10.4 AI/ML workflow procedures 47

10.4.1 rApps register, discover AI/ML models, and subscribe to models changes 47

10.4.2 AI/ML training procedures 50

10.4.3 AI/ML model storage and retrieval procedures 57

Annex (informative): Change History 59

# Foreword

This Technical Specification (TS) has been produced by WG2 of the O-RAN Alliance.

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

version xx.yy.zz

where:

xx: the first digit-group is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc. (the initial approved document will have xx=01). Always 2 digits with leading zero if needed.

yy: the second digit-group is incremented when editorial only changes have been incorporated in the document. Always 2 digits with leading zero if needed.

zz: the third digit-group included only in working versions of the document indicating incremental changes during the editing process. External versions never include the third digit-group. Always 2 digits with leading zero if needed.

# 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 O-RAN Drafting Rules (Verbal forms for the expression of provisions).

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

# 1 Scope

The present document specifies the technical specification for the Non-RT RIC (RAN Intelligent Controller) architecture. This document collects requirements on the Non-RT RIC framework, hosted applications, and services of the R1 interface. This document specifies Non-RT RIC framework functionalities and services exposed to the applications.

# 2 References

## 2.1 Normative 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, O-RAN cannot guarantee their long-term validity.

The following referenced documents are necessary for the application of the present document.

[1] Void.

[2] O-RAN TS: “O-RAN Architecture Description” (“OAD”).

[3] O-RAN TS: “Non-RT RIC and A1/R1 interface: Use Case and Requirements” (“WG2 UCR”).

[4] O-RAN TS: “A1 interface: General Aspects and Principles” (“A1GAP”).

[5] O-RAN TS: “A1 interface: Application Protocol” (“A1AP”).

[6] O-RAN TS: “A1 interface: Transport Protocol” (“A1TP”).

[7] O-RAN TS: “A1 interface: Type Definitions” (“A1TD”).

[8] O-RAN TS: “O-RAN Operations and Maintenance Architecture” (“OAM Arch”).

[9] O-RAN TS: “R1 interface: General Aspects and Principles” (“R1GAP”).

[10] O-RAN TS: “Security Requirements and Controls Specification” (“SRS”).

[11] O-RAN TS: “O-RAN Operations and Maintenance Interface Specification” (“O1 interface”).

[12] O-RAN TS: “Topology Exposure and Inventory Management Services Use Cases and Requirement Specification” (“TE&IV”).

[13] O-RAN TS: “O2 Interface General Aspects and Principles” (“O2GAP”).

## 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, O-RAN 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 regards to a particular subject area.

[i.1] 3GPP TR 21.905: “Vocabulary for 3GPP Specifications”.

# 3 Definition of terms, symbols and abbreviations

## 3.1 Terms

For the purposes of the present document, the terms and definitions given in 3GPP TR 21.905 [i.1] 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 [i.1].

**Functions anchored inside the Non-RT RIC framework:** Logical functions in the SMO framework that are part of the Non-RT RIC framework.

**Functions anchored outside the Non-RT RIC framework:** Logical functions in the SMO framework that are not part of the Non-RT RIC framework.

**Non-RT RIC (O-RAN Non-Real-Time RAN Intelligent Controller):** A logical function within SMO that enables non-real-time control and optimization of RAN elements and resources, AI/ML workflow including model training and updates, and policy-based guidance of applications/features in Near-RT RIC.

**Near-RT RIC (O-RAN Near-Real-Time RAN Intelligent Controller):** A logical function that enables near-real-time control and optimization of RAN elements and resources via fine-grained data collection and actions over E2 interface.

**Non-anchored functions:** Logical functions in the SMO framework that may or may not be part of the Non-RT RIC framework.

**R1 Data Producer:** An entity capable of producing data which can be consumed by other entities and of registering for discovery information about the types of these data in the context of the data management and exposure service.

**R1 Data Consumer:** An entity capable of consuming data originating from other entities in the context of the data management and exposure service.

**R1 Service Producer:** An entity capable of producing an R1 service which can be consumed by other entities.

**R1 Service Consumer:** An entity capable of consuming an R1 service produced by other entities.

**AI/ML model:** An algorithm that applies AI/ML techniques to produce model output data based on model input data.

**AI/ML model training:** A process to train an AI/ML model by learning the input/output relationship in a data driven manner and obtain a trained AI/ML model that can be used for inference.

**AI/ML model parameter:** Internal parameter of an AI/ML model that is learnt during AI/ML model training.

**Hyperparameter:** A parameter whose value controls the learning process and the topology of an AI/ML model and which influences the quality of the learning process.

**AI/ML model inference:** A process of using an AI/ML model to produce model output data based on model input data.

**AI/ML model performance monitoring:** A procedure that monitors the inference performance of a deployed AI/ML model.

**rApp instance:** An individual occurrence of a Non-RT RIC application running in the Non-RT RIC runtime environment.

**rApp instance identifier:** A unique identifier for each rApp instance, assigned by the SMO/Non-RT RIC framework during rApp registration.

**DME type**: Data type managed and exposed by the DME services and identified by a DME type identifier.

## 3.2 Symbols

Void.

## 3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply.

API Application Programing Interface

AI Artificial Intelligence

CM Configuration Management

DME Data Management and Exposure

DMS Deployment Management Services

EI Enrichment Information

FCAPS Fault, Configuration, Accounting, Performance, Security

FM Fault Management

FOCOM Federated O-Cloud Orchestration and Management

HTTP Hypertext Transfer Protocol

IMS Infrastructure Management Services

ML Machine Learning

MOI Managed Object Instance

Near-RT RIC Near-Real-Time RAN Intelligent Controller

NFO Network Function Orchestration

Non-RT RIC Non-Real-Time RAN Intelligent Controller

OAM Operation, Administration and Maintenance

O-RU O-RAN Radio Unit

PM Performance Management

QoS Quality of Service

rAppId rApp instance Identifier

RAN Radio Access Network

SME Service Management and Exposure

SMO Service Management and Orchestration

TE&IV Topology Exposure and Inventory Management

UE User Equipment

URI Uniform Resource Identifier

# 4 Overview

## 4.1 Non-RT RIC in O-RAN overall architecture

Non-Real Time RAN Intelligent Controller (Non-RT RIC) is the functionality internal to the Service Management and Orchestration (SMO) framework in O-RAN overall architecture, as shown in O-RAN OAD[2], Figure 4.1-2.

* Non-RT RIC represents a subset of functionality of the SMO framework.
* Non-RT RIC supports intelligent RAN operation and optimization.
* Non-RT RIC logically terminates the A1 interface, and it provides policy-based guidance and enrichment information to the Near-RT RICs. AI/ML model management on the A1 interface is not specified in the present document.
* Non-RT RIC can access other SMO framework functionalities, for example influencing what is carried across the O1, open fronthaul M-plane and O2 interface.

## 4.2 Void

# 5 Requirements

## 5.1 Requirements for the Non-RT RIC framework

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN1] | The Non-RT RIC framework shall support functionality to register services along with their Service Producers in the Non-RT RIC and SMO. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN2] | The Non-RT RIC framework shall support functionality to allow Service Consumers to discover services. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN3] | The Non-RT RIC framework shall support functionality to allow Service Consumers to subscribe/unsubscribe notifications about newly registered/updated/deregistered services. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN4] | The Non-RT RIC framework shall support functionality to notify subscribed Service Consumers about newly registered/updated/deregistered services. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN5] | The Non-RT RIC framework shall support functionality to authenticate Service Consumers, if such functionality is not provided by the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN6] | The Non-RT RIC framework shall support functionality to authorize Service Consumers to access services, if such functionality is not prodived by the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN7] | The Non-RT RIC framework shall support functionality to send messages to and receive messages from the Near-RT RIC via the A1 interface. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN8] | The Non-RT RIC framework shall support functionality to allow Data Consumers (including rApps) to discover available DME types, if such functionality is not supported in the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN9] | The Non-RT RIC framework shall support functionality to allow Data Producers (including rApps) to register their DME types production capabilities, if such functionality is not supported in the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN10] | The Non-RT RIC framework shall support functionality to allow Data Consumers (including rApps) to subscribe/request instances of registered DME types for consumption, if such functionality is not supported in the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN10a] | The Non-RT RIC framework shall support functionality to allow Data Producers (including rApps) to offer instances of registered DME types for collection and storage, if such functionality is not supported in the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN11] | The Non-RT RIC framework shall support functionality to train AI/ML models, if such functionality is not supported in the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN12] | The Non-RT RIC framework shall support functionality to allow Service Consumers to store and retrieve trained AI/ML models, if such functionality is not supported in the SMO framework |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN13] | The Non-RT RIC framework shall support functionality to monitor the performance for deployed AI/ML models in runtime, if such functionality is not supported in the SMO framework |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN14] | The Non-RT RIC framework may support functionality to collect external enrichment information from external enrichment information sources. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN15] | The Non-RT RIC framework may support functionality to retrieve trained AI/ML models (and metadata) from external AI/ML service providers. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN16] | The Non-RT RIC framework may support functionality to allow external sources to inject RAN intents, suspend/resume/check rApps, and configure/check/initiate/suspend/resume/terminate AI/ML training processes. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN17] | The Non-RT RIC framework shall support functionality to consolidate the alarm information from multiple managed entities, if such functionality is not supported in the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN18] | The Non-RT RIC framework shall support functionality to store alarm acknowledgement state for alarms originating from managed entities that do not support alarm acknowledgement state change, if such functionality is not supported in the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN19] | The Non-RT RIC framework may have the capability to identify the potentially applicable Near-RT RIC(s) for A1 policy creation if the Near-RT RIC identifier is absent in the create A1 policy request received from the rApp. |

NOTE: Selection of potentially applicable Near-RT RIC(s) can be based, e.g., on the content of the received create A1 policy request or on configuration.

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN20] | The Non-RT RIC framework shall support the functionality to collect trace data, from multiple managed entities, if such functionality is not supported in the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN21] | The Non-RT RIC framework shall support the functionality to collect analytical data from the Near-RT RIC, if such functionality is not supported in the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN22] | The Non-RT RIC framework shall support functionality to allow Data Consumers (including rApps) to consume Deployment Telemetry metrics collected by the O2-related functions. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN23] | The Non-RT RIC framework shall support functionality to allow Data Consumers (including rApps) to consume Infrastructure Telemetry metrics collected by the O2-related functions. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN24] | The Non-RT RIC framework shall support functionality that allows managing the configuration of an rApp, if such functionality is not supported in the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN25] | The Non-RT RIC framework shall support functionality that allows obtaining, from an rApp, information about that rApp's performance, if such functionality is not supported in the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN26] | The Non-RT RIC framework shall support functionality that allows obtaining, from an rApp, fault information related to that rApp, if such functionality is not supported in the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN27] | The Non-RT RIC framework shall support functionality that allows obtaining, from an rApp, logging information reported by the rApp, if such functionality is not supported in the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN28] | The Non-RT RIC framework shall support functionality to store log information received from rApps, if such functionality is not supported in the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN29] | The Non-RT RIC framework shall support functionality that allows an rApp to obtain, over the R1 interface, TE&IV information based on the requirements defined in O-RAN TE&IV [12], if such functionality is not supported in the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN30] | The Non-RT RIC framework shall support functionality that allows registration and deregistration of an AI/ML model, if such functionality is not supported in the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN31] | The Non-RT RIC framework shall support functionality that allows the discovery of registered AI/ML models, if such functionality is not supported in the SMO framework. |

|  |  |
| --- | --- |
| [REQ-NRTFWK-FUN32] | The Non-RT RIC framework shall support functionality that allows the deployment of an AI/ML model, if such functionality is not supported in the SMO framework. |

In addition, the security requirements for the Non-RT RIC specified in O-RAN SRS [10], clause 5.1.2 apply.

## 5.2 Requirements for R1

O-RAN WG2 UCR [3] specifies R1 interface functional requirements.

## 5.3 Requirements for rApps

The security requirements for rApps specified in O-RAN SRS [10], clause 5.1.2 apply.

# 6 Non-RT RIC Architecture

## 6.1 General

Non-RT RIC is comprised of:

* Non-RT RIC Framework – Functionality internal to the SMO Framework that:
* Logically terminates the A1 interface to the Near-RT RIC.
* Exposes set of R1 services to Non-RT RIC Applications (rApps).
* Non-RT RIC Applications (rApps) – Applications that leverage the functionalities available in the SMO/Non-RT RIC framework to provide value added services related to RAN operation and optimization. The scope of rApps includes, but is not limited to, RAN configuration management, data analytics, producing training services, generating A1 policies, and providing enrichment information.

The following figure depicts the reference architecture of the Non-RT RIC as part of the SMO framework.



Figure 6.1-1: Non-RT RIC reference architecture

There are three categories of logical functions in Non-RT RIC framework and SMO framework:

* Functions anchored inside the Non-RT RIC framework, which are indicated in solid blue box in figure 6.1-1.
* Functions anchored outside the Non-RT RIC framework, which are indicated in solid orange box in figure 6.1-1.
* Non-anchored functions, which are indicated in dashed line box in figure 6.1-1.

The categories are defined in clause 3.1. The definitions make no assumption on the mandatory or optional qualifier of the functions being anchored or non-anchored.

The decomposition of logical functions is not mandatory, and it is up to vendor’s implementation.

NOTE 1: There might be other SMO framework functions or other Non-RT RIC framework functions which may provide additional services exposed via the R1 interface or provide EI for A1 enrichment information services. These are not specified in the present document.

NOTE 2: Other SMO framework functions may be defined by other O-RAN WGs and the potential impacts on Non-RT RIC architecture are not specified in the present document.

NOTE 3: Whether the A1-related functions are anchored functions or not is not specified in the present document.

NOTE 4: Among other functionalities, the Other SMO framework functions also realize OAM functionality related to RAN-specific network slicing.

NOTE 5: The Topology Exposure and Inventory Management functions are not elaborated in this present document.

Service management and exposure functions enable (not limited to): service registration, service discovery, service notification, authorization, authentication, communication support, and optional bootstrap and heartbeat services.

The rApp management functions enable the management of rApps within the context of the SMO/Non-RT RIC framework. Such functions enable (not limited to) the configuration of rApps, providing access to fault and performance related information from rApps, and facilitation of rApp logging functionality. Through the use of R1, rApp management functions can provide access to these functionalities in a consistent manner to all rApps. Such consistency is particularly valuable as rApps can be provided independently by different rApp vendors.

The implementation-specific interfaces, which are indicated by the dashed line in figure 6.1-1, allow the logical functions in SMO/Non-RT RIC framework to communicate with each other by means outside the scope of the present document.

## 6.2 R1 interface

The R1 interface is a service-based interface via which the R1 services can be consumed and produced.

The R1 services are a collection of services produced and consumed by logical functions in the SMO/Non-RT RIC framework or by rApps.

The invoker of an R1 service is referred to as R1 Service Consumer and the provider of an R1 service is referred to as R1 Service Producer. R1 Service Consumers can be rApps and/or SMO/Non-RT RIC framework functions. Further, R1 Service Producers can be rApps and/or SMO/Non-RT RIC framework functions.

The R1 termination enables the R1 Service Producers and R1 Service Consumers to exchange messages via the R1 interface.

An rApp as well as a SMO/Non-RT RIC framework function can produce zero or more R1 Services, likewise an rApp as well as a SMO/Non-RT RIC framework function can consume zero or more R1 Services.

The R1 termination enables the SMO/Non-RT RIC framework and the rApps to exchange messages via the R1 interface, assuming that the R1 service APIs and the rApps are registered with the R1 service management and exposure functions.

The communication support provided by the SMO/Non-RT RIC framework allows R1 Service Producers and R1 Service Consumers to exchange messages related to the R1 services. Different deployment options are possible, such as:

* Plain connectivity that enables the point-to-point exchange of service-related messages between R1 Service Producers and R1 Service Consumers e.g. over HTTP.
* Service meshes that enable the combination of discovery of R1 service API endpoints with the routing of service-related messages between R1 Service Consumers and R1 Service Producers.
* Message buses that enable R1 Service Producers to send service-related messages to zero or more R1 Service Consumers simultaneously.

# 7 R1 Service Definitions

## 7.1 General description

A service is a set of capabilities related to each other, offered by a Service Producer to Service Consumer(s) for consumption. The capabilities within a service typically have a common purpose and can be inter-dependent. In order to improve re-usability, the mutual coupling across any two services is usually avoided or minimized. Services are produced by Service Producers and consumed by Service Consumers through service endpoints (typically APIs). The interoperability of Service Producers and Service Consumers is ensured by support of the related service endpoints, but it does not depend on the decomposition of logical functions. Being a Service Producer and/or a Service Consumer is a role of an rApp or of a logical function in the SMO or the Non-RT RIC framework (see clause 6).

The set of R1 services which can be accessed via the R1 interface includes (but is not limited to):

* Service management and exposure services, which include service registration and service discovery services, authentication services, authorization services and communication support services. The service management and exposure services support extensibility of the set of R1 services.
* Data management and exposure services, which deliver data created or collected by Data Producers to Data Consumers according to their needs.
* A1-related services, which provide access to functionality related to A1.
* AI/ML workflow services, which provide access to AI and ML workflow.
* RAN OAM-related services, which provide access to RAN OAM functionality related to the O1 and open fronthaul M-plane interfaces, and to the OAM functionality related to RAN-specific network slicing in the SMO framework.
* O2-related services, which provide access to functionality related to O2.
* rApp management services, which provide functionality for managing rApps.
* TE&IV-related services, which provide access to topology and inventory data.

NOTE: A part of the services listed above may be used on other O-RAN-defined interfaces as well.

## 7.2 Service management and exposure services

The following service management and exposure services are defined:

* Bootstrap service (optional): It enables an rApp to discover the endpoints of the service management and exposure service.
* Registration of services: It allows to register services which includes storing and maintaining profile about each service. It further allows to deregister services and delete the service profile.
* Discovery of services: It supports service discovery for a Service Consumer to find services based on its selection criteria if it is provided. Further, based on the subscription, it notifies its Service Consumers about the newly registered/updated/deregistered services.
* Heartbeat service (optional): It enables an rApp to maintain its R1 service registration status with the service management and exposure services Producer.
* Authentication: It allows the authentication of a Service Producer and/or a Service Consumer.
* Authorisation: It supports to grant Service Consumers access to registered R1 services and to allow Service Producers to produce R1 services.
* Communication support (optional): It supports interaction of R1 Service Consumers and Producers with each other.

NOTE: Communication support is not specified in the present document.

## 7.3 Data management and exposure services

The following data management and exposure services are defined:

* Data registration service: Data registration allows Data Producers to communicate information about the DME types that they are able to produce. Data Producers need to be authorized to register DME types.
* Data discovery service: Data discovery allows Data Consumers to discover available DME types in the SMO/Non-RT RIC framework. Data Consumers need to be authorized to discover DME types.
* Data subscription service: Using the subscribe data procedure, it allows Data Consumers to consume data continuously from the SMO/Non-RT RIC framework and it further allows the SMO/Non-RT RIC framework to collect data continuously from Data Producers.
* Data request service: Using the request data procedure, it allows Data Consumers to consume data once from the SMO/Non-RT RIC framework and it further allows the SMO/Non-RT RIC framework to collect data once from Data Producers.
* Data delivery services: These allow to deliver data for consumption from the SMO/Non-RT RIC framework to Data Consumers. They further allow to deliver data for collection from Data Producers to the SMO/Non-RT RIC framework.
* Data offer service: It allows a Data Producer to trigger the collection of a data instance by the SMO/Non-RT RIC framework from this Data Producer.
* Data processing services (optional): These allow Service Consumers to access data processing in the SMO/Non-RT RIC framework.

NOTE: Data processing services are not specified in the present document.

Data Producers (e.g., rApps) communicate information about the DME types that they produce in data registration. Data Producers can communicate constraints related to the authorization of the Data Consumers to discover registered DME types and access produced data. Data Producers communicate information about how the data are collected by the SMO/Non-RT RIC framework (e.g., collection periodicity, event-trigger conditions, push or pull, etc.).

Data Consumers (e.g., rApps) communicate information about the DME types that they want to discover. Data Consumers subscribe/request data from the SMO/Non-RT RIC framework for consumption with specific DME type, periodicity, and delivery (or reporting) method, scope, etc. The delivery (or reporting) method contains information about how the subscribed/requested data are delivered from the SMO/Non-RT RIC framework (e.g., delivery periodicity, event-trigger conditions, push or pull, etc.).

