O-RAN.WG2.Non-RT-RIC-ARCH-TR-v01.01

Technical Report

O-RAN Working Group 2 Non-RT RIC: Functional Architecture

O-RAN ALLIANCE e.V.
Buschkauler Weg 27, 53347 Alfter, Germany
Register of Associations, Bonn VR 11238
VAT ID DE321720189

Revision History

Date	Revision	Author	Description
2020.04.10	01.00.00	CMCC, Intel	Document skeleton
2020.05.07	01.00.01	Intel, CMCC	Addressed comments from email discussion Captured agreements made in WG2 weekly meeting on May 07th, 2020
2020.06.17	01.00.02	Intel, CMCC	Following three CRs are endorsed *ATT.AO-2020.05.05-WG2-CR-002-RAPPS-INTF-Non-RT-RIC-v03.docx* *INT.AO-2020.06.11-WG2-CR-0001-Non-RT RIC Functional Architecture Diagram-v05.docx* *INT.AO-2020.06.03-WG2-CR-0002-Skeleton update-v01.docx*
2020.10.29	01.00.03	Intel, CMCC	Following seven CRs are endorsed *SAM.AO-2020.07.23-WG2-CR-0002-Non-RT-RIC-ARCH-TR-v02.docx* *INT.AO-2020.10.01-WG2-CR-0005-A1 termination and funtions-v03.docx* *ATT.AO-2020.07.22-WG2-CR-003-R1-Intf-Reqts-v15.docx* *SAM.AO-2020.10.21-WG2-CR-0003-Non-RT-RIC-ARCH-TR-v02.docx* *INT.AO-2020.10.21-WG2-CR-0008-ML training host and model repository-v03.docx* *INT.AO-2020.10.29-WG2-CR-0009-Clean up-v01.docx* *NOK-2020.10.28-WG2-CR-0001-Non-RT-RIC Functional Architecture Scope-v02.docx*
2020.10.30	01.00.04	Intel, CMCC	Removed Annex A: Deployment Scenarios Corrected some cross references
2020.11.02	01.00.05	CMCC, Intel	Following CR is endorsed *CMCC.AO-2020.10.15-WG2-CR-0003-Non-RT-RIC-ARCH-TR-v03.docx* Editorial updates with removing the editor notes.
2020.11.16	01.00	CMCC, Intel	Addressed editorial updates in the following document *WG2-Non-RT RIC ARCH TR-v01.00.05-review comments-INT_ERI_NOK_JIO.xlsx*
2020.12.13	01.01.01	Intel, CMCC	Following CR is endorsed *IBM.AO-2020.07.15-WG2-CR-0004-Non-RT RIC Functional Architecture Diagram-v06.docx*
2021.01.28	01.01.02	Intel, CMCC	Following CR is endorsed *INT.AO-2020.12.16-WG2-CR-0010-Human-Machine Interface for Training-v01.docx*

2021.03.05	01.01.03	Intel, CMCC	Following twelve CRs are endorsed *NEC.AO-2021.02.03-O-RAN-CR-0002- Definition of function in Non-RT RIC ARCH-v03.docx* *NOK-2021.02.03-WG2-CR-0004-mapping-SBA-vs-functional-arch-v05.docx* *NOK-2020.12.07-WG2-CR-0001-Non-RT-RIC-Functional-Architecture-SBA-v07.docx* *NOK-2021.01.19-WG2-CR-0003-R1-services-v07.docx* *NOK-2021.02.09-WG2-CR-0006-Integration-services-v03.docx* *NOK-2021.02.03-WG2-CR-0007-non-RT-RIC-FWK-in-SB-view-v02.docx* *NEC.AO-2021.02.09-O-RAN-CR-0003- SBA-based Non-RT RIC ARCH-v01.docx* *NOK-2021.02.09-WG2-CR-0005-harmonizing-functionalities-functions-services-v04.docx* *NOK-2021.03.02-WG2-CR-0010-title-of-Section-5.4.3-v01.docx* *NOK-2021.03.02-WG2-CR-0011-title-of-Section-5.4.2-v01.docx* *NOK-2021.03.01-WG2-CR-0008-harmonizing-AI-ML-functionalities-v03.docx* *NOK-2021.03.02-WG2-CR-0009-Integration-services-in-the-R1-interface-general-principles-v02.docx*
2021.03.12	01.01	Intel, CMCC	Addressed editorial comments in the following document *WG2-Non-RT-RIC-ARCH-TR-v01.01.03-review comments-JIO-Intel.xlsx*