A Data Producer that intends to trigger the collection of a data instance that it produces defines a data offer by providing to the SMO/Non-RT RIC framework information about the data instance it intends to produce. The information about the data offer includes the properties of the data instance and the supported delivery services. The collection of the data instance starts when the data offer creation has been confirmed by the SMO/Non-RT RIC framework.

SMO/Non-RT RIC framework collects data produced by Data Producers. SMO/Non-RT RIC framework delivers data to Data Consumers for consumption. SMO/Non-RT RIC framework stores collected data. If the collection of the data was triggered by a data subscription or data request, data storage is optional. SMO/Non-RT RIC framework can optionally process collected and/or stored data (e.g., quantization, normalization, labelling, correlation, etc.).

## 7.4 A1-related Services

### 7.4.1 A1 policy management services

A1 policy management services provided by the SMO/Non-RT RIC framework enable:

* Discovery of supported A1 policy types.
* rApp queries supported A1 policy types.
* rApp subscribes/unsubscribes notification for changes of supported A1 policy types. SMO/Non-RT RIC framework notifies the subscribed events (e.g., new policy types are added into the supporting list, existing policy types are removed, etc.).
* Creating, updating, querying, and deleting A1 policies.
* rApp creates A1 policies in the SMO/Non-RT RIC framework. SMO/Non-RT RIC framework assigns unique policy identifiers for created A1 policies.
* rApp updates A1 policies created by itself using assigned policy identifiers.
* rApp queries A1 policies in the SMO/Non-RT RIC framework.
* rApp queries A1 policies applied to a given scope (e.g., a UE, a group of UEs, a slice, a QoS flow, a cell, a Near-RT RIC, a policy type, etc.). SMO/Non-RT RIC framework returns queried A1 policies along with assigned policy identifiers.
* rApp queries A1 policies using policy identifiers, if identifiers are known to the rApp.
* rApp can query A1 policies created by itself.
* rApp can query A1 policies created by other rApp with SMO/Non-RT RIC framework’s authorization.
* rApp deletes A1 policies created by itself using assigned policy identifiers.
* Querying enforcement status of A1 policies.
* rApp queries enforcement status of A1 policies using policy identifiers, if identifiers are known to the rApp.
* rApp can query enforcement status of A1 policies created by itself.
* rApp can query enforcement status of A1 policies created by other rApp with SMO/Non-RT RIC framework’s authorization.
* Subscription to event notifications related to A1 policies.
* rApp subscribes/unsubscribes notifications for the creation, modification, and deletion of A1 policies created by other rApps. SMO/Non-RT RIC framework notifies rApp of the subscribed events (e.g., a new A1 policy is created, an A1 policy is updated, an A1 policy is removed, etc.), if the rApp is authorized by the SMO/Non-RT RIC framework to receive notifications related to A1 policies created by other rApps.
* Subscription request can be based on a given scope to which A1 policies are applied (e.g., a UE, a group of UEs, a slice, a QoS flow, a cell, a Near-RT RIC, a policy type, etc.).
* Subscription request for A1 policy modification and deletion can be based on policy identifiers, if policy identifiers are known to the rApp.
* rApp subscribes/unsubscribes notifications for changes in enforcement status of A1 policies. SMO/Non-RT RIC framework notifies rApp of the subscribed events. If the subscription request applies to the A1 policies created by other rApps, rApp needs authorization from the SMO/Non-RT RIC framework to receive notifications for enforcement status changes of A1 policies created by other rApps.
* Subscription request can be based on a given scope to which A1 policies are applied (e.g., a UE, a group of UEs, a slice, a QoS flow, a cell, a Near-RT RIC, a policy type, etc.).
* Subscription request can be based on policy identifiers, if identifiers are known to the rApp.

### 7.4.2 A1 enrichment information related services

A1 enrichment information related services produced by the SMO/Non-RT RIC framework enable:

* Registration/deregistration of EI types.
* rApp registers EI types of which data it can produce as the source for the EI job results delivered over the A1 interface.
* rApp deregisters EI types it has registered previously.

NOTE: EI types cannot per definition, see O-RAN A1AP [5], be updated. If a type definition is modified, the rApp needs to register a new EI type and may deregister the previous EI type.

## 7.5 RAN OAM-related Services

RAN OAM-related services produced by the SMO/Non-RT RIC framework enable rApps:

* To obtain information about alarms related to the network elements.
* To obtain performance information related to the network elements, including but not limited to information related to network slicing.
* To obtain the current configuration of the network elements, including but not limited to information related to network slicing.
* To provision changes to the configuration of the network elements, including but not limited to information related to network slicing.
* To obtain trace information related to the network.
* To obtain additional information related to the network elements, including but not limited to information related to network slicing.

Here, the network elements refer to the E2 nodes, O-RU and the Near-RT RIC. The RAN OAM-related services provide access to OAM functionality for (i) the E2 nodes and the Near-RT RIC via the O1 interface, (ii) the O-RUs via the open fronthaul M-plane interface, and (iii) RAN-specific network slicing in the SMO framework.

## 7.6 O2-related services

O2-related services produced by the SMO/Non-RT RIC framework enable rApps:

* To obtain information related to O-Cloud infrastructure management services such as,
* Infrastructure inventory.
* Infrastructure monitoring.
* Infrastructure provisioning.
* Infrastructure lifecycle management.
* Infrastructure software management.
* To obtain information related to O-Cloud deployment management services such as,
* Deployment inventory.
* Deployment monitoring.
* Deployment lifecycle management.
* To provision changes of the configuration of the O-Cloud.
* To obtain additional information related to the O-Cloud.

NOTE: Details about O-Cloud deployment management services, provisioning of configuration changes and additional related information are not specified in the present document.

## 7.7 AI/ML workflow services

The following AI/ML workflow services are defined:

* AI/ML training services: These services allow an AI/ML training service Consumer to request training of an AI/ML model by specifying the training requirements (e.g. required data, model, validation criteria, etc). AI/ML training services can be produced by the SMO/Non-RT RIC framework or by rApps.
* AI/ML model management and exposure services: Services are produced by the SMO/Non-RT RIC framework and enable
* AI/ML model registration/deregistration: It allows an rApp to register, update the registration of, and deregister the AI/ML model that it provides. An rApp needs to be authorised to register, update the registration of, or deregister the AI/ML model it provides.
* AI/ML model discovery: It allows an rApp to discover registered AI/ML model(s) based on its selection criteria if it is provided. An rApp needs to be authorised to discover registered AI/ML models.
* AI/ML model change subscription: It allows an rApp to subscribe and receive notifications of changes of a registered AI/ML model.
* AI/ML model storage: It allows an rApp to request the storage of an AI/ML model it registered and the AI/ML model associated information.
* AI/ML model training capability registration/deregistration (optional service): It allows an AI/ML training service Producer to register/update/deregister its capability of training an AI/ML model.
* AI/ML model training capability query (optional service): It allows an rApp to query registered AI/ML model training capability based on its selection criteria if it is provided. An rApp needs to be authorised to query registered AI/ML model training capability.
* AI/ML model retrieve: It allows an rApp to request model location details of a specific version of a registered AI/ML model.
* AI/ML model performance monitoring services: These services allow an authorised AI/ML performance monitoring service Consumer to request monitoring the performance of a deployed AI/ML model. The AI/ML model performance monitoring service can be produced by the SMO/Non-RT RIC framework or by rApps. The performance information of an AI/ML model is produced by the rApp within which the model is deployed or by AI/ML model inference service Producer performing the model inference.
* AI/ML model inference services: These services allow an rApp to request and or to cancel the inference for a registered AI/ML model. The rApp needs to be authorised to request inference for registered AI/ML models. AI/ML inference services can be produced by the SMO/Non-RT RIC framework or by rApps.

NOTE 1: Void.

NOTE 2: An authorized rApp can request deployment of a specific version of a registered AI/ML model.

## 7.8 rApp management services

The following rApp management services are defined:

* rApp registration service: This service allows rApp to register with the rApp management services Producer.
* rApp configuration service: This service allows rApp configuration settings to be set and read.
* rApp performance service: This service allows an rApp to report on the performance of the rApp itself or its communication with other entities.
* rApp fault service: This service allows an rApp to report faults that are related to the rApp itself or to its communication with other entities.
* rApp log information service: This service allows an rApp to convey the log information that the rApp wishes to report.

## 7.9 TE&IV-related services

TE&IV-related services in SMO/Non-RT RIC framework enable rApps to access topology and inventory information. Relationship information between various TE&IV resource types (e.g., RAN resources and O-Cloud resources) is an example of topology information.

# 8 Non-RT RIC Function Definitions

## 8.1 General description

In the Non-RT RIC architecture in clause 6, R1 services are provided by a collection of logical functions in SMO/Non-RT RIC framework or rApps. These logical functions, as defined in clause 6, include (but are not limited to):

* Service management and exposure functions: These functions produce service management and exposure services.
* Data management and exposure functions: These functions produce data management and exposure services.
* AI/ML workflow functions: These functions produce AI/ML workflow services.
* rApp management functions: These functions produce rApp management services.
* A1-related functions: These functions produce A1-related services.
* RAN OAM-related functions: These functions produce RAN OAM-related services.
* O2-related functions: These functions produce O2-related services.
* R1 termination: It enables the R1 Service Producers and R1 Service Consumers to exchange messages via the R1 interface.
* A1 termination: It enables the Non-RT RIC framework and the Near-RT RIC to exchange messages over the A1 interface.
* O1 termination: It enables the SMO framework to exchange messages with the Near-RT RIC/E2 nodes over the O1 interface.
* Open fronthaul M-plane termination: It enables the SMO framework to exchange messages with the O-RUs over the open fronthaul M-plane interface.
* O2 termination: It enables the SMO framework to exchange messages with the O-Cloud over the O2 interface.
* Other SMO functions: Among other functionalities, these functions also offer RAN-specific slice management functionality.
* External terminations: These functions enable the SMO/Non-RT RIC framework to exchange messages with external entities over interfaces outside the scope of O-RAN.

NOTE: The SMO functions offering the slice management services and whether these shall be considered as R1 services is for future study.

## 8.2 Service management and exposure functions

### 8.2.1 Service registration and discovery functions

The service registration and discovery functions are logical functions that produce the service registration, service deregistration, service discovery, service notification, communication support services and optional bootstrap and heartbeat services specified in clause 7.2.

Examples of Service Producers include: service management and exposure services producer, A1-related services producer, etc. Examples of Service Consumers are rApps. It is noted that rApps can also be Service Producers and logical functions can also be Service Consumers.

A Service Producer or an entity acting on its behalf can send a service registration request to the service registration and discovery functions to register services. A Service Consumer can either send a service discovery request to the service registration and discovery functions to discover the services and the endpoints on which they can be consumed, or it can send a service request directly if it receives a service notification with the related information on the service and on the endpoints on which it can be consumed.

### 8.2.2 Authentication and authorisation functions

The authentication and authorisation functions are logical functions that produce the authentication and authorisation services specified in clause 7.2.

## 8.3 Data management and exposure functions

The data management and exposure functions are logical functions that produce the data registration and discovery service, the data request and subscription service, the data offer service and optional data processing services specified in clause 7.3. They further initiate data collection by consuming the data request and subscription service. Additionally, they produce and consume the data delivery services in order to obtain collected data from Data Producers and to allow Data Consumers to obtain data from the data management and exposure functions for consumption.

The data management and exposure functions track DME types that are registered. They match data discovery requests against such DME types. If the DME type is registered, the data management and exposure functions identify a valid Data Producer, which can be a previously registered Data Producer rApp or other Data Producers in the SMO/Non-RT RIC framework (e.g., RAN OAM-related functions). The data management and exposure functions optionally provide data processing functionality on collected data (e.g., quantization, normalization, correlation, labelling, etc.).

When Data Consumers use the subscribe data procedure to initiate continuous data consumption, the data management and exposure functions check the availability of the subscribed data after receiving a data subscription request from a Data Consumer. If the subscribed data are already being collected, the data management and exposure functions deliver the data to the Data Consumer when they become available. If the data are not already being collected and a valid Data Producer is identified, the data management and exposure functions initiate data collection to obtain data from the identified Data Producer. When the collected data become available, the data management and exposure functions deliver the data to the Data Consumer.

When Data Consumers use the request data procedure to initiate one-time data consumption, the data management and exposure functions check the availability of the required data after receiving a one-time data request from a Data Consumer. If the requested data are available, the data management and exposure functions deliver the data to the Data Consumer. If the data are not available and a valid Data Producer is identified, the data management and exposure functions initiate data collection to obtain data from the identified Data Producer. When the collected data become available, the data management and exposure functions deliver the data to the Data Consumer.

When a Data Producer uses the create data offer procedure to trigger data collection, the data management and exposure functions set up an endpoint for receiving data availability notifications or push data messages (depending on the delivery method chosen) and confirm the creation of the data offer to the Data Producer. Data collection starts once the data offer creation is confirmed and proceeds until the data offer is terminated by the Data Producer or by the data management and exposure functions. The data management and exposure functions store the collected data for later consumption.

The use of a data repository for storing data during data collection is optional if data collection was triggered by a data request or data subscription from a Data Consumer. It is required if data collection was triggered by a data offer from a Data Producer. Data can be optionally pre-processed before being stored and can be optionally post-processed after being read.

NOTE: Instead of storing data in a data repository, data management and exposure functions can broker data from Data Producers to Data Consumers.

## 8.4 A1-related functions

### 8.4.1 A1 policy functions

The logical functions that produce the A1 policy management services which are exposed to rApps via the R1 interface are called the “A1 policy functions". They realize the following functionalities:

* Producing A1 policy management services which are exposed to rApps via the R1 interface.
* The corresponding service definition can be found in clause 7.4.1.
* Performing A1 policy conflict mitigation.
* If more than one policy generation rApps set contradicting A1 policy statements to the same or overlapping policy scope identifiers, A1 policy function decides to reject the creation/modification request of one or more A1 policies or to temporarily suspend the enforcement of one or more A1 policies.
* Interfacing with Near-RT RICs through A1 termination
* A1 policy function determines the Near-RT RIC for generated A1 policies.
* A1 policy function sends A1 policy creation, update, and deletion requests to the Near-RT RIC.
* A1 policy function queries supported A1 policy types of connected Near-RT RICs.
* A1 policy function queries enforcement status of A1 policies.
* A1 policy function receives A1 policy feedback from the Near-RT RIC.

### 8.4.2 A1 enrichment information functions

“A1 enrichment information functions” produce the A1 enrichment information related services which are exposed to rApps via the R1 interface. The corresponding services definition can be found in clause 7.4.2. “A1 enrichment information functions” also produce the A1-EI service which is exposed to Near-RT RICs via the A1 interface. The corresponding service definition can be found in the A1 interface specification, see references in clause 9.1. “A1 enrichment information functions” collect data that are to be delivered as EI job results to Near-RT RICs via the data management and exposure functions.

NOTE: The data delivered as EI job results can be produced by rApps or by external enrichment information sources.

“A1 enrichment information functions” realize the following functionalities:

* Handling of EI types registered by rApps via the R1 interface.
* Handling of EI jobs created by the Near-RT RICs via the A1 interface.
* Collecting data that are to be delivered as EI job results to the Near-RT RICs via the A1 interface.
* Interfacing with Near-RT RICs through A1 termination.

## 8.5 RAN OAM-related functions

The logical functions that produce the RAN OAM-related services which are exposed to rApps via the R1 interface are called the “RAN OAM-related functions”. They realize the following functionalities:

* Producing RAN OAM-related services which are exposed to rApps via the R1 interface.
* The corresponding service definition can be found in clause 7.5.
* Performing CM conflict mitigation.
* Interfacing with Near-RT RICs and E2 nodes through O1 termination.
* RAN OAM-related functions receive fault notifications and obtain alarm list from Near-RT RICs and E2 nodes.
* RAN OAM-related functions provision configuration changes to Near-RT RICs and E2 nodes.
* RAN OAM-related functions collect performance data from Near-RT RICs and E2 nodes.
* RAN OAM-related functions collect trace data from E2 nodes.
* Interfacing with the O-RUs through open fronthaul M-plane termination.
* RAN OAM-related functions receive fault notifications and obtain alarm list from the O-RUs.
* RAN OAM-related functions provision configuration changes to the O-RUs.
* RAN OAM-related functions collect performance data from the O-RUs.
* Interfacing with the RAN-specific slice management functionality in the SMO framework.
* RAN OAM-related functions provision configuration changes, related to RAN-specific network slicing.
* RAN OAM-related functions collect performance data, related to RAN-specific network slicing.

## 8.6 O2-related functions

The O2 interface is an open logical interface within O-RAN architecture. The logical functions that produce the O2-related services which are exposed to rApps via the R1 interface are called the “O2-related functions”. O2-related functions include FOCOM and NFO. They realize the following functionalities:

* Producing O2-related services which are exposed to rApps via the R1 interface.
* The corresponding service definition can be found in clause 7.6.
* Interfacing with O-Cloud through O2 termination.
* O2-related functions receive O-Cloud infrastructure FCAPS (PM, CM, FM) data via O-Cloud IMS, see O-RAN O2GAP [13].
* O2-related functions receive O-Cloud deployment FCAPS (PM, CM, FM) data via O-Cloud DMS, see O-RAN O2GAP [13].
* O2-related functions provision configuration changes to O-Cloud.
* Performing conflict mitigation for O-Cloud configuration changes.

## 8.7 AI/ML workflow functions

The AI/ML workflow functions are logical functions that produce the following AI/ML workflow services as specified in clause 7.7:

* AI/ML training services.
* AI/ML model management and exposure services, which enables AI/ML model registration/deregistration, AI/ML model discovery, AI/ML model change subscription, AI/ML model storage and optionally AI/ML model training capability registration/deregistration/query.
* AI/ML model performance monitoring services.
* AI/ML model inference services.

# 9 External Interfaces

## 9.1 A1 Interface

O-RAN A1GAP [4] specifies A1 interface general aspects and principles.

O-RAN A1AP [5] specifies A1 interface application protocols.

O-RAN A1TP [6] specifies A1 interface transport protocols.

O-RAN A1TD [7] specifies data model and data types used in the A1 interface.

## 9.2 Void

# 10 Procedures

## 10.1 Data management and exposure procedures

### 10.1.0 Overview

This clause demonstrates example procedures related to data management and exposure (DME) services produced and consumed by SMO/Non-RT RIC framework. A Data Consumer rApp requests/subscribes data from SMO/Non-RT RIC framework, and the requested/subscribed data are collected from another Data Producer rApp or other O-RAN entities.

NOTE: It is not required to use the same type of DME services procedure in data consumption and data collection.

Data consumption denotes the procedures of Data Consumer rApp requesting/subscribing data from SMO/Non-RT RIC framework. Data collection denotes the procedures of SMO/Non-RT RIC framework requesting/subscribing data from Data Producer rApps or other O-RAN entities.

### 10.1.1 Subscribe data procedures in data consumption and collection

Figure 10.1.1-1 illustrates an example workflow of using DME services to subscribe data and deliver data over the R1 interface. In this example, it is assumed that

* Two Data Consumer rApps subscribe the same data instance (i.e., same DME type and scope) from the DME functions.
* The requested data are produced by a Data Producer rApp.
* The DME functions subscribe data instance from the Data Producer rApp.
* Data are delivered using pull or push data delivery or by point to multipoint streaming data delivery. In case of pull delivery, the pull delivery details can be contained in subscribe data response or data availability notification. The Data Consumer makes a pull request based on the received pull delivery details. In case of point to multipoint streaming data delivery, DME functions inform the Data Consumer which data stream endpoint to consume the subscribed data from.

When the first Data Consumer rApp initiates the Subscribe data procedure to obtain data, the DME functions first identify the Data Producer for the requested data instance. In this example, one Data Producer rApp which is already registered as being capable of producing the subscribed data are identified as the Data Producer. The DME functions figure out there is no data subscription for the requested data instance yet, and the DME functions subscribe data from Data Producer rApp. In case of pull delivery, the Data Producer rApp notifies the data availability to the DME functions and provides pull data details in the notification. The DME functions then pull data from the Data Producer rApp. In case of push delivery, the Data Producer rApp pushes the data to the DME functions. The DME functions may optionally process the data or store the data in the data repository for future usage. In case of pull delivery, the DME functions further notify the data availability to the first Data Consumer rApp. The rApp then pulls the data from the DME functions. In case of push delivery, the DME functions push the data to the Data Consumer rApp. In case of point to multipoint streaming data delivery, the DME functions configures the data stream endpoint unless it is already configured. The Data Producer rApp writes the data to the stream endpoint and Data Consumer rApp reads the data from the stream endpoint.

When the second Data Consumer rApp sends a data subscription request to the DME functions, the DME functions find out that the data subscription for the requested data instance is already established. Therefore, the DME functions do not set up additional data subscription with the Data Producer rApp. Once new data are available, the following steps are executed: In case of pull delivery, the Data Producer notifies the data availability to the DME functions, and these pull the data from the Data Producer. In case of push delivery, the Data Producer pushes the data to the DME functions. The DME functions may optionally process the data and store the data in the data repository for future usage. In case of pull delivery, the DME functions further notify the data availability to both Data Consumer rApps which then pull the data from the DME functions. In case of push delivery, the DME functions push the data to both Data Consumer rApps. In case of point to multipoint streaming data delivery, the DME functions inform the second Data Consumer rApp which data stream endpoint to use. The Data Producer rApp writes the data to the data stream endpoint and the Data Consumer rApp reads the data from the data stream endpoint.

The DME functions can use different procedures to deliver data to Data Consumer rApps even if both Data Consumer rApps subscribe the same data instance. For example, the DME functions can notify data availability to the one Data Consumer rApp allowing the rApp to pull data, meanwhile the DME functions push data to another Data Consumer or both the Data Consumer rApps read the data from the data stream endpoint provided by the DME functions.

NOTE 1: Push content can be (1) notifications that carry a data payload or (2) streams of data.

NOTE 2: Notifications, pull data delivery, and push data delivery may require the set-up of a dedicated communication channel that allows the content to be pulled/pushed. Such dedicated communication channels may map to, e.g., Kafka topic, WebSocket connection, gRPC connection, etc. Alternatively, individual HTTP requests may be used as push messages.

@startuml

!pragma teoz true

skinparam ParticipantPadding 5

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam lifelineStrategy solid

skinparam SequenceGroupBackgroundColor Transparent

skinparam SequenceGroupBodyBackgroundColor Transparent

autonumber

box "Non-RT RIC" #whitesmoke

box #ivory

participant "Data Consumer rApp #1" as app1

participant "Data Consumer rApp #2" as app2

endbox

box "Non-anchored functions in SMO/Non-RT RIC FWK" #cadetBlue

participant "Data Management and Exposure functions" as dme

endbox

box #ivory

participant "Data Producer rApp" as src

endbox

endbox

group Data Consumer rApp #1 subscribes data

app1 -> dme : <<R1>> Subscribe data request

alt Pull or push data delivery

app1 <- dme : <<R1>> Subscribe data response

else Point to multipoint streaming data delivery

dme --> dme : Configure data stream endpoint

app1 <- dme : <<R1>> Subscribe data response (data stream endpoint)

end

end

dme --> dme : Find out that data instance\nis not being collected;\nidentify the Data Producer

group DME functions subscribes data from Data Producer rApp

alt Pull or push data delivery

src <- dme : <<R1>> Subscribe data request

else Point to multipoint streaming data delivery

src <- dme: <<R1>> Subscribe data request (data stream endpoint)

end

src -> dme : <<R1>> Subscribe data response

end

loop data available

alt Pull delivery

src -> dme : <<R1>> Notify data availability (pull delivery details)

src <- dme : <<R1>> Pull data request

src -> dme : <<R1>> Pull data response (with data payload)

else Push delivery

src -> dme : <<R1>> Push data message (with data payload)

end

opt

dme --> dme : Processing data

end

opt

dme --> dme: Storing data

end

alt Pull delivery

app1 <- dme : <<R1>> Notify data availability (pull delivery details)

app1 -> dme : <<R1>> Pull data request

app1 <- dme : <<R1>> Pull data response (with data payload)

else Push delivery

app1 <- dme : <<R1>> Push data message (with data payload)

end

end

group Data Consumer rApp #2 subscribes data

app2 -> dme : <<R1>> Subscribe data request

alt Pull or push data delivery

app2 <- dme : <<R1>> Subscribe data response

else Point to multipoint streaming data delivery

dme <-- dme : Configure data stream endpoint

app2 <- dme : <<R1>> Subscribe data response (data stream endpoint)

end

end

dme --> dme : Find out that data instance\nis already being collected

loop data available

alt Pull delivery

src -> dme : <<R1>> Notify data availability (pull delivery details)

src <- dme : <<R1>> Pull data request

src -> dme : <<R1>> Pull data response (with data payload)

else Push delivery

src -> dme : <<R1>> Push data message (with data payload)

end

opt

dme --> dme : Processing data

end

opt

dme --> dme: Storing data

end

alt Pull delivery

app1 <- dme : <<R1>> Notify delivery details (pull delivery details)

app1 -> dme : <<R1>> Pull data request

app1 <- dme : <<R1>> Pull data response (with data payload)

app2 <- dme : <<R1>> Notify data availability (pull delivery details)

app2 -> dme : <<R1>> Pull data request

app2 <- dme : <<R1>> Pull data response (with data payload)

else Push delivery

app1 <- dme : <<R1>> Push data message (with data payload)

app2 <- dme : <<R1>> Push data message (with data payload)

end

end

@enduml

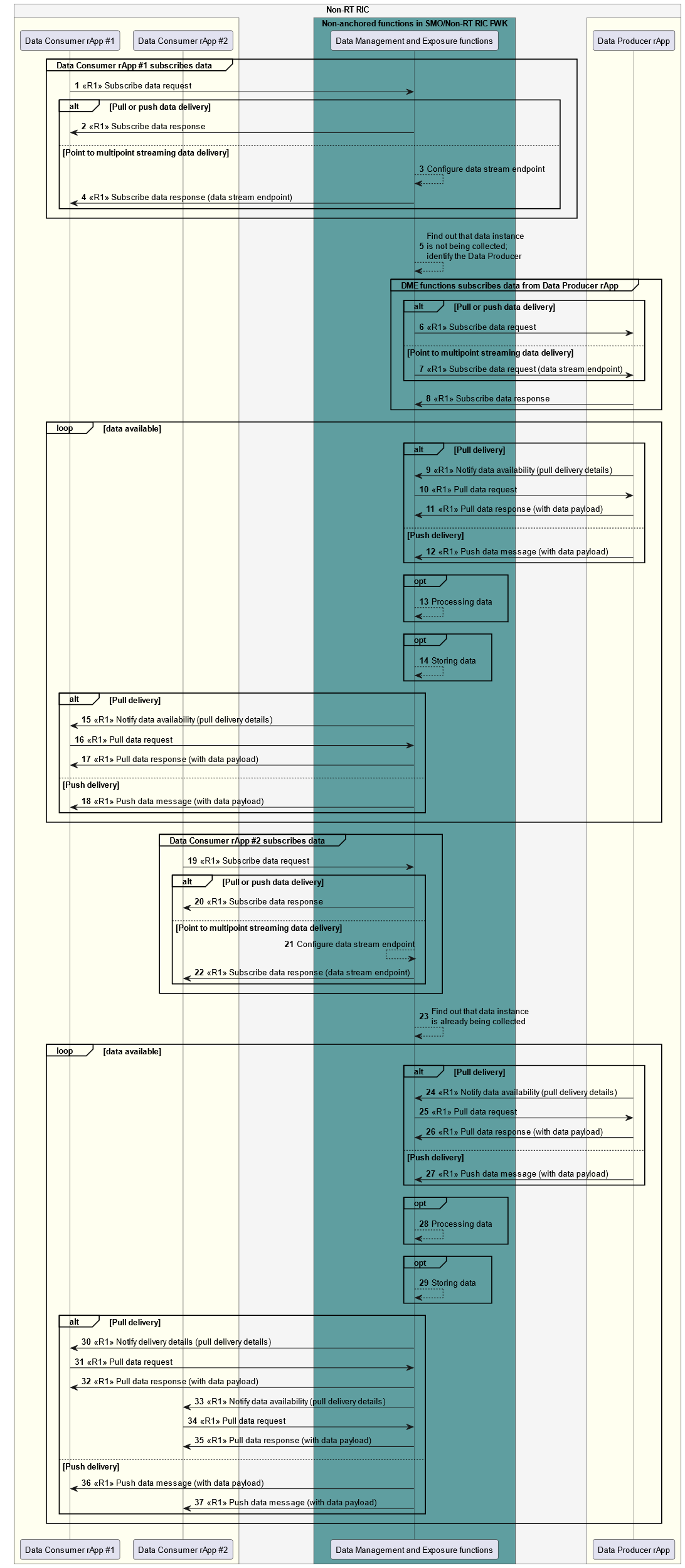


Figure 10.1.1-1: Example workflow using subscribe data and notify data availability procedures

Figure 10.1.1-2 illustrates an example workflow of using unsubscribe data procedures to terminate data subscription in data consumption data and data collection. In the example, it is assumed that the two Data Consumer rApps in the figure are the only two Data Consumers which have subscribed the data instance produced by the Data Producer rApp.

When the second Data Consumer rApp initiates the unsubscribe data procedure, the DME functions find out that the produced data are also subscribed by the first Data Consumer rApp. Therefore, the DME functions can't unsubscribe from the Data Producer rApp.

When the first Data Consumer rApp initiates the unsubscribe data procedure, the DME functions figure out that the data subscription is no longer needed. Hence, in this example, the DME functions identify the Data Producer rApp and send an unsubscription request to cancel the data subscription. In case of point to multipoint streaming data delivery, the Data Producer rApp stops producing the data to the data stream endpoint.

NOTE 3: The DME functions are not required to unsubscribe data from Data Producers even if there are no Data Consumers requesting or subscribing these data. The DME functions may collect data from Data Producers for its own purposes.

@startuml

!pragma teoz true

skinparam ParticipantPadding 5

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam lifelineStrategy solid

skinparam SequenceGroupBackgroundColor Transparent

skinparam SequenceGroupBodyBackgroundColor Transparent

autonumber

box "Non-RT RIC" #whitesmoke

box #ivory

participant "Data Consumer rApp #1" as app1

participant "Data Consumer rApp #2" as app2

endbox

box "Non-anchored functions in SMO/Non-RT RIC FWK" #cadetBlue

participant "Data Management and Exposure functions" as dme

endbox

box #ivory

participant "Data Producer rApp" as src

endbox

endbox

group Data Consumer rApp #2 unsubscribes data

app2 -> dme : <<R1>> Unsubscribe data request

app2 <- dme : <<R1>> Unsubscribe data response

end

dme --> dme : Find out that data are subscribed \n by other Data Consumers

group Data Consumer rApp #1 unsubscribes data

app1 -> dme : <<R1>> Unsubscribe data request

app1 <- dme : <<R1>> Unsubscribe data response

end

dme --> dme : Find out that data subscription \n is no longer required;\nidentify the Data Producer

opt DME unsubscribes data from Data Producer rApp

src <- dme : <<R1>> Unsubscribe data request

src -> dme : <<R1>> Unsubscribe data response

end

@enduml

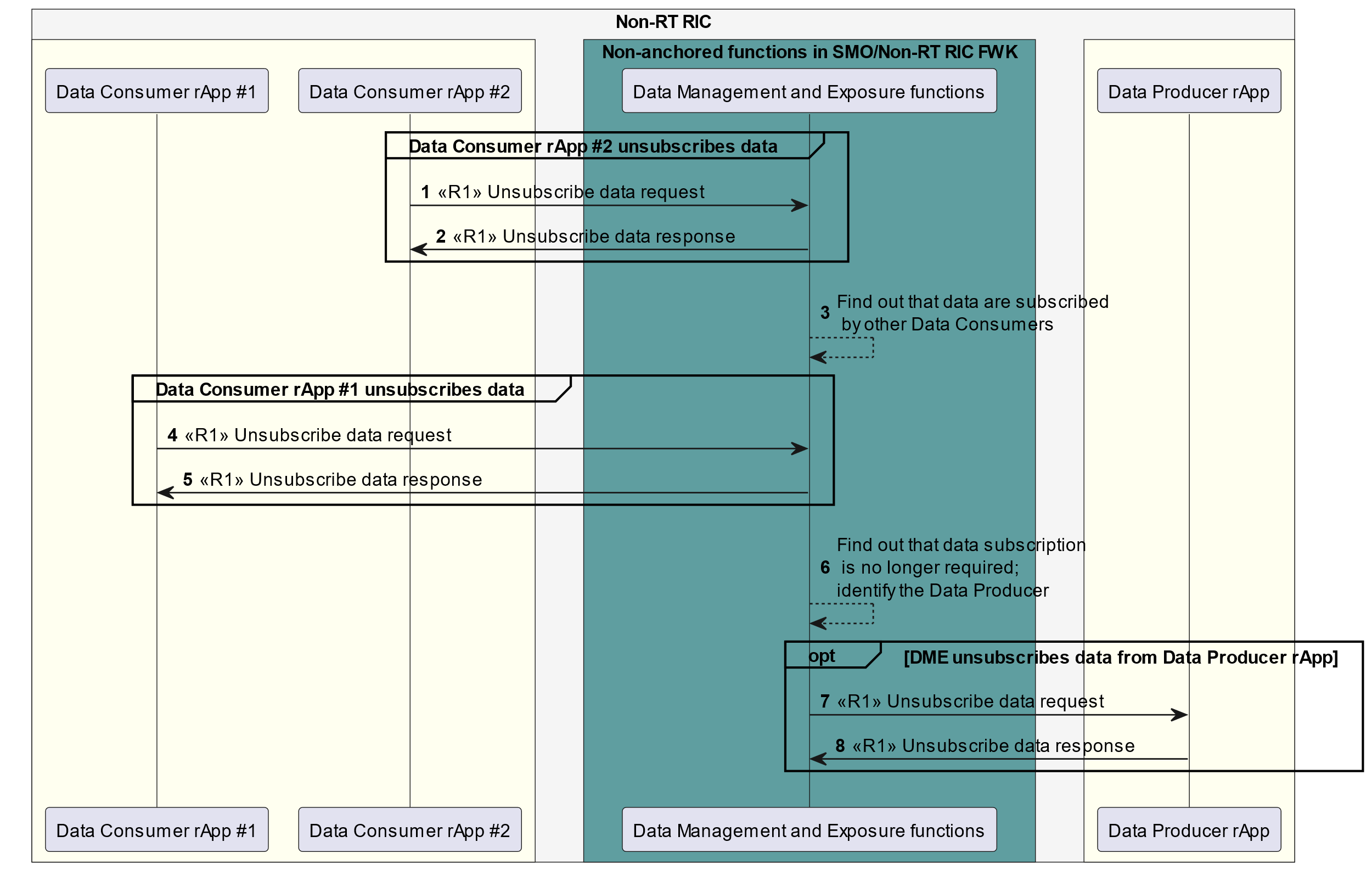


Figure 10.1.1-2: Example workflow using unsubscribe data procedure

### 10.1.2 Request data procedure in data consumption and data collection

Figure 10.1.2-1 illustrates an example workflow of using DME services to request and deliver data over the R1 interface. In this example, it is assumed that

* Two Data Consumer rApps request the same data instance (i.e., same DME type and scope) using request data procedure.
* The requested data are produced by a Data Producer rApp.
* The DME functions request data from the Data Producer rApp using request data procedure.
* Data are delivered using pull delivery. The pull delivery details are contained in the request data response. The Data Consumer makes a pull request based on the received pull delivery details.
* The pull delivery details provide Data Consumer information of how to formulate a pull data request for the delivery of requested data (e.g., a URI from which to fetch the data).

When the first Data Consumer rApp initiates the request data procedure to obtain data, the DME functions first check whether the requested data are available in the data repository. In this example, it figures out that data are not available. Then the DME functions identify the Data Producer rApp as the Data Producer, e.g., based on DME type registration records. The DME functions send data request to the Data Producer rApp, and the data are delivered from the Data Producer rApp using pull delivery. The DME functions then send data request response to the first Data Consumer rApp carrying pull delivery details. The first Data Consumer rApp pulls data from the DME functions. The DME functions may optionally process data and store the data in the data repository for future usage.

When the second Data Consumer rApp sends data request to the DME functions, the DME functions notice that the requested data are already available in the data repository. Then the DME functions retrieve requested data from data repository and responds the second Data Consumer rApp with pull delivery details. The data are pulled by the second Data Consumer rApp.

@startuml

!pragma teoz true

skinparam ParticipantPadding 5

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam lifelineStrategy solid

skinparam SequenceGroupBackgroundColor Transparent

skinparam SequenceGroupBodyBackgroundColor Transparent

autonumber

box "Non-RT RIC" #whitesmoke

box #ivory

participant "Data Consumer rApp #1" as app1

participant "Data Consumer rApp #2" as app2

endbox

box "Non-anchored functions in SMO/Non-RT RIC FWK" #cadetBlue

participant "Data Management and Exposure functions" as dme

database "Data repository" as db

endbox

box #ivory

participant "Data Producer rApp" as src

endbox

endbox

group Data Consumer rApp #1 requests data

app1 -> dme : <<R1>> Request data request

dme --> dme : Find out that requested data \n are not available in the data repository

app1 <- dme : <<R1>> Request data response (pull delivery details)

group Obtain data not available in the repository

src <- dme : <<R1>> Request data request

src -> dme : <<R1>> Request data response (pull delivery details)

src <- dme : <<R1>> Pull data request

src -> dme : <<R1>> Pull data response (with data payload)

opt

dme --> dme : Processing data

end

opt

dme <--> db: Storing data

end

end

app1 -> dme : <<R1>> Pull data request

app1 <- dme : <<R1>> Pull data response (with data payload)

end

group Data Consumer rApp #2 requests data

app2 -> dme : <<R1>> Request data request

dme --> dme : Find out that requested data \n are available in the data repository

app2 <- dme : <<R1>> Request data response (pull delivery details)

group Obtain data that are available in the repository

dme <--> db: Retrieving data

end

app2 -> dme : <<R1>> Pull data request

app2 <- dme : <<R1>> Pull data response (with data payload)

end

@enduml

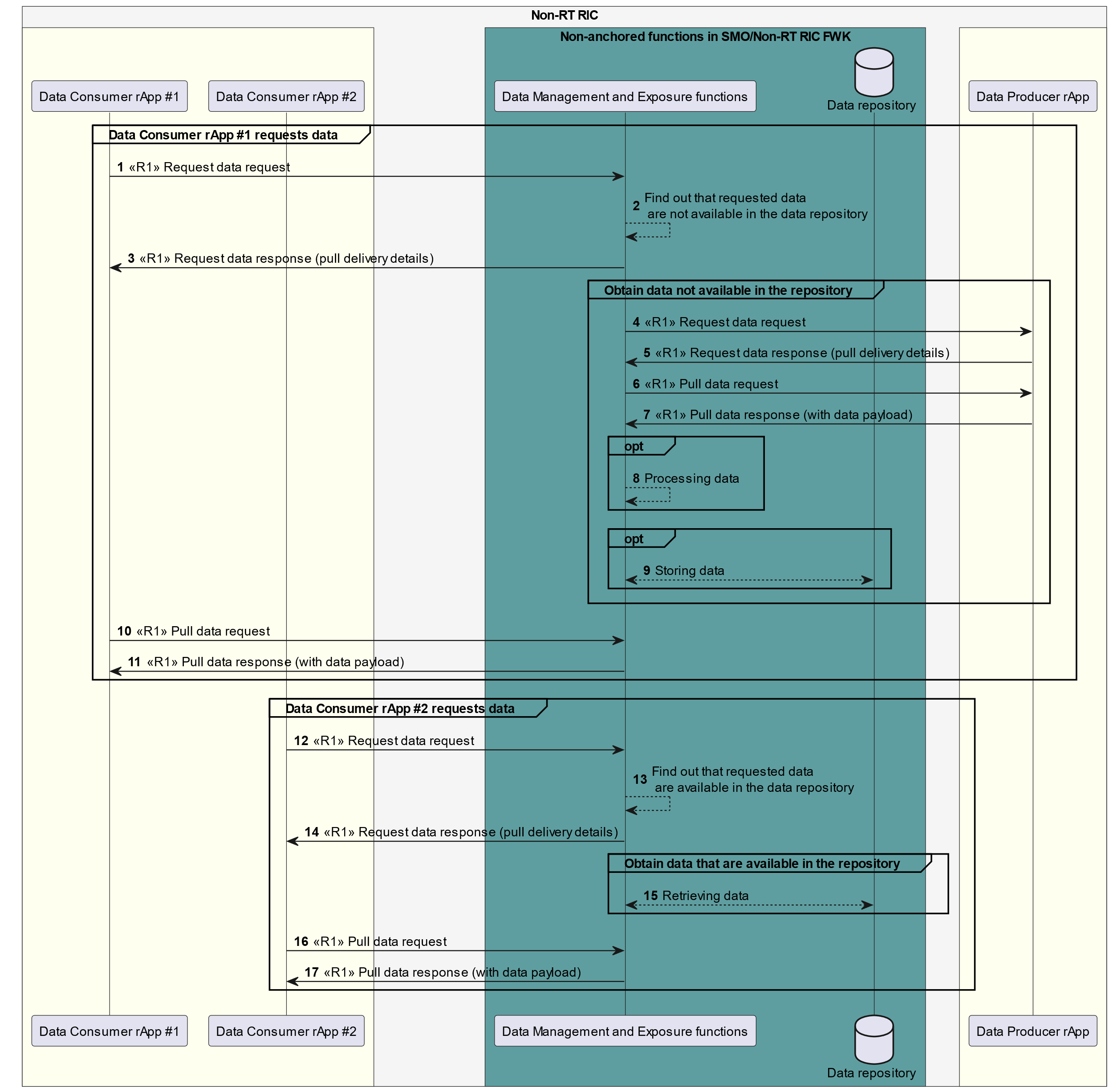


Figure 10.1.2-1: Example workflow using request data procedure

### 10.1.3 O1 related data collection procedures based on data subscription

Figure 10.1.3-1 illustrates an example workflow of how DME functions in the SMO/Non-RT RIC framework collect O1-related data based on the data subscription from a Data Consumer rApp.