Contents

1 Introduction

1.1 Scope

This Technical Report has been produced by 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 release date and an increase in version number as follows:

Release xx.yy.zz

where:

xx the first two-digit value is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc. (the initial approved document shall have xx=01).

yy the second two-digit value is incremented when editorial only changes have been incorporated in the document.

zz the third two-digit value is included only in working versions of the document indicating incremental changes during the editing process; externally published documents never have this third two-digit value included.

The present document provides the technical report for the overall functional architecture of the Non-RT RIC (RAN Intelligent Controller). This document collects concepts and requirements that are intended to be used in the subsequent specification of the Non-RT RIC architecture. It is not intended to define normative requirements for the conformance of implementations.

1.2 References

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

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

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

[2] O-RAN-WG3.E2GAP, “O-RAN Working Group 3, Near-Real-time RAN Intelligent Controller, E2 General Aspects and Principles”.

[3] O-RAN-WG3.E2AP, “O-RAN Working Group 3, Near-Real-time RAN Intelligent Controller, E2 Application Protocol (E2AP)”.

[4] O-RAN-WG1.OAM Architecture, “O-RAN Operations and Maintenance Architecture”.

[5] O-RAN-WG1.O1-Interface, “O-RAN Operations and Maintenance Interface Specification”.

[6] O-RAN-WG2.A1.UseCaseRequirements, “O-RAN Working Group 2, Use Case and Requirements”.

[7] O-RAN-WG2.A1.GA&P, "O-RAN Working Group 2, A1 interface: General Aspects and Principles".

[8] O-RAN-WG2.A1.AP, "O-RAN Working Group 2, A1 Interface: Application Protocol".

[9] O-RAN-WG2.A1.TP, "O-RAN Working Group 2, A1 Interface: Transport Protocol".

[10] O-RAN-WG1.O-RAN Architecture, "O-RAN Working Group 1, O-RAN Architecture Description".

[11] O-RAN-WG2.O-RAN Architecture, "O-RAN Working Group 2, AI/ML Workflow description and Requirements".

[12] ETSI GS ZSM 002, "Zero-touch network and Service Management (ZSM); Reference Architecture".

1.3 Definitions and Abbreviations

1.3.1 Definitions

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

Non-RT RIC(O-RAN non-real-time RAN Intelligent Controller): a logical function 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 (e.g. UE basis, Cell basis) data collection and actions over E2 interface.

O-CU: O-RAN Central Unit: a logical node hosting RRC, SDAP and PDCP protocols.

O-CU-CP: O-RAN Central Unit – Control Plane: a logical node hosting the RRC and the control plane part of the PDCP protocol.

O-CU-UP: O-RAN Central Unit – User Plane: a logical node hosting the user plane part of the PDCP protocol and the SDAP protocol.

O-DU: O-RAN Distributed Unit: a logical node hosting RLC/MAC/High-PHY layers based on a lower layer functional split.

O-RU: O-RAN Radio Unit: a logical node hosting Low-PHY layer and RF processing based on a lower layer functional split. This is similar to 3GPP's "TRP" or "RRH" but more specific in including the Low-PHY layer (FFT/iFFT, PRACH extraction).

O-eNB (O-RAN eNB): an eNB or ng-eNB that supports E2 interface.

O1: Interface between orchestration & management entities (Orchestration/NMS) and O-RAN managed elements, for operation and management, by which FCAPS management, Software management, File management and other similar functions shall be achieved.

SMO: Service Management and Orchestration system.

A1: Interface between Non-RT RIC and Near-RT RIC to enable policy-driven guidance of Near-RT RIC applications/functions, and support AI/ML workflow.

E2: Interface connecting the Near-RT RIC and one or more O-CU-CPs, one or more O-CU-UPs, and one or more O-DUs.