In this example, when the Data Consumer rApp initiates the subscribe data procedure to obtain O1 related data, the DME functions acknowledge the data subscription, and check whether subscribed data are being collected already or not. If the subscribed data are not being collected, the DME functions initiates data collection procedure to collect from nodes terminating the O1 interface via the RAN OAM-related functions. The related procedures have been specified in O-RAN OAM Arch [8], clause 4.2.2 “O-RAN Measurement Data Collection Use Case”. When the DME functions receive the collected data from the RAN OAM-related functions, they notify the Data Consumer rApp, and the subscribed data are delivered using pull or push data services.

NOTE: Data delivery procedures are defined in O-RAN R1GAP [9], clause 5.2.6.

@startuml

!pragma teoz true

skinparam ParticipantPadding 5

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam lifelineStrategy solid

skinparam SequenceGroupBackgroundColor Transparent

skinparam SequenceGroupBodyBackgroundColor Transparent

skinparam sequenceReferenceHeaderBackgroundColor Transparent

autonumber

box "Non-RT RIC" #whitesmoke

box #ivory

participant "Data Consumer rApp" as dc

endbox

box "Non-anchored functions in SMO/Non-RT RIC FWK" #cadetBlue

participant "Data Management and Exposure functions" as dme

participant " RAN OAM-related functions" as OAM

endbox

endbox

participant “Nodes terminating O1 interface” as dp

autonumber

dc -> dme : <<R1>> Subscribe data request

dme ->dc : <<R1>> Subscribe data response

dme --> dme : check if the subscribed data are being collected already

group if subscribed data are not being collected yet

dme <--> OAM : Implementation specific communication

ref over OAM, dp

As specified in Clause 4.2.2.3.1

“Measurement Data Collection Creation”

in O-RAN.WG10.OAM-Architecture

end

end

loop data available

alt File-based delivery

ref over OAM, dp

As specified in Clause 4.2.2.3.2

“Measurement Data File Consumption”

in O-RAN.WG10.OAM-Architecture

end

else Streaming delivery

ref over OAM, dp

As specified in Clause 4.2.2.3.3

“Measurement Streaming Data Consumption”

in O-RAN.WG10.OAM-Architecture

end

end

OAM --> dme : implementation-specific data delivery

alt pull delivery

dme -> dc: <<R1>> Notify data availability (pull delivery details)

dc -> dme: <<R1>> Pull data request

dme -> dc: <<R1>> Pull data response (data payload)

else push delivery

dme -> dc : <<R1>>Push data message (data payload)

end

end

@enduml

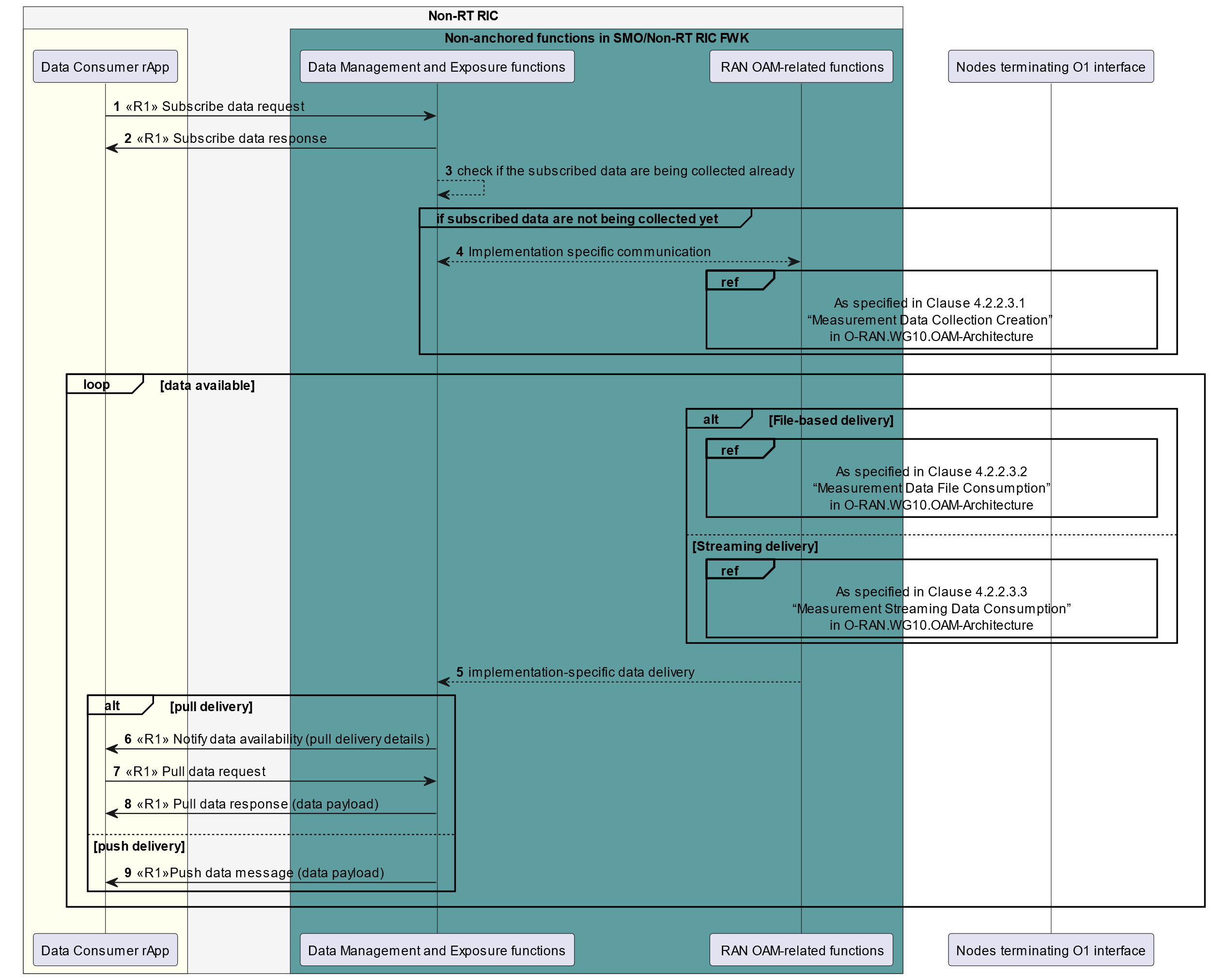


Figure 10.1.3-1: Example workflow for O1 related data collection procedure via DME

### 10.1.4 O2 related data collection procedures based on data subscription

Figure 10.1.4-1 illustrates an example workflow of how DME functions in the SMO/Non-RT RIC framework collect O2 related data based on the data subscription from a Data Consumer rApp.

In this example, when the Data Consumer rApp initiates the subscribe data procedure to obtain O2 related data, the DME functions acknowledge the data subscription, and check whether subscribed data are being collected already or not. If the subscribed data are not being collected, the DME functions initiate Data collection procedure to collect from DMS via the O2 related functions. This procedure has been specified in O-RAN OAM Arch [8], clause 4.2.2 “O-RAN Measurement Data Collection Use Case”. When the DME functions receive the collected data from the O2 related functions, it notifies the Data Consumer rApp, and the subscribed data are delivered using pull or push data services.

NOTE: Data delivery procedures are defined in O-RAN R1GAP [9], clause 5.2.6.

@startuml

skinparam ParticipantPadding 5

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam lifelineStrategy solid

skinparam SequenceGroupBackgroundColor Transparent

skinparam SequenceGroupBodyBackgroundColor Transparent

skinparam sequenceReferenceHeaderBackgroundColor Transparent

autonumber

box "Non-RT RIC" #whitesmoke

box #ivory

participant "Data Consumer rApp" as dc

endbox

box "Non-anchored functions in SMO/Non-RT RIC FWK" #cadetBlue

participant "Data Management and Exposure functions" as dme

participant "O2 related functions" as OAM

endbox

endbox

participant “DMS” as dp

autonumber

dc -> dme : <<R1>> Subscribe data request

dme ->dc : <<R1>> Subscribe data response

dme --> dme : Check if the subscribed data are being collected already

group if subscribed data are not being collected yet

dme <--> OAM : Implementation specific communication

ref over OAM, dp

As specified in Clause 4.2.2.3.1

“Measurement Data Collection Creation”

in O-RAN.WG10.OAM-Architecture

End

End

loop data available

alt File-based delivery

ref over OAM, dp

As specified in Clause 4.2.2.3.2

“Measurement Data File Consumption”

in O-RAN.WG10.OAM-Architecture

end

else Streaming delivery

ref over OAM, dp

As specified in Clause 4.2.2.3.3

“Measurement Streaming Data Consumption”

in O-RAN.WG10.OAM-Architecture

end

end

OAM --> dme : Implementation-specific data delivery

alt pull delivery

dme -> dc: <<R1>> Notify data availability (pull delivery details)

dc -> dme: <<R1>> Pull data request

dme -> dc: <<R1>> Pull data response (data payload)

else push delivery

dme -> dc : <<R1>> Push data message (data payload)

end

end

@enduml

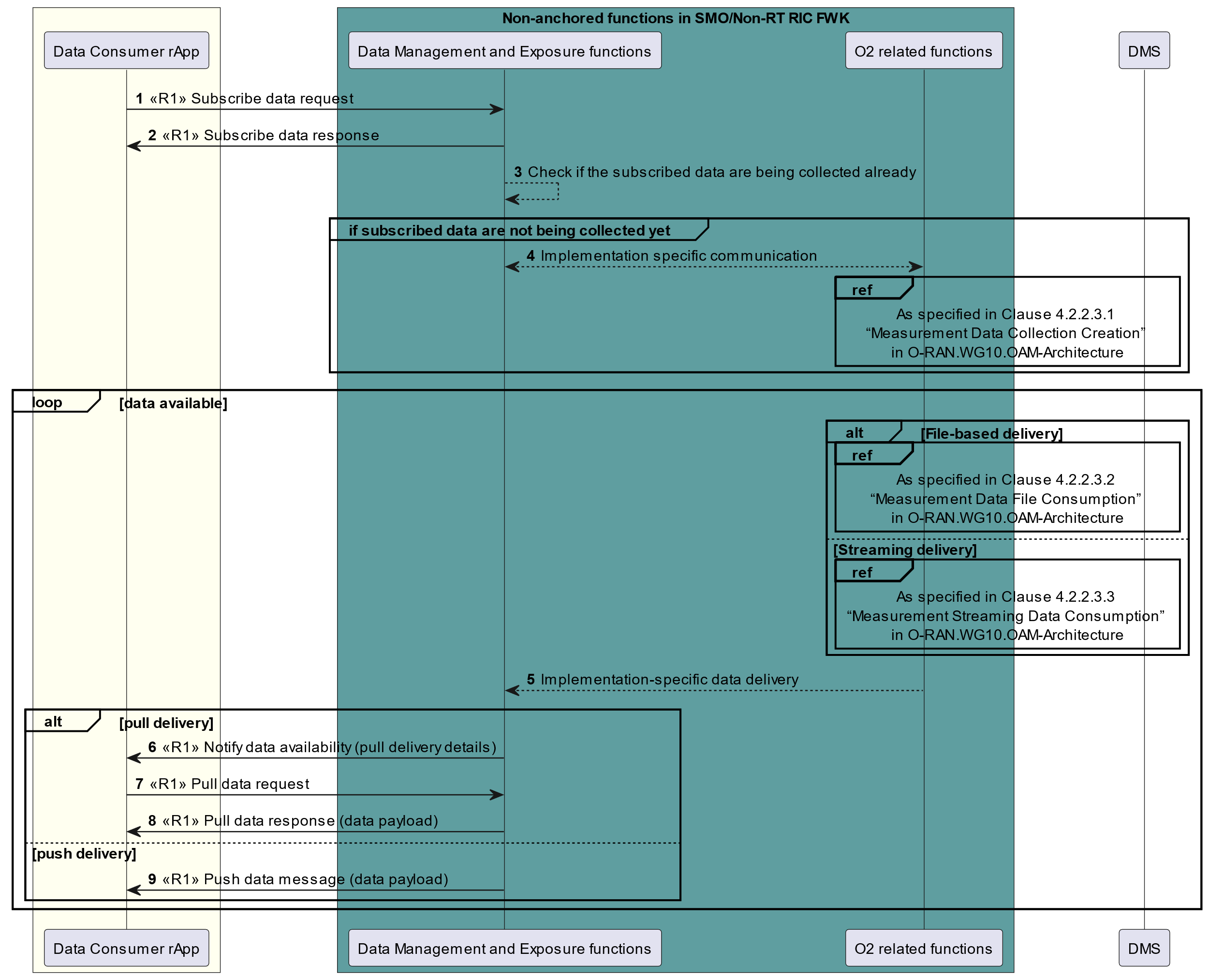


Figure 10.1.4-1: Example workflow for O2 related data collection procedure via DME

### 10.1.5 Data offer procedures

Figure 10.1.5-1 illustrates an example workflow of using the DME services by a Data Producer to trigger the DME functions to collect a data instance produced by that Data Producer and to store it for later consumption.

The Data Producer rApp first needs to register a DME type for the data instance it intends to produce. The rApp then initiates the create data offer procedure to trigger the collection of a data instance of that DME type which it intends to produce. The parameters of the data instance and the supported delivery services are indicated in the request. After receiving the request, the DME functions first check whether the Data Producer has registered the DME type for which it intends to produce an instance. Upon success, the DME functions set up an endpoint for the reception of data availability notifications or push data messages, depending on the delivery service chosen, and communicate this information to the Data Producer. Data collection can start immediately after the data offer creation has been acknowledged by the DME functions.

When data are available, the Data Producer sends a data availability notification to the DME functions and these pull the data, or the Data Producer pushes the data to the DME functions. Then, the DME functions store these data for subsequent consumption.

When the Data Producer intends to stop data production, it terminates the data offer by using the terminate data offer procedure. Data collection ends upon successful execution of that procedure. When the data management and exposure functions intend to stop receiving produced data, they send a data offer termination notification to the Data Producer. Subsequently, possibly after a delay to allow the Data Producer to react to the notification, the data management and exposure functions stop data collection and terminate the data offer. Upon receiving this notification, the Data Producer stops data production.

@startuml

!pragma teoz true

skinparam ParticipantPadding 5

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam SequenceGroupBackgroundColor Transparent

skinparam SequenceGroupBodyBackgroundColor Transparent

skinparam lifelineStrategy solid

autonumber

box "Non-RT RIC" #whitesmoke

box #ivory

participant "Data Producer rApp" as rapp

endbox

box "Non-anchored functions in SMO/Non-RT RIC FWK" #cadetBlue

participant "Data Management and\nExposure functions" as dme

database "Data repository" as db

endbox

endbox

group Data Producer rApp registers DME type

rapp -> dme: <<R1>> Register DME type request

rapp <- dme: <<R1>> Register DME type response

end

group Data Producer rApp creates data offer

rapp -> dme: <<R1>> Create data offer request(info about data instance and delivery)

dme --> dme: Create data offer\nand prepare\ndelivery endpoint

dme -> rapp: <<R1>> Create data offer response (endpoint details for notify data availability\nor push data)

end

group Data are collected from Data Producer rApp

loop data available

alt pull delivery

rapp -> dme: <<R1>> Notify data availability (pull details)

dme -> rapp: <<R1>> Pull data request

rapp -> dme: <<R1>> Pull data response (data)

else push delivery

rapp -> dme: <<R1>> Push data (data)

end

dme <--> db: Store data

end

end

group Data offer termination

alt Data Producer rApp terminates data offer

rapp -> dme: <<R1>> Terminate data offer request

dme --> dme: Delete data offer and\nstop collecting data

dme -> rapp: <<R1>> Terminate data offer response

else DME functions terminate data offer

dme -> rapp: <<R1>> Notify data offer termination

dme --> dme: Stop collecting data\nand terminate data offer

rapp --> rapp: Stop producing data

end

end

@enduml

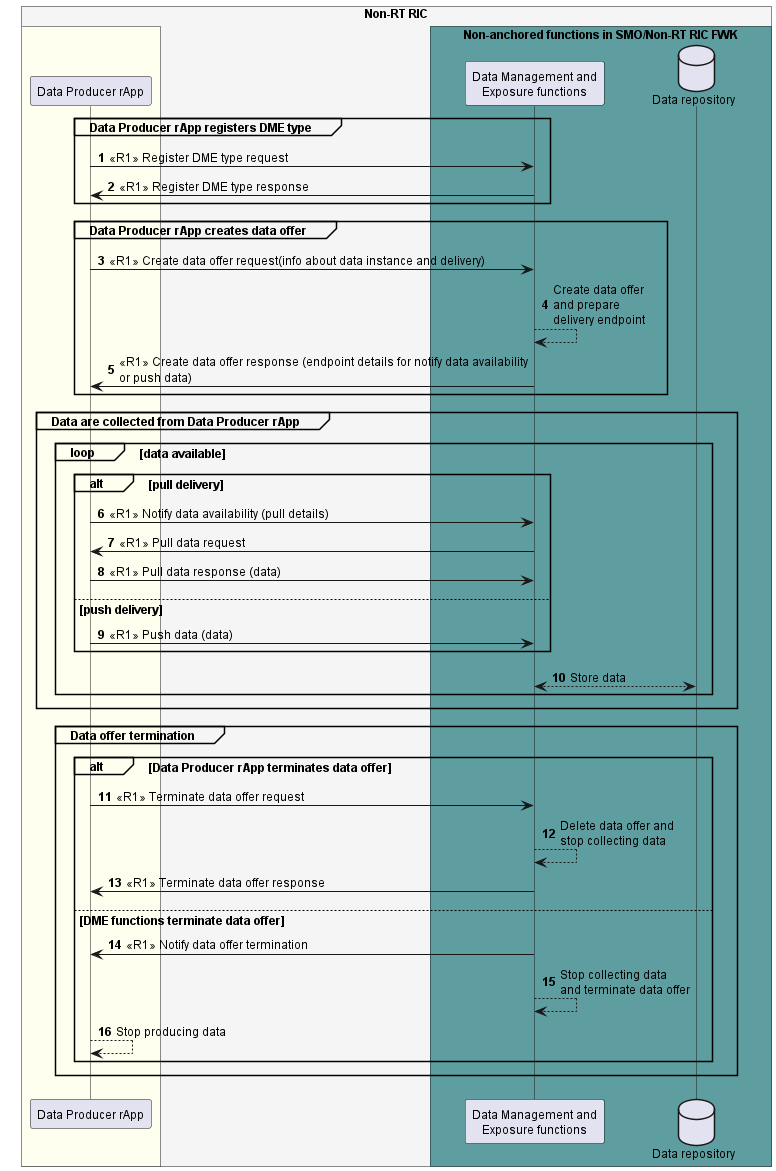


Figure 10.1.5-1: Example workflow using data offer procedures to trigger the collection and storage of data for later consumption

## 10.2 A1-related procedures

### 10.2.1 EI results collection from rApps

Figure 10.2.1-1 illustrates an example workflow of collecting data from rApps (over the R1 interface) that is delivered as EI job results to the Near-RT RICs (over the A1 interface). EI job results refer to the data being transferred from the Non-RT RIC to the Near-RT RICs, while EI source data refers to the data produced by the rApp that acts as the source for the data provided as EI job results. Correspondingly, EI type refers to the data type that is made available to the Near-RT RICs while the corresponding DME type registered by the rApp is referred to as EI source DME type. Moreover, EI job refers to a request for EI by the Near-RT RIC while EI source data subscription refers to the corresponding subscription to the rApp that acts the source for the data.

NOTE 1: EI source data type refers to any DME type that is registered with the DME functions and the corresponding data are used as the source of EI job results.

The rApp which generates the enrichment information first registers the EI source DME type in the DME functions. After successful DME type registration, the rApp registers the EI type for which it can produce enrichment information to the A1 enrichment information functions. In the registration request, the rApp should include the EI source DME type registered in the DME functions. The EI type is to be exposed via the A1 interface to the Near-RT RIC. The EI source DME type allows the A1 enrichment information functions to subscribe EI source data from DME functions.

The A1 enrichment information functions query DME functions about the EI source DME type. If the query is successful, the EI type registered by the rApp is made available over the A1 interface. The Near-RT RIC queries available EI type identifiers in the Non-RT RIC. The Near-RT RIC queries EI type and creates an EI job via the A1 interface. After receiving the EI job creation request from the Near-RT RIC, the A1 enrichment information functions subscribe EI source data from DME functions. The DME functions further subscribe EI source data from the rApp over the R1 interface.

Each time new EI source data are available, the rApp delivers EI source data to the DME functions using pull or push delivery. The DME functions deliver EI source data to the A1 enrichment information functions. The A1 enrichment formation functions establish the connection with Near-RT RIC and push the EI job results via the A1 interface.

NOTE 2: SMO/Non-RT RIC framework internal interactions are not specified.

NOTE 3: As an alternative to using the data subscription procedure, the data request procedure can be used to collect EI source data from the rApp.

@startuml

!pragma teoz true

skinparam ParticipantPadding 5

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam lifelineStrategy solid

skinparam SequenceGroupBackgroundColor Transparent

skinparam SequenceGroupBodyBackgroundColor Transparent

autonumber

box "Non-RT RIC" #whitesmoke

box #ivory

participant "rApp" as app

endbox

box "Non-anchored functions in SMO/Non-RT RIC FWK" #cadetBlue

participant "Data Management and Exposure functions" as dme

participant "A1 enrichment information functions" as eif

endbox

endbox

Participant "Near-RT RIC" as ran

group rApp register EI source DME type to DME functions

app -> dme : <<R1>> EI source DME type registration request

app <- dme : <<R1>> EI source DME type registration response

end

group rApp register EI type

app -> eif : <<R1>> EI type registration request

app <- eif : <<R1>> EI type registration response

end

group A1 EI functions query EI source DME type from DME

eif --> dme : EI source DME type query request

eif <-- dme : EI source DME type query response

end

group Query EI type identifiers

ran -> eif : <<A1>> EI type identifiers query request

ran <- eif : <<A1>> EI type identifiers query response

end

group Query EI type

ran -> eif : <<A1>> EI type query request

ran <- eif : <<A1>> EI type query response

end

group Create EI job

ran -> eif : <<A1>> EI job creation request

ran <- eif : <<A1>> EI job creation response

end

group A1 EI functions subscribe EI source data from DME

eif --> dme : EI source data subscription request

eif <-- dme : EI source data subscription response

end

group DME functions subscribe EI source data from rApp

dme -> app : <<R1>> EI source data subscription request

dme <- app : <<R1>> EI source data subscription response

end

loop EI source data available

alt pull delivery

app -> dme: <<R1>> Notify data availability (pull delivery details)

app <- dme: <<R1>> Pull data request

app -> dme: <<R1>> Pull data response (EI source data in payload)

else push delivery

app -> dme : <<R1>> Push data message (EI source data in payload)

end

dme --> eif : EI source data delivery

group Deliver EI job results

eif -> ran : <<A1>> Push EI job results request

eif <- ran : <<A1>> Push EI job results response

end

end

@enduml

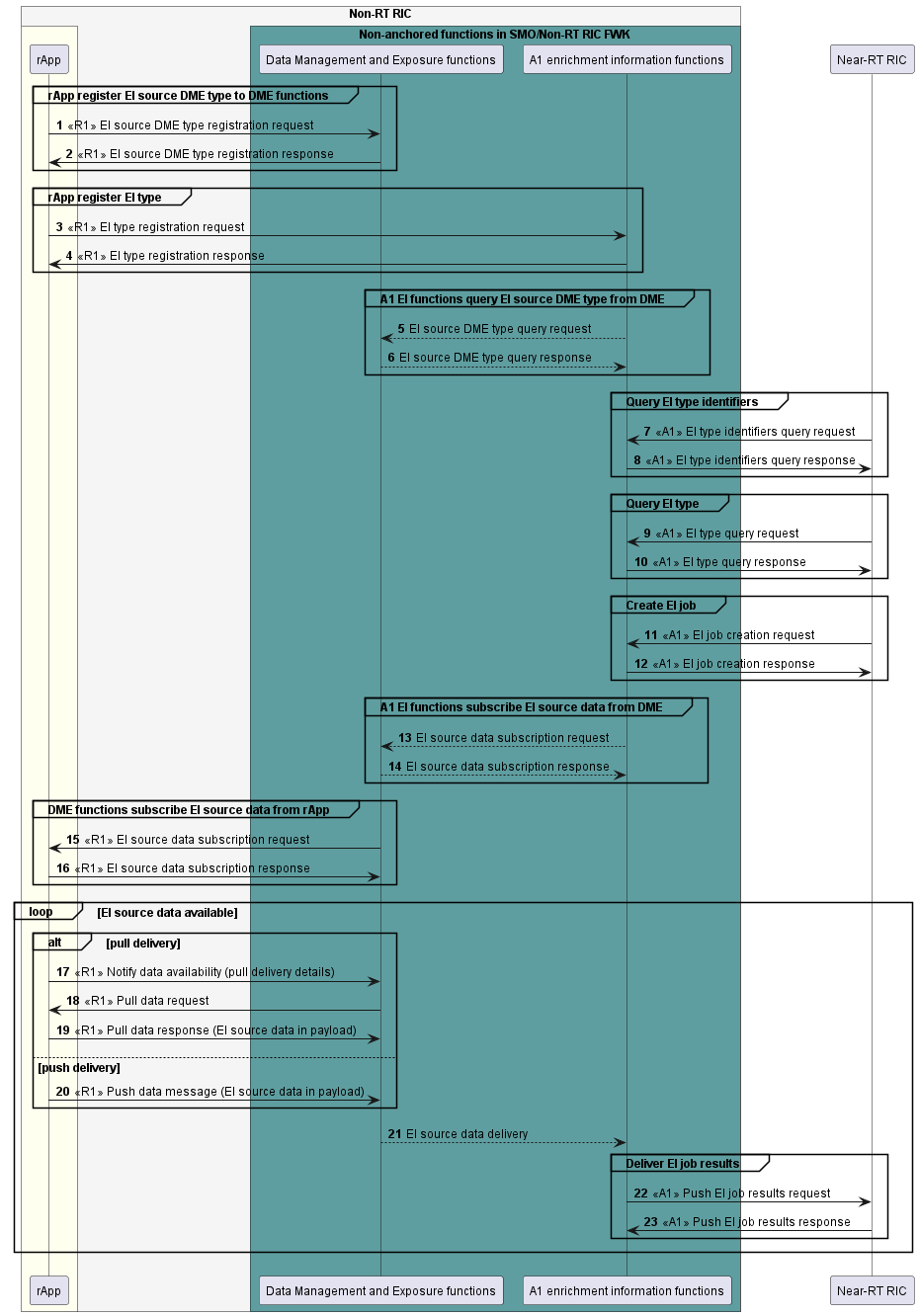


Figure 10.2.1-1: Example workflow for EI results collection procedures

NOTE 4: A procedure for EI source data collection from external sources is not specified in the present document.

### 10.2.2 rApp creating a policy with a Near-RT RIC identifier

Figure 10.2.2-1 illustrates an example workflow of rApp creating a policy over the R1 interface with a Near-RT RIC identifier, and the Non-RT RIC creates the corresponding A1 policy at a Near-RT RIC over the A1 interface. In this example, it is assumed that the rApp obtained the Near-RT RIC identifier by consuming other R1 services.

The rApp sends create policy request to the A1 policy functions in the SMO/Non-RT RIC framework, providing the policy object and a Near-RT RIC identifier. The A1 policy functions validate the creation request, e.g., whether the identified Near-RT RIC is operating, whether the rApp is authorized to create an A1 policy in the identified Near-RT RIC, whether the policy object prodived in request is valid against the policy type schema, etc.

If the validation fails, then A1 policy functions send the failure indication in the create policy response to rApp. Otherwise, the A1 policy functions generate an identifier for the policy and send a create A1 policy request to the target Near-RT RIC, including the policy identifier and the A1 policy object.

If the Near-RT RIC indicates a successful creation in the create A1 policy response, the create policy response to the rApp over R1 carries the successful indication and the policy identifier. The rApp assumes that the policy is now enforced in the target Near-RT RIC. If the policy creation in A1 fails, then the Near-RT RIC sends proper error codes in the create A1 policy response. The A1 policy functions forward the failure indication to the rApp.

@startuml

!pragma teoz true

skinparam ParticipantPadding 5

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam lifelineStrategy solid

skinparam SequenceGroupBackgroundColor Transparent

skinparam SequenceGroupBodyBackgroundColor Transparent

autonumber

box "Non-RT RIC" #whitesmoke

box #ivory

participant "rApp" as app

endbox

box "Non-anchored functions in \nSMO/Non-RT RIC FWK" #cadetBlue

participant "A1 Policy functions" as a1p

endbox

endbox

participant "Near-RT RIC" as ric

app -> a1p : <<R1>> Create policy request (Near-RT RIC id)

a1p -> a1p : Validate

alt Validation of the create policy request fails

app <- a1p : <<R1>> Create policy response ("Fail")

else Validation of the create policy request is successful

a1p -> a1p : Generate policy id

a1p -> ric : <<A1>> Create A1 policy request (policy id)

alt A1 policy creation is successful

a1p <- ric : <<A1>> Create A1 policy response ("Success")

app <- a1p : <<R1>> Create policy response (policy id, "Success")

else A1 policy creation is failed

a1p <- ric : <<A1>> Create A1 policy response ("Fail")

app <- a1p : <<R1>> Create policy response ("Fail")

@enduml

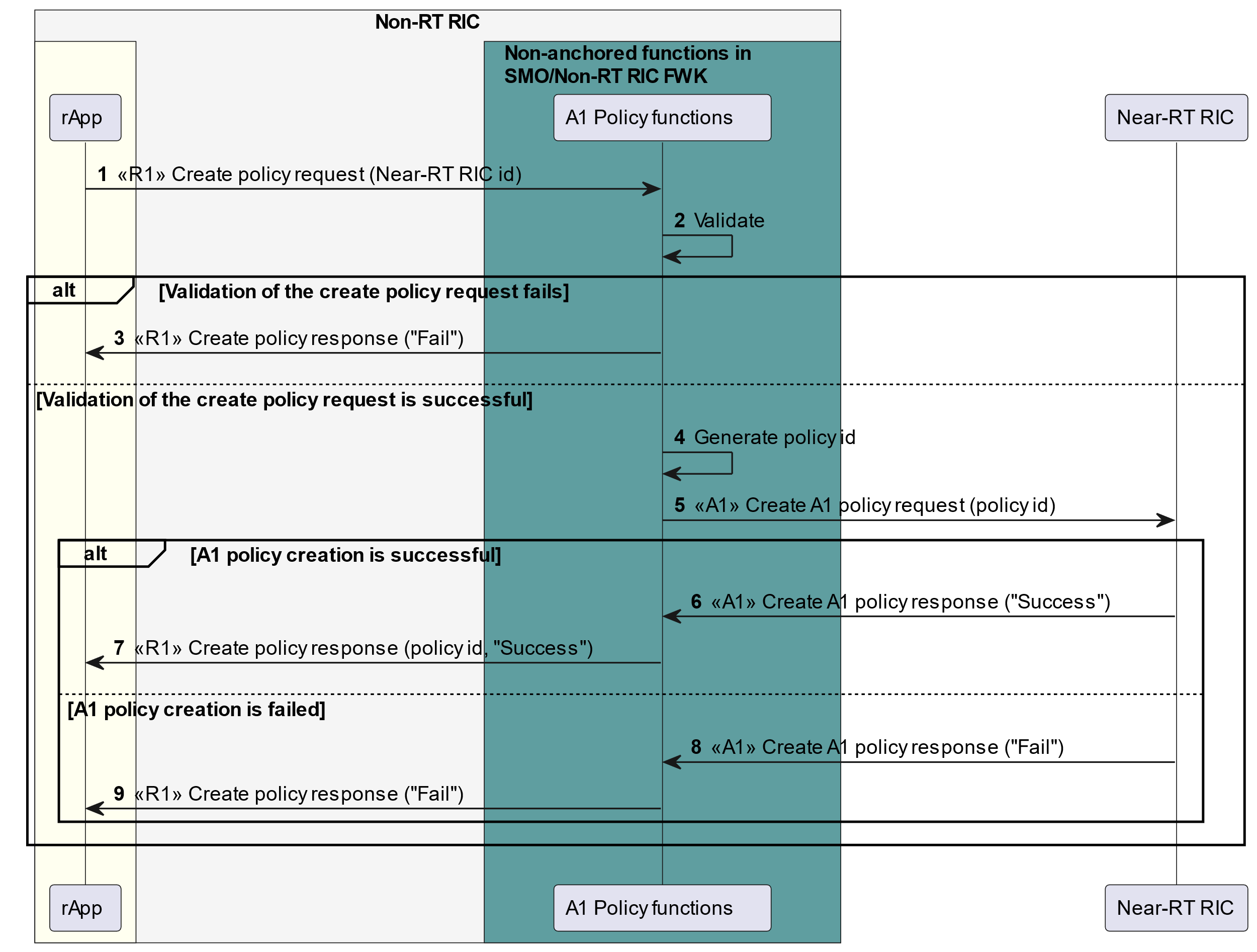


Figure 10.2.2-1: Example workflow of rApp creating a policy with a Near-RT RIC identifier

## 10.3 RAN OAM-related procedures

### 10.3.1 Alarm query procedure

Figure 10.3.1-1 illustrates an example workflow of querying alarm information. Alarm information from the nodes terminating the O1 interface is typically received by the SMO via O1. It is consolidated by the SMO/Non-RT RIC framework and can be queried by rApps via the R1 interface.

NOTE: M-Plane alarms and slice alarms are not specified in the present document.

@startuml

!pragma teoz true

skinparam ParticipantPadding 5

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam SequenceGroupBackgroundColor Transparent

skinparam SequenceGroupBodyBackgroundColor Transparent

skinparam lifelineStrategy solid

box "Non-RT RIC" #whitesmoke

box #ivory

participant "rApp" as rapp

endbox

box "Non-anchored functions in SMO/Non-RT RIC FWK" #cadetBlue

participant " RAN OAM-related functions" as o1rf

endbox

endbox

participant "Nodes terminating O1 interface" as me

group SMO/Non-RT RIC FWK consolidates alarm information

alt Using alarm notifications

ref over o1rf, me

<b>1</b> As specified in Clause 6.2.1.3 "Fault Notification - Procedures"

in O-RAN.WG10.O1-Interface

end

else Using alarm list

ref over o1rf, me

<b>2</b> As specified in Clause 6.2.2.3 "Fault Supervision Control -

Procedures"

in O-RAN.WG10.O1-Interface

end

end

autonumber 3

o1rf --> o1rf: Consolidate alarm information

end

rapp -> o1rf: <<R1>> Query alarm information request(rApp identifier, filter criteria)

rapp <- o1rf: <<R1>> Query alarm information response(alarm information)

@enduml

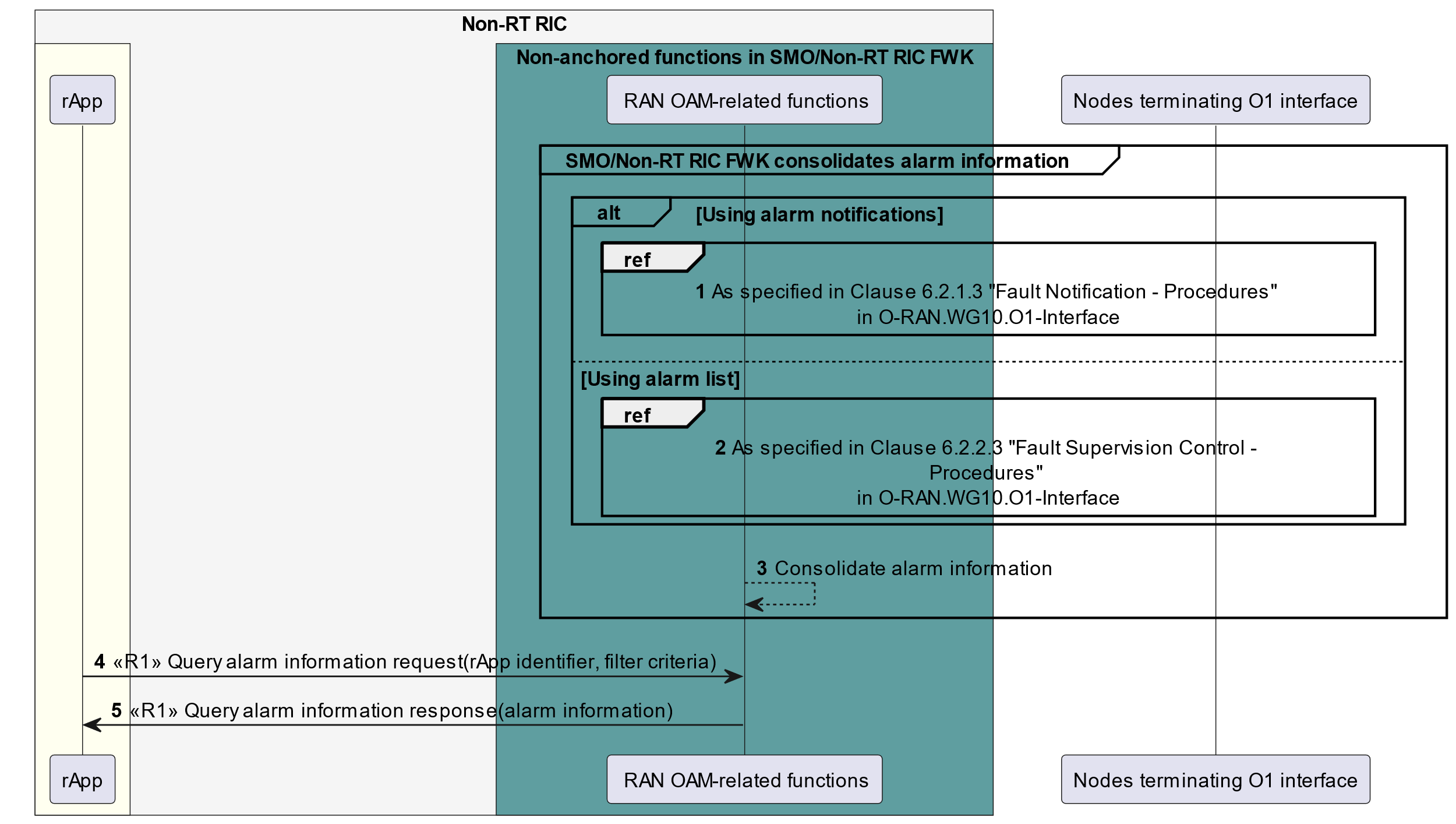


Figure 10.3.1-1: Example workflow of alarm query

### 10.3.2 Change alarm acknowledgement state procedure

Figure 10.3.2-1 illustrates an example workflow of changing the acknowledgement state of an alarm. Alarm information from the nodes terminating the O1 interface is typically received by the SMO via O1. It is consolidated by the SMO/Non-RT RIC framework. The rApps can change the acknowledgement state of the alarms that are known by the SMO/Non-RT RIC framework. Alarm acknowledgment state information is usually managed by the SMO/Non-RT RIC framework and is only passed to the radio nodes / Near-RT RIC if alarm acknowledgement / unacknowledgement is supported by these entities.