E2 Node: a logical node terminating E2 interface. In this version of the specification, ORAN nodes terminating E2 interface are:

- for NR access: O-CU-CP, O-CU-UP, O-DU or any combination as defined in [4];
- for E-UTRA access: O-eNB.

rApp: An application designed to run on the Non-RT RIC. Such modular application leverages the functionality exposed by the Non-RT RIC to provide added value services relative to intelligent RAN optimization and operation

xApp: An application designed to run on the Near-RT RIC. Such an application is likely to consist of one or more microservices and at the point of on-boarding will identify which data it consumes and which data it provides. The application is independent of the Near-RT RIC and may be provided by any third party. The E2 enables a direct association between the xApp and the RAN functionality.

O-Cloud: O-Cloud is a cloud computing platform comprising a collection of physical infrastructure nodes that meet O-RAN requirements to host the relevant O-RAN functions (such as Near-RT RIC, O-CU-CP, O-CU-UP, and O-DU), the supporting software components (such as Operating System, Virtual Machine Monitor, Container Runtime, etc.) and the appropriate management and orchestration functions.

1.3.2 Abbreviations

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

EI	Enrichment Information
LCM	Lifecycle Management
ML	Machine Learning
Non-RT RIC	Non-real-time RAN Intelligent Controller
Near-RT RIC	Near-real-time RAN Intelligent Controller
RAN	Radio Access Network
SLA	Service Level Agreement
SMO	Service Management and Orchestration
TLS	Transport Layer Security
ZSM	Zero-touch network and Service Management

2 Overview of Non-RT RIC

2.1 Non-RT RIC in O-RAN Overall Architecture

In the O-RAN Logical Architecture, as seen in Figure 2.1-1 below, the Non-RT RIC is portrayed as a function that resides within the Service Management and Orchestration Framework and which has a direct association with the A1 interface. The Non-RT RIC's position internal to, as opposed to interfacing with, the SMO is intended to communicate that the Non-RT RIC is comprised of a subset of functionality of the SMO itself.

The direct association with the A1 interface is intended to convey that the functionality of the Non-RT RIC is directly responsible for driving that which is sent and received across the A1 interface. The O1 and O2 interfaces are shown as being directly associated with the SMO Framework itself, that which is not specific to the Non-RT RIC. This representation is intended to communicate that the SMO Framework functionality, but not the Non-RT RIC functionality therein, is directly responsible for driving that which is carried across these interfaces.

The representation of the Non-RT RIC in the center (as opposed to the bordering the bottom edge) of the SMO Framework visibly indicates that the Non-RT RIC can access SMO Framework functionality, including influencing that which is carried over the O1 interface. While it is not prohibited from accessing any SMO Framework functionality, the Non-RT RIC's role of RAN resource optimization per current O-RAN definition implies that the Non-RT RIC would only access SMO Framework functionality for that purpose. Thus, the Non-RT RIC would influence that which is carried across the O2 interface only to the extent that the O-Cloud is considered a RAN resource.