NOTE: M-Plane alarms and slice alarms are not specified in the present document.

@startuml

!pragma teoz true

skinparam ParticipantPadding 5

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam SequenceGroupBackgroundColor Transparent

skinparam SequenceGroupBodyBackgroundColor Transparent

skinparam lifelineStrategy solid

box "Non-RT RIC" #whitesmoke

box #ivory

participant "rApp" as rapp

endbox

box "Non-anchored functions in SMO/Non-RT RIC FWK" #cadetBlue

participant "RAN OAM-related functions" as o1rf

endbox

endbox

participant "Nodes terminating O1 interface" as me

group SMO/Non-RT RIC FWK consolidates alarm information

alt Using alarm notifications

ref over o1rf, me

<b>1</b> As specified in Clause 6.2.1.3 "Fault Notification - Procedures"

in O-RAN.WG10.O1-Interface

end

else Using alarm list

ref over o1rf, me

<b>2</b> As specified in Clause 6.2.2.3 "Fault Supervision Control -

Procedures" in O-RAN.WG10.O1-Interface

end

end

autonumber 3

o1rf --> o1rf: Consolidate alarm information

end

rapp -> o1rf: <<R1>> Change alarm acknowledgement state request\n(rApp identifier, alarm identifier, new acknowledgment state)

o1rf --> o1rf: Store alarm acknowledgement state information

rapp <- o1rf: <<R1>> Change alarm acknowledgement state response

opt if alarm acknowledgement state change is supported by managed entity

ref over o1rf, me

<b>7</b> Update alarm state as specified in Clause 6.2.2.3

"Fault Supervision Control - Procedures" in O-RAN.WG10.O1-Interface

end

end

@enduml

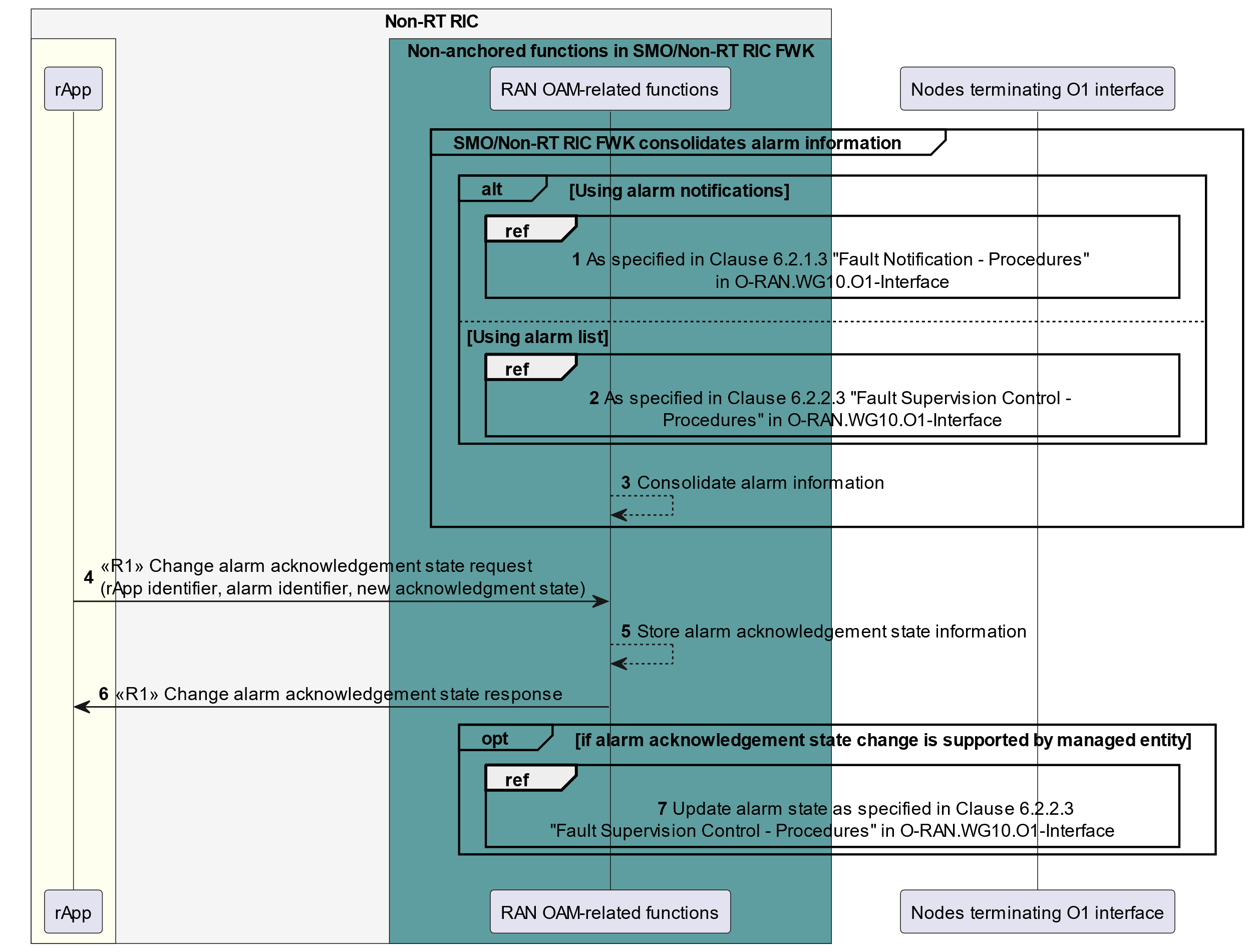


Figure 10.3.2-1: Example workflow of alarm acknowledgement state change

### 10.3.3 Read configuration data procedure

Figure 10.3.3-1 and figure 10.3.3-2 illustrate an example workflow of reading the current CM configurations of the network elements. CM configuration data from the RAN nodes terminating the O1 interface is typically received by the SMO via O1. It is cached and continuously synchronized by the SMO/Non-RT RIC framework (see figure 10.3.3-1) and can be retrieved by rApps via the R1 interface (see figure 10.3.3-2). The read configuration data request initiated by rApps may contain requests for reading configuration data for multiple RAN nodes in order to save bandwidth on the R1 interface.

NOTE: Read configuration data procedure is defined in O-RAN R1GAP [9], clause 5.4.5.

@startuml

!pragma teoz true

skinparam ParticipantPadding 70

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam lifelineStrategy solid

autonumber

box "Non-RT RIC" #whitesmoke

box "Non-anchored functions in SMO/Non-RT RIC Framework" #cadetBlue

participant "RAN OAM-related functions" as fmsp

endbox

endbox

participant “Nodes terminating O1 interface” as dp

loop Synchronize configuration data with RAN nodes

alt

ref over fmsp,dp

As specified in Clause 6.1.5.3

"Read Managed Object Instance Attributes - Procedures"

in O-RAN.WG10.O1-Interface

end

else

ref over fmsp,dp

As specified in Clause 6.1.6.3

"Notify Managed Object Instance Attributes - Procedures"

in O-RAN.WG10.O1-Interface

end

end

fmsp --> fmsp: Consolidate information

end

@enduml

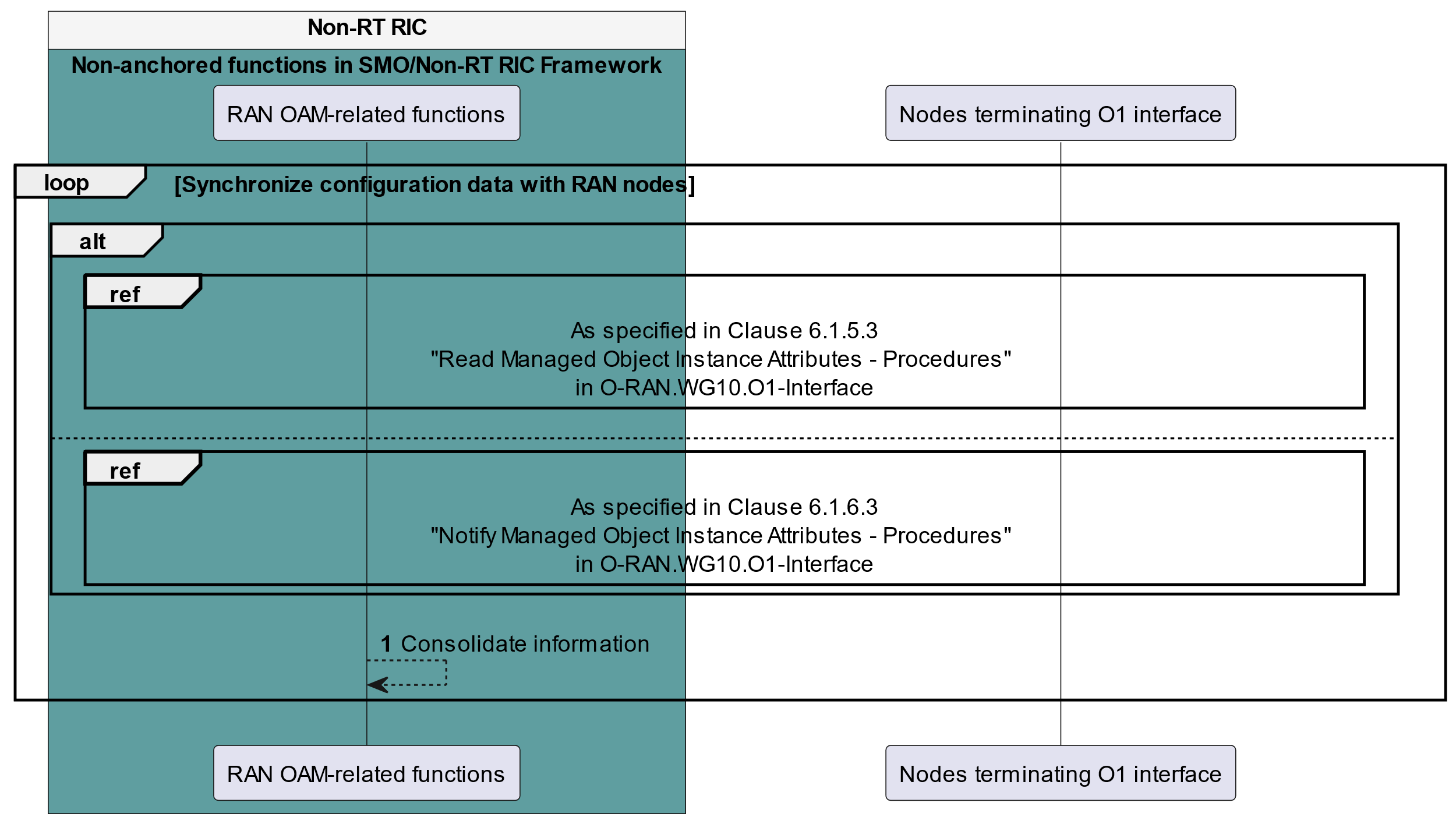


Figure 10.3.3-1: Example workflow for synchronizing configuration data in the SMO/Non-RT RIC framework with RAN nodes

@startuml

!pragma teoz true

skinparam ParticipantPadding 70

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam lifelineStrategy solid

box "Non-RT RIC" #whitesmoke

box #ivory

participant "rApp" as rapp

endbox

box "Non-anchored functions in SMO/Non-RT RIC Framework" #cadetBlue

participant "RAN OAM-related functions" as fmsp

endbox

endbox

participant “Nodes terminating O1 interface” as dp

autonumber

rapp -> fmsp: <<R1>> Request for reading configuration data (rAppId, queryCriteria)

fmsp --> fmsp: Check if the requested data has been \n collected already and in sync with the RAN nodes

opt if the requested data has not being collected yet or not in sync with the RAN nodes

ref over fmsp,dp

As specified in Clause 6.1.5.3

"Read Managed Object Instance Attributes - Procedures"

in O-RAN.WG10.O1-Interface

end

end

fmsp -> rapp: <<R1>> Read configuration data response <configurationData>

@enduml

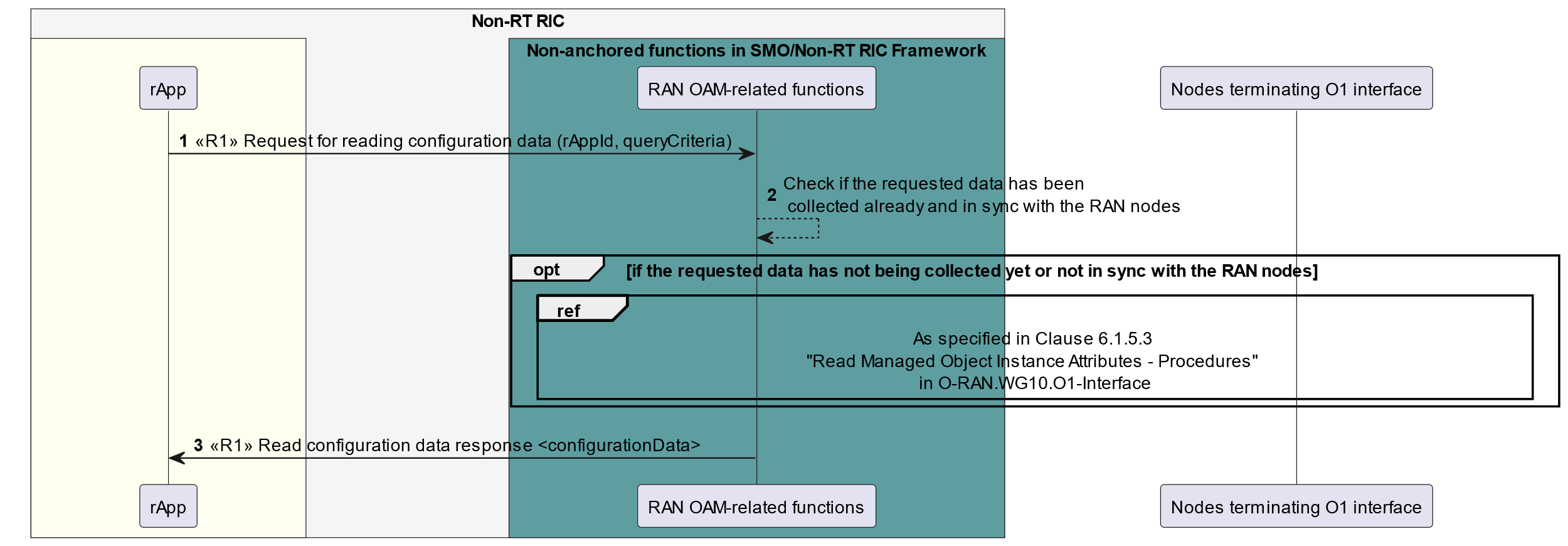


Figure 10.3.3-2: Example workflow for read configuration data procedure

The RAN OAM-related functions may maintain a local data store (implementation specific) caching part or all of the CM configurations of the RAN nodes for fast retrieval of configuration data from rApps and minimizing the signalling overhead on the O1 interface. In this example, as illustrated in figure 10.3.3-1, the RAN OAM-related functions synchronize the configuration data in its local data store with the RAN nodes continuously based on read MOI attributes procedures or notifications of MOI attribute changes over the O1 interface or internal knowledge of the attribute changes made by the rApps and framework functions within the SMO which is implementation specific. When the rApp initiates the read configuration data procedure, as illustrated in figure 10.3.3-2, the RAN OAM-related functions optionally check if the requested configuration data has been collected already and is up to date. The requested configuration data by the rApp might not be always cached in the local data store. If the requested data has not being collected, the RAN OAM-related functions initiate read MOI attributes procedures to collect the latest configuration data from the RAN nodes via the O1 interface as specified in O-RAN O1 interface [11], clause 6.1.5.3. Finally, a read configuration data response with the latest configuration data is sent to rApp by the RAN OAM-related functions.

### 10.3.4 Write configuration changes procedure

Figure 10.3.4-1 illustrates an example workflow of writing the CM configuration changes to the network elements. CM configurations in the RAN nodes terminating the O1 interface are typically managed by the SMO via O1. Configuration information is consolidated by the SMO/Non-RT RIC framework and can be modified by rApps via the R1 interface. The write configuration changes request initiated by rApps may contain bundled write requests for multiple RAN nodes in order to save bandwidth on the R1 interface.

@startuml

!pragma teoz true

skinparam ParticipantPadding 70

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam lifelineStrategy solid

box "Non-RT RIC" #whitesmoke

box #ivory

participant "rApp" as rapp

endbox

box "Non-anchored functions in SMO/Non-RT RIC Framework" #cadetBlue

participant "RAN OAM-related functions" as fmsp

endbox

endbox

participant “Nodes terminating O1 interface” as dp

autonumber

opt

rapp -> fmsp: <<R1>> Subscribe for write configuration job status notifications request

fmsp -> rapp: <<R1>> Subscribe for write configuration job status notifications response

end

rapp -> fmsp: <<R1>> Request for writing configuration changes\n(rAppId, configuration changes information)

fmsp --> fmsp: AuthZ

fmsp --> fmsp: Validate request

fmsp --> fmsp: Store request

fmsp -> rapp: <<R1>> Write configuration changes response (requestId)

fmsp --> fmsp: Process request

ref over fmsp,dp

As specified in Clause 6.1.3.3

"Modify Managed Object Instance Attributes - Procedures"

in O-RAN.WG10.O1-Interface

end

alt if subscribed

fmsp -> rapp: <<R1>> Write configuration changes job status notification \n(requestId, configuration changes result)

else otherwise

rapp -> fmsp: <<R1>> Query write configuration job status (requestId)

fmsp -> rapp: <<R1>> Query write configuraiton job status reponse (configuration changes result)

end

@enduml

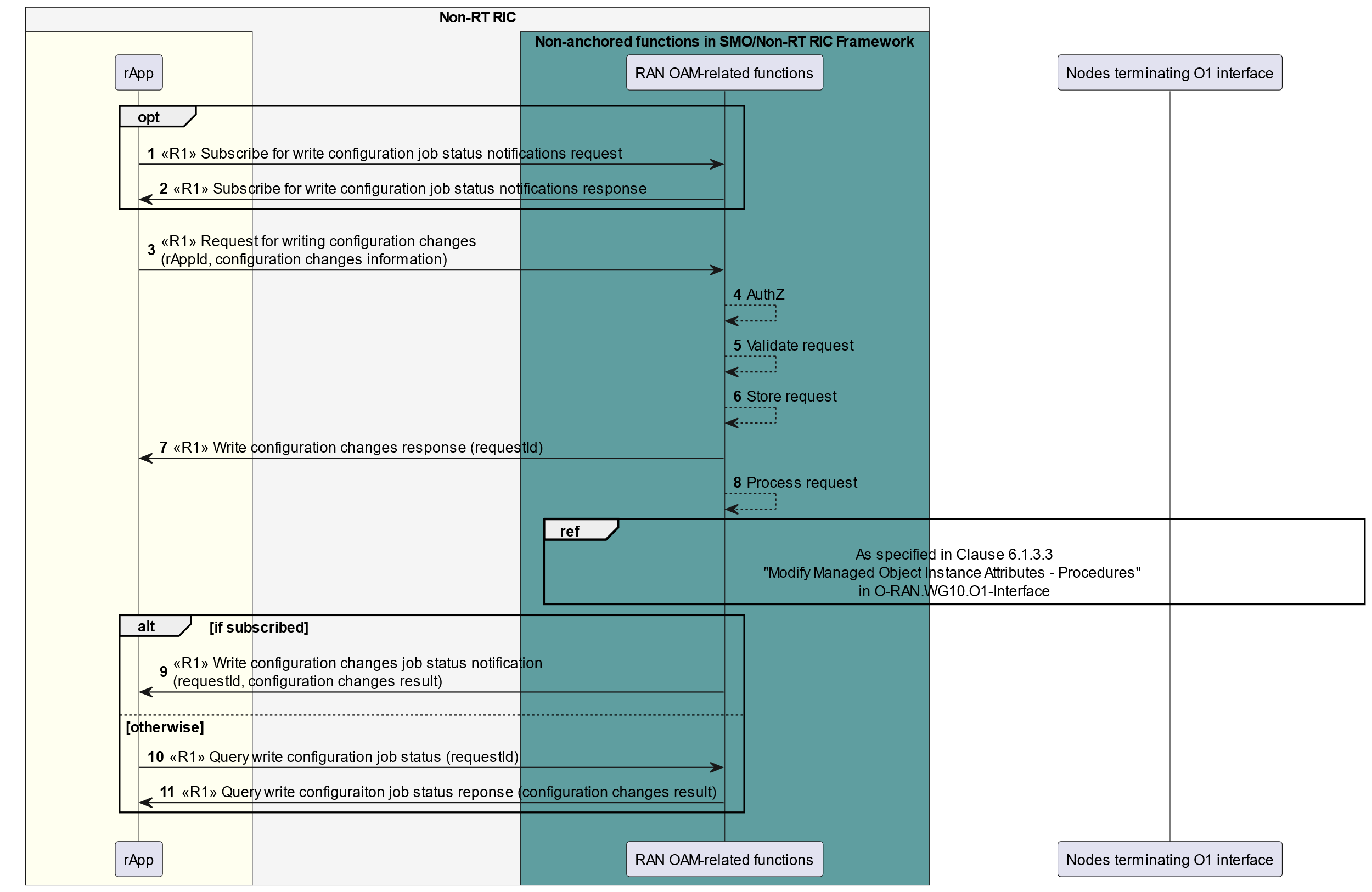


Figure 10.3.4-1: Example workflow for write configuration changes procedure

The write configuration changes request is handled in asynchronous approach. When the rApp initiates the write configuration changes procedure, the RAN OAM-related functions authenticate the request with SME, validate the request, store the request as job and acknowledge the request by sending write configuration changes response indicating that the request has been accepted for processing, but the processing job has not been completed. An assigned requested identifying the created processing job is included in the response. The actual processing job for the write configuration changes request which includes e.g., conflict detection, conflict mitigation and writing the configuration changes to each RAN node as requested via the O1 interface, happens asynchronously. If the rApp has subscribed for the notification of the write configuration changes job status, the RAN OAM-related functions send a notification to the rApp once the write configuration changes job is complete, indicating the completion of the job and providing information about success, partial success, or failure result with failure reasons. Otherwise, the rApp can query for the write configuration changes job status and result with query procedures.

## 10.4 AI/ML workflow procedures

### 10.4.1 rApps register, discover AI/ML models, and subscribe to models changes

Figure 10.4.1-1 illustrates an example workflow of rApps using AI/ML workflow functions to register, discover AI/ML models and subscribe to registered models changes.

A model producer rApp sends a request to the AI/ML workflow functions to register an AI/ML model. The register AI/ML model request contains AI/ML model information (e.g., model accuracy, model usage scenario, model training dataset information, etc.). Upon receipt of the registration request, the AI/ML workflow functions authorize the model producer rApp and register the AI/ML model. After successful registration, the AI/ML workflow functions assign an AI/ML model registration identifier and communicate this registration identifier to the model producer rApp in a register AI/ML model response.

A model consumer rApp sends a request to the AI/ML workflow functions to discover registered AI/ML models based on optional selection criteria. After receiving the request, the AI/ML workflow functions authorize the model consumer rApp and communicate to the model consumer rApp the model identifier(s) and metadata of AI/ML model(s) that match the provided selection criteria.

The model consumer rApp can send a subscribe AI/ML model change request to the AI/ML workflow functions to subscribe to the changes in a registered AI/ML model. Upon receipt of the subscription request, AI/ML workflow functions authorize the model consumer rApp and create a subscription. After successful subscription, the AI/ML workflow functions communicate to the model consumer rApp the subscription ID in a subscribe AI/ML model change response.

When there is a change in the subscribed AI/ML model, the AI/ML workflow functions send notifications of changes in the registered AI/ML model to the subscribed AI/ML model consumer rApp.

When the model producer rApp sends a deregister request to the AI/ML workflow functions to deregister a registered AI/ML model, the AI/ML workflow functions deregister the AI/ML model and communicate to the rApp that the deregistration was successful. The AI/ML workflow functions send notifications of the deregistration of the registered AI/ML model to the subscribed AI/ML model consumer rApp.

The model consumer rApp can send an unsubscribe request to the AI/ML workflow functions to unsubscribe to changes in a registered AI/ML model. Upon reception of the unsubscribe request, the AI/ML workflow functions unsubscribe the model consumer rApp to the AI/ML model change. After successfully unsubscribing, the AI/ML workflow functions communicate to the model consumer rApp that unsubscribing was successful.

@startuml

!pragma teoz true

skinparam ParticipantPadding 5

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam lifelineStrategy solid

skinparam SequenceGroupBackgroundColor Transparent

skinparam SequenceGroupBodyBackgroundColor Transparent

autonumber

box "Non-RT RIC" #whitesmoke

box #ivory

participant "Model Consumer rApp" as rApp

endbox

box " Non-anchored functions in SMO/Non-RT RIC Framework " #cadetBlue

participant "AI/ML workflow functions" as ML

endbox

box #ivory

participant "Model Producer rApp" as src

endbox

endbox

group Model Producer rApp register model

src -> ML: <<R1>> Register AI/ML model request (rAppID, AI/ML model information)

ML -> ML: Authz

ML -> ML: register AI/ML model

ML -> src: <<R1>> Register AI/ML model response (AI/ML model registration identifier)

end

group Model Consumer rApp discover model

rApp -> ML: <<R1>> Discover AI/ML model request (rAppID, selection criteria)

ML -> ML: Authz

ML -> rApp: <<R1>> Discover AI/ML model response（AI/ML model identifier(s) and metadata)

end

group Model Consumer rApp subscribe model change

rApp -> ML: <<R1>> Subscribe AI/ML model change request (rAppID, AI/ML model identifier(s))

ML -> ML: Create subscription

ML -> rApp: <<R1>> Subscribe AI/ML model change response (subscription ID)

end

group The subscribed model has been changed

ML -> rApp: <<R1>> Notify AI/ML model changes (subscription ID, AI/ML model identifier, available change details)

end

group The subscribed model has been deregistered

group Model Producer rApp deregister model

src -> ML: <<R1>> Deregister AI/ML model (rAppID, AI/ML model registration identifier)

ML -> ML: Deregister AI/ML model

ML -> src: <<R1>> Deregister AI/ML model response

end

ML -> rApp: <<R1>> Notify AI/ML model deregistration (subscription ID, AI/ML model identifier)

end

group Model Consumer rApp unsubscribe model change

rApp -> ML: <<R1>> Unsubscribe AI/ML model change request (rAppID, subscription ID)

ML -> ML: Unsubscription model change

ML -> rApp: <<R1>> Unsubscribe AI/ML model change response

end

@enduml

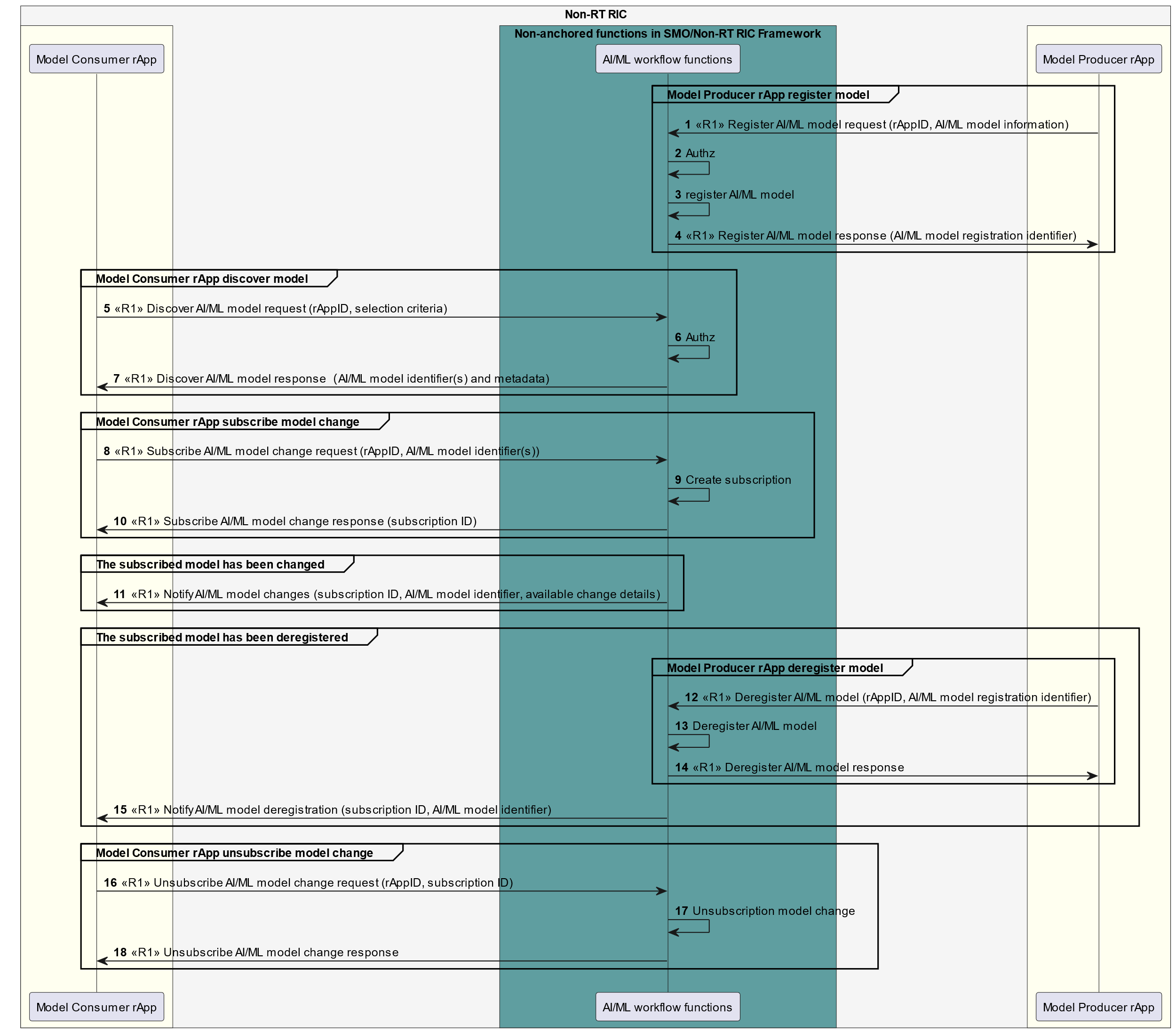


Figure 10.4.1-1: Example workflow of AI/ML model registration, discovery, and subscription to a registered AI/ML model changes

NOTE: The metadata includes model related information, e.g., training dataset information, etc.

### 10.4.2 AI/ML training procedures

The following figures illustrate an example workflows of an AI/ML training services Consumer rApp requesting AI/ML training of an AI/ML model. In figure 10.4.2-1, the AI/ML workflow functions in the Non-RT RIC framework are the AI/ML training services Producer. In figure 10.4.2-2, an AI/ML training services Producer rApp produces the AI/ML training services. In figure 10.4.2-3, the AI/ML workflow functions coordinate AI/ML training in the SMO/Non-RT RIC, and they relay training request and subsequent messages between the AI/ML training services Consumer App and a proper AI/ML training services Producer rApp based on registered training capabilities.

The AI/ML training services Consumer rApp sends a training request to the AI/ML training services Producer, providing model identifier of the model to be trained, information about required data for training, training requirements such as validation criteria, and a URI to receive notifications, etc. The AI/ML training services Producer, if accepts the training request, sends back request training response containing a training identifier.

After accepting the training request, the AI/ML training services Producer may retrieve the model from the model repository and consumes training data via DME based on the information provided by the AI/ML training services Consumer rApp. The AI/ML training services Producer then performs the training.

The AI/ML training services Consumer rApp can query the training job status using the received training job identifier. The AI/ML training services Producer provides the training job status in the response to the query.

If the AI/ML training services Consumer rApp wants to cancel the on-going training, it sends a cancel training request to the AI/ML training services Producer providing the training job identifier. The AI/ML training services Producer then terminates the training and response to the AI/ML training services Consumer rApp.

If the training completes, the AI/ML training services Producer may store the trained model in the model repository, and it then notifies the AI/ML training services Consumer rApp about the completion of the training, sending the notification to the notification URI received in the training request.

The AI/ML training services Consumer rApp can retrieve the trained model from the model repository, if the AI/ML training services Producer has stored the trained model in the model repository.

NOTE: The model repository can be implemented as part of the AI/ML workflow functions.

@startuml

!pragma teoz true

skinparam ParticipantPadding 5

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam lifelineStrategy solid

skinparam SequenceGroupBackgroundColor Transparent

skinparam SequenceGroupBodyBackgroundColor Transparent

box "Non-RT RIC" #whitesmoke

box #ivory

participant "AI/ML training services \n Consumer rApp" as con

endbox

box "Non-anchored functions in SMO/Non-RT RIC FWK" #cadetBlue

participant "AI/ML workflow functions" as aif

participant "DME functions" as dme

database "Model repository" as rep

endbox

endbox

autonumber

con -> aif: <<R1>> Request training request \n (model id, required data, validation criteria, notifcation URI, etc.)

con <- aif: <<R1>> Request training response (training job id)

rnote over aif, rep

Retrieve model (optional)

end note

rnote over aif, dme

Data consumption for training data

end note

rnote over aif

Perform training

end note

con -> aif: <<R1>> Query training job status request (training job id)

con <- aif: <<R1>> Query training job status response (training job status)

alt AI/ML training services Consumer cancels training

con -> aif: <<R1>> Cancel training request (training job id)

rnote over aif

Terminate training

end note

con <- aif: <<R1>> Cancel training response

else AI/ML training services Producer completes training

rnote over aif

Training completed

end note

rnote over aif, rep

Store model (optional)

end note

con <- aif: <<R1>> Training job status notification \n (training job id, model id, training job status: "Completed")

end

@enduml

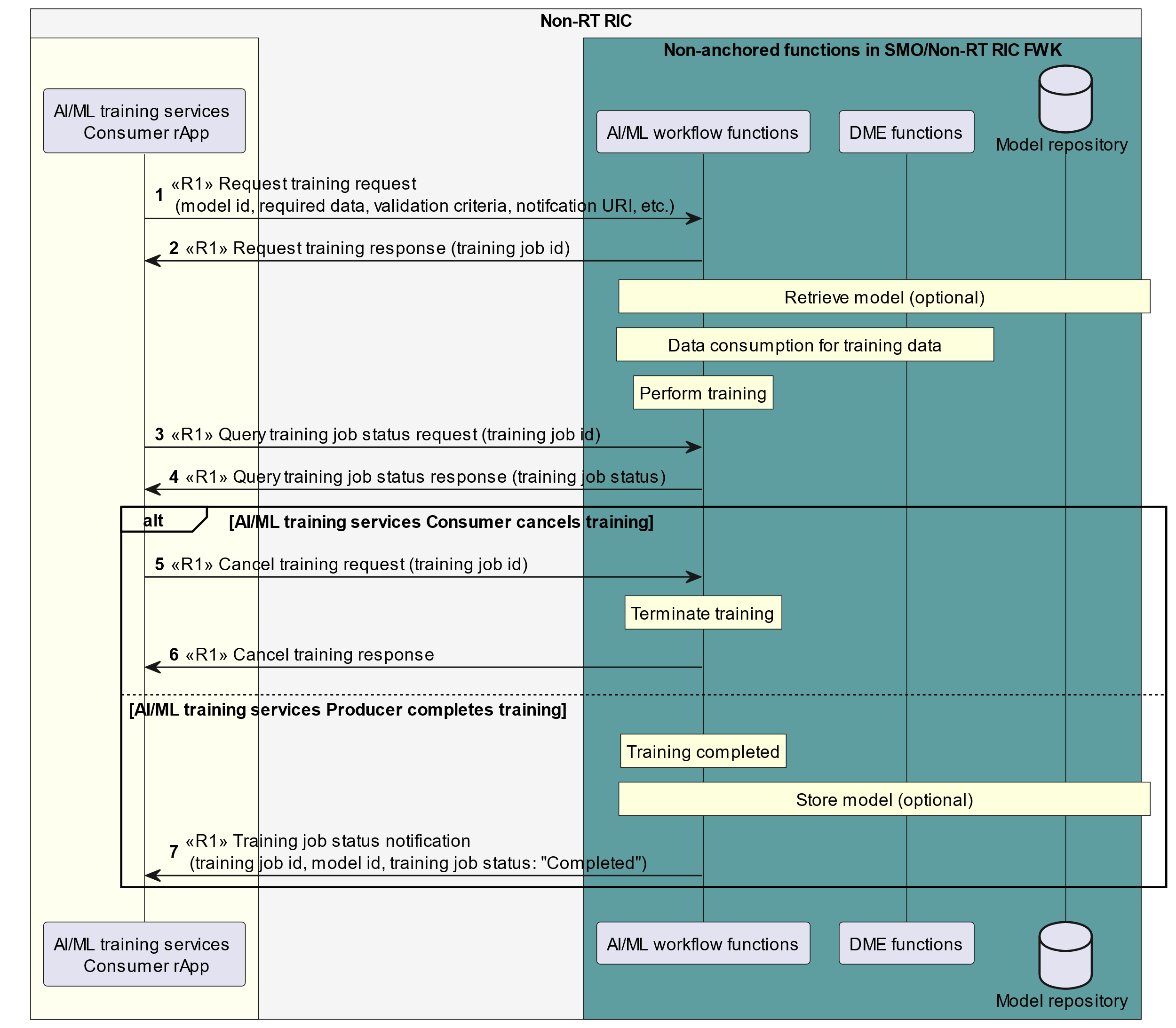


Figure 10.4.2-1: Example workflow of AI/ML training: AI/ML workflow functions act as the training service Producer

@startuml

!pragma teoz true

skinparam ParticipantPadding 5

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam lifelineStrategy solid

skinparam SequenceGroupBackgroundColor Transparent

skinparam SequenceGroupBodyBackgroundColor Transparent

box "Non-RT RIC" #whitesmoke

box #ivory

participant "AI/ML training services \n Consumer rApp" as con

participant "AI/ML training services \n Producer rApp" as pro

endbox

box "Non-anchored functions in \n SMO/Non-RT RIC FWK" #cadetBlue

participant "DME functions" as dme

database "Model repository" as rep

endbox

endbox

autonumber

con -> pro: Request training request \n (model id, required data, \n validation criteria, notification URI, etc.)