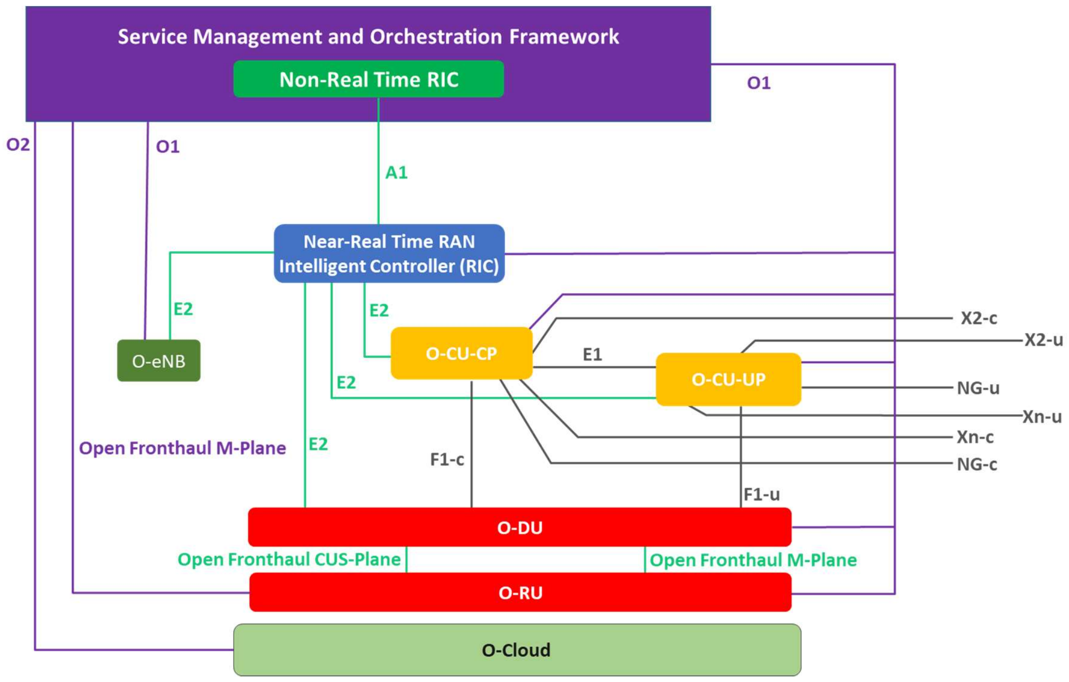


Figure 2.1-1: Logical Architecture of O-RAN

Similar to the Near-RT RIC, there is business value in visualizing the Non-RT RIC as being modularly extensible, that functional extensibility being accomplished through modular applications that can be understood as running within the Non-RT RIC function itself. Such applications have different lifecycles from the "framework" in which they run. For convenience we will use the term "rApp" to refer to such applications.

Because rApps are associated with the Non-RT RIC, any framework services exposed to the rApps can be considered as being exposed by the Non-RT RIC. Some of these framework services, for example access to O1, are not specific to the Non-RT RIC, whereas others, such as access to A1, are specific to the Non-RT RIC.

There is a clear business benefit to allow "porting" of a Non-RT RIC application from one SMO Framework implementation to another. Hence it would be useful for the O-RAN Alliance to define an open and standard interface through which the Non-RT RIC exposes SMO Framework functionalities to "rApps" via the R1 Services exposure functionality. We will refer to this as the "R1" interface.

With the above understandings, the Non-RT RIC itself can now be described as follows:

- Non-RT RIC – That functionality that drives the content carried across the A1 interface to the RAN. It can also access other SMO Framework functionality, for example influencing that which is carried across the O1-CM interface. It is comprised of two sub-functions:
    - Non-RT RIC Framework – That functionality internal to the SMO Framework itself that:
        - Logically terminates the A1 interface to the Near-RT RIC.

        - Exposes to rApps, via its R1 interface, the set of internal SMO Framework services needed for their runtime processing.
    - Non-RT RIC Applications (rApps) – Modular applications that leverage the functionality exposed by the Non-RT RIC Framework to provide added value services relative to RAN operation. Examples of such added value services include:
        - Driving content (Policy or Enrichment Information) across the A1 interface.
        - Recommending content for the O1 interface.
        - Generating "enrichment information" for the use of other rApps.

Figure 2.1-2 graphically illustrates these definitions.

As observed above, rApps will require services from the R1 interface that are not necessarily associated with the Non-RT RIC Framework, such as access to O1, data sharing services, and access to RAN inventory. The Non-RT RIC Framework, via its R1 Services exposure functionality, would be responsible for exposing all of these services to the rApps, irrespective of the location within the SMO Framework of the provider of those services.

A declared business need to "port" a Non-RT RIC Framework from one SMO Framework implementation to another would drive the need for a formal interface within the SMO Framework between the Non-RT RIC Framework and the rest of the SMO Framework. Absent this, different implementations of SMO Framework could make different design choices as to how to manifest the boundary between the Non-RT RIC Framework and the rest of the SMO Framework, or even to choose not to implement a clear boundary at all.

The lack of a clear boundary between the Non-RT RIC and the rest of the SMO Framework need not negatively impact the ability to proceed with a formal definition of the R1 interface nor impact definition of the R1 Services exposure functionality needed to support this interface. Rather, such definitions can proceed, agnostic to the source of the SMO Framework services being exposed. However, it may be useful to define some terms that we can use to help us in discussion of these topics.

Particularly it would be useful to have terms to distinguish between:

- An indication that a particular R1-exposed functionality is sourced from the Non-RT RIC Framework. This functionality would be considered "inherent" to the Non-RT RIC Framework.
- An indication that it is left as an SMO Framework implementation decision whether a particular R1-exposed functionality is sourced from the Non-RT RIC Framework or not.
- An indication that a particular R1-exposed functionality is not sourced from the Non-RT RIC Framework. This functionality would be considered "inherent" to the SMO Framework itself, but not the Non-RT RIC Framework.

The functionality that is inherent to the Non-RT RIC Framework is that functionality related to the two interfaces that the Non-RT RIC "owns", specifically that functionality needed to drive the A1 and R1 interfaces. The O1 and O2 interfaces are clear examples of functionality of the third bullet.

The term "Implementation Variable Functionality" is descriptive of that functionality described in the second bullet. Examples might include functionalities such as "Data Sharing", "Analytic Services", "Policy", "RAN Inventory", "General ML/AI Model LCM/Catalog", RAN Mediation/Control", "RAN Configuration Database", "PM Short Term Data Repository", "Operations Intent Interface" and "External Interfaces".

These terms and the relationships between the functions they describe can be seen in Figure 2.1-2 below.

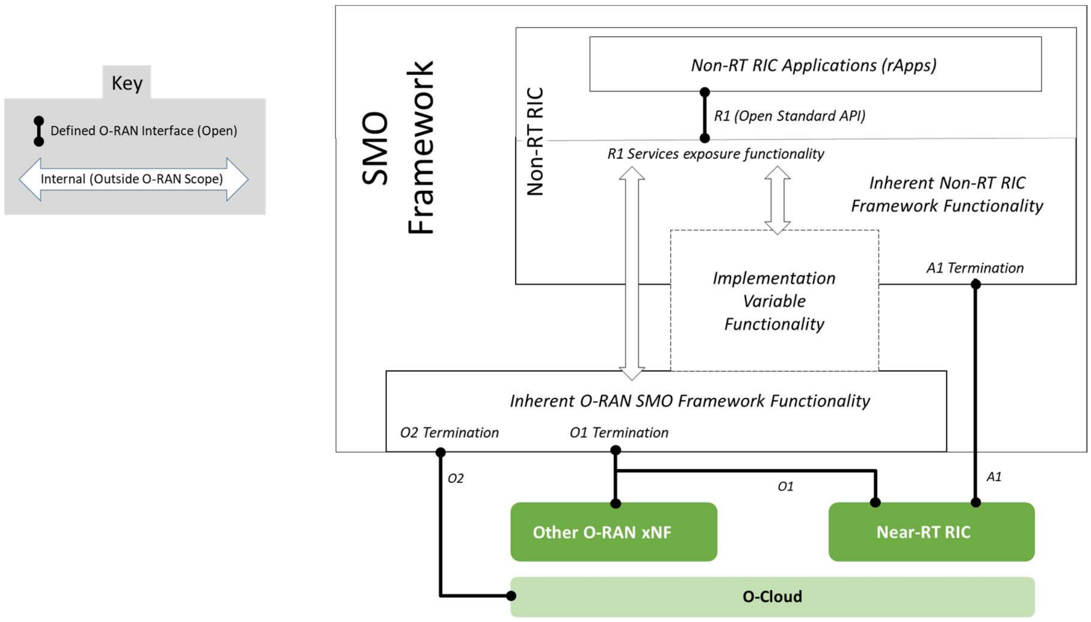

Key
Defined O-RAN Interface (Open)
Internal (Outside O-RAN Scope)
SMO Framework
Non-RT RIC
Non-RT RIC Applications (rApps)
R1 (Open Standard API)
R1 Services exposure functionality
Inherent Non-RT RIC Framework Functionality
Implementation Variable Functionality
A1 Termination
Inherent O-RAN SMO Framework Functionality
O2 Termination
O1 Termination
O2
O1
A1
Other O-RAN xNF
Near-RT RIC
O-Cloud

Figure 2.1-2: Exposure of SMO Framework Services to rApps

The "Internal" arrows in the diagram represent that it is within the scope of the Non-RT RIC Framework's R1 Services exposure functionality to expose all required services of the SMO Framework, even those that are not or may not be associated with the Non-RT RIC Framework itself. It is outside