con <- pro: Request training response (training job id)

rnote over pro, rep

<<R1>> Retrieve model (optional)

end note

rnote over pro, dme

<<R1>> Data consumption for training data

end note

rnote over pro

Perform training

end note

con -> pro: Query training job status request (training job id)

con <- pro: Query training job status response (training job status)

alt AI/ML training services Consumer cancels training

con -> pro: Cancel training request (training job id)

rnote over pro

Terminate training

end note

con <- pro: Cancel training response

else AI/ML training services Producer completes training

rnote over pro

Training completed

end note

rnote over pro, rep

<<R1>> Store model (optional)

end note

con <- pro: Training job status notification \n (training job id, model id, training job status: "Completed")

end

@enduml

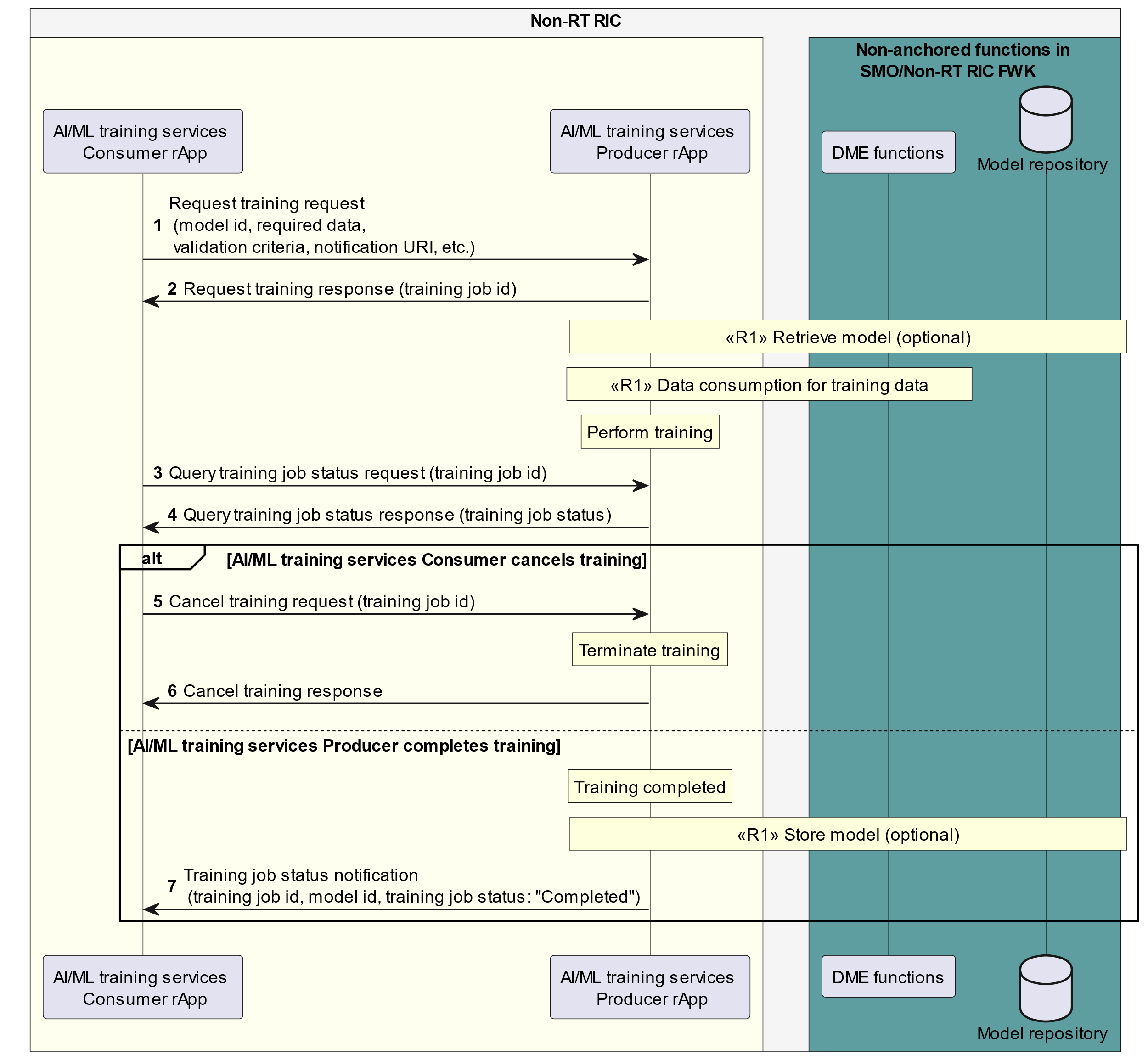


Figure 10.4.2-2: Example workflow of AI/ML training: AI/ML training services Producer rApp acts as the AI/ML training services Producer

@startuml

!pragma teoz true

skinparam ParticipantPadding 5

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam lifelineStrategy solid

skinparam SequenceGroupBackgroundColor Transparent

skinparam SequenceGroupBodyBackgroundColor Transparent

box "Non-RT RIC" #whitesmoke

box #ivory

participant "AI/ML training services \n Consumer rApp" as con

participant "AI/ML training services \n Producer rApp" as pro

endbox

box "Non-anchored functions in SMO/Non-RT RIC FWK" #cadetBlue

participant "AI/ML workflow functions" as aif

participant "DME functions" as dme

database "Model repository" as rep

endbox

endbox

autonumber

con -> aif: <<R1>> Request training request \n (model id, required data, validation criteria, notifcation URI, etc.)

aif --> aif: Identify proper training \n service Producer rApp

pro <- aif: <<R1>> Request training request \n (model id, required data, \n validation criteria, notifcation URI, etc.)

pro -> aif: <<R1>> Request training response \n (training job id)

con <- aif: <<R1>> Request training response (training job id)

rnote over pro, rep

<<R1>> Retrieve model (optional)

end note

rnote over pro, dme

<<R1>> Data consumption for training data

end note

rnote over pro

Perform training

end note

con -> aif: <<R1>> Query training job status request (training job id)

pro <- aif: <<R1>> Query training job status request \n (training job id)

pro -> aif: <<R1>> Query training job status response \n (training job status)

con <- aif: <<R1>> Query training job status response (training job status)

alt AI/ML training services Consumer cancels training

con -> aif: <<R1>> Cancel training request (training job id)

pro <- aif: <<R1>> Cancel training request \n (training job id)

rnote over pro

Terminate training

end note

pro -> aif: <<R1>> Cancel training response

con <- aif: <<R1>> Cancel training response

else AI/ML training services Producer completes training

rnote over pro

Training completed

end note

rnote over pro, rep

<<R1>> Store model (optional)

end note

pro -> aif: <<R1>> Training job status notification \n (training job id, model id, \n training job status: "Completed")

con <- aif: <<R1>> Training job status notification \n (training job id, model id, training job status: "Completed")

end

@enduml

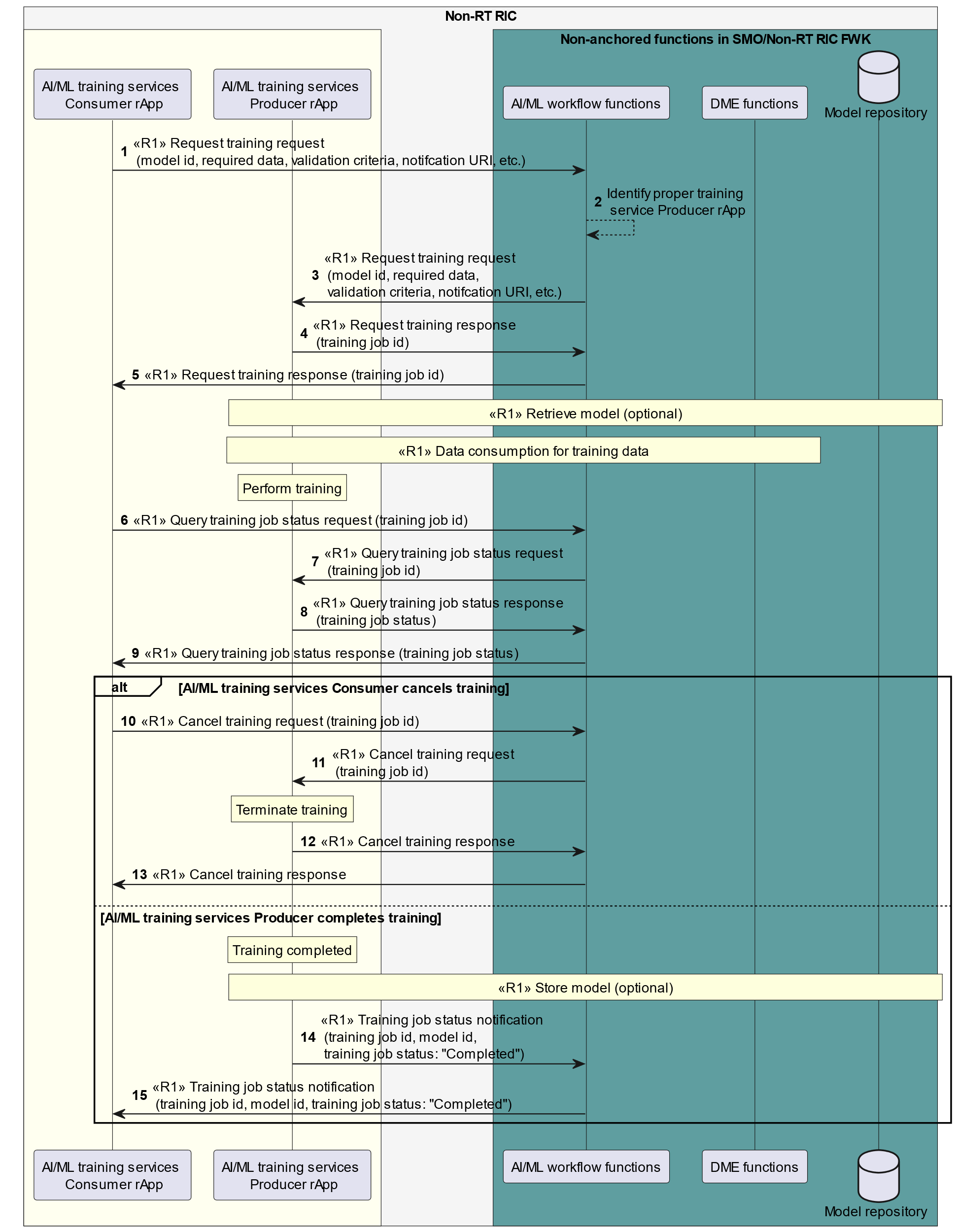


Figure 10.4.2-3: Example workflow of AI/ML training: AI/ML workflow functions coordinate AI/ML training

### 10.4.3 AI/ML model storage and retrieval procedures

Figure 10.4.3-1 illustrates an example workflow of AI/ML model storage and retrieval.

The AI/ML training services Consumer rApp sends training request to the AI/ML training services Producer rApp. After accepting the training request, the AI/ML training services Producer rApp may send a retrieve model request to the AI/ML workflow functions, containing the model identifier of the AI/ML model to be trained. The AI/ML workflow functions validate the retrieval request and provide the model location details of the requested AI/ML model. The AI/ML training services Producer rApp may retrieve the model to be trained via the model location details.

The AI/ML training services Producer rApp trains the AI/ML model. After the completion of the model training, the AI/ML training services Producer rApp may send store model request to the AI/ML workflow functions to store the trained model. It provides the model identifier and model location details for the trained model in the request. The AI/ML workflow functions response to the storage request with an updated identifier for the trained model.

The AI/ML training services Producer rApp notifies the AI/ML training services Consumer rApp about the completion of the model training. The AI/ML training services Consumer rApp then may send retrieve model request to the AI/ML workflow functions with the updated model identifier to retrieve the trained model. The AI/ML workflow functions response to the retrieval request with the model location details of the trained model.

@startuml

!pragma teoz true

skinparam ParticipantPadding 5

skinparam BoxPadding 10

skinparam defaultFontSize 12

skinparam lifelineStrategy solid

skinparam SequenceGroupBackgroundColor Transparent

skinparam SequenceGroupBodyBackgroundColor Transparent

box "Non-RT RIC" #whitesmoke

box #ivory

participant "AI/ML Training service \n Consumer rApp" as con

participant "AI/ML Training service \n Producer rApp" as pro

endbox

box "Non-anchored functions \n in SMO/Non-RT RIC FWK" #cadetBlue

participant "AI/ML workflow functions" as aif

endbox

endbox

autonumber

con -> pro: request training request \n (model id, required data, \n validation criteria, notifcation URI, etc.)

con <- pro: request training response \n (training job id)

opt

pro -> aif: <<R1>> retrieve model request \n (rApp id, model id)

pro <- aif: <<R1>> retrieve model response \n (model location details)

end

rnote over pro

Train AI/ML model

end note

opt

pro -> aif: <<R1>> store model request \n (rApp id, model id, \n model location details)

pro <- aif: <<R1>> store model response \n (updated model id)

end

con <- pro: Notify training complete \n (training job id, training job status, \n updated model id)

opt

con -> aif: <<R1>> retrieve model request (rApp id, updated model id)

con <- aif: <<R1>> retrieve model response (model location details)

end

@enduml

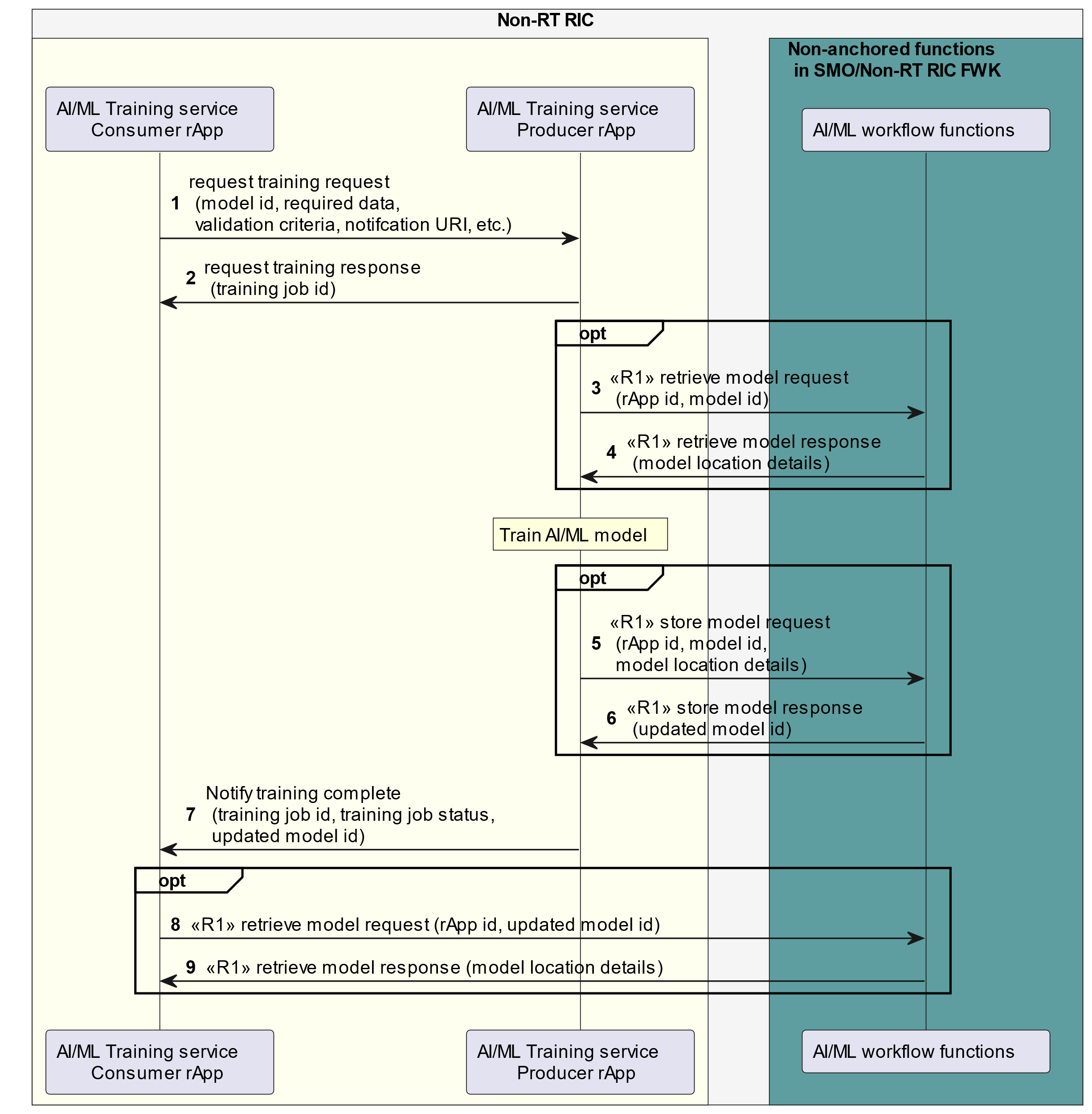


Figure 10.4.3-1: Example workflow of AI/ML model storage and retrieval

Annex (informative):   
Change History

|  |  |  |
| --- | --- | --- |
| Date | Revision | Description |
| 2024.07.17 | 06.00 | Address the received editorial comments |
| 2024.07.09 | 05.00.07 | Implemented CRs:  INT-2024.06.20-WG2-CR-0083-NonRT RIC Clarification in architecture description-v02.docx  INT-2024.07.02-WG2-CR-0085-NonRT RIC hanging paragraphs and other fixes-v01.docx |
| 2024.06.25 | 05.00.06 | Implemented CRs:  ERI.AO-2024.10.06-WG2-CR-0327-Non-RT RIC-AIML workflow services function inference-v02.docx  ERI.AO-2024.05.31-WG2-CR-0318-Non-RT RIC-R1InterfaceUpdate-v03.docx  INT-2024.06.14-WG2-CR-0082-NonRT RIC Remove EN of performance monitoring-v01.docx |
| 2024.06.14 | 05.00.05 | Implemented CRs:  INT-2024.02.16-WG2-CR-0071-E2E flow for A1 policy creation with a Near-RT RIC id-v04.docx  ERI-2024.06.06-WG2-CR-0324-Non-RT RIC-R1 Service manangement and exposure-v01.docx |
| 2024.05.16 | 05.00.04 | Implemented CRs:  NEC.AO-2023.11.22-WG2-CR-0046-Non RT RIC ARCH-AIML model deployment requirement-v3.docx  ERI-2024.03.20-WG2-CR-0291-Non-RT RIC -Updates to R1 Termination-v04.docx |
| 2024.04.03 | 05.00.03 | Implemented CR:  INT.AO-2024.03.11-WG2-CR-0073-Non-RT RIC TS AIML performance monitoring update-v03.docx |
| 2024.03.11 | 05.00.02 | Implemented CRs:  NOK-2024.02.14-WG2-CR-0129-NonRT RIC ARC\_producer consumer spelling\_v01.docx  INT-2024.02.15-WG2-CR-0068-Update Non-RT RIC clause 9-v01.docx  INT-2024.02.16-WG2-CR-0069-Update references in Non-RT RIC-v01.docx  INT-2024.02.15-WG2-CR-0065-Update Non-RT RIC clause 4-v02.docx  CMCC.AO-2023.05.30-WG2-CR-0017-AIML model training capability discover services-v3.docx  NOK-2024.02.20-WG2-CR-0138-NonRT RIC ARCH\_NearRT RIC selection optional\_v01.docx  NOK-2024.02.15-WG2-CR-0135-NonRT RIC ARCH\_AuthN AuthZ functionality placement\_v03.docx |
| 2024.02.07 | 05.00.01 | Implemented CRs:  INT-2023.12.28-WG2-CR-0064-Update notes on anchored and non-anchored functions-v01.docx  NEC.AO-2023.11.22-WG2-CR-0049-Non RT RIC ARCH-AIML model registration requirement-v3.docx  NEC.AO-2023.11.22-WG2-CR-0048-Non RT RIC ARCH-AIML model discovery requirement-v3.docx  ERI-2024.01.25-WG2-CR-0258-Non-RT-RIC-Arch-update on the DME Requirement-v02.docx |
| 2023.11.16 | 05.00 | Address the received editorial comments |
| 2023.11.10 | 04.00.04 | Implemented CRs:  NOK-2023.10.30-WG2-CR-0115-NonRT RIC ARCH-DME type terminology cleanup\_v01.docx  INT-2023.11.01-WG2-CR-0062-Editable reference architecture figure-v01.docx |
| 2023.11.07 | 04.00.03 | Implemented CRs:  NOK-2023.10.17-WG2-CR-0110-NonRT RIC ARCH-Update rApp instance definition\_v01.docx  INT.AO-2023.10.10-WG2-CR-0056-Non-RT RIC ARCH TS ODR update-v03.docx  INT-2023.10.19-WG2-CR-0058-rApp registration from SME to rApp management-v01.docx  INT.AO-2023.10.20-WG2-CR-0060-Non-RT RIC ARCH TS ODR update part 2-v01.zip  INT.AO-2023.10.19-WG2-CR-0059-ML model terms-v01.docx |

|  |  |  |
| --- | --- | --- |
| 2023.10.17 | 04.00.02 | Implemented CR:  INT.AO-2023.09.27-WG2-CR-0054-rAppId definition in Non-RT RIC Arch-v03.docx |
| 2023.08.31 | 04.00.01 | Implemented CR:  NEC.AO-2023.08.02-WG2-CR-0040-Non RT RIC ARCH-AIML model deployment services-v4.docx |
| 2023.07.28 | 04.00 | Address the received editorial comments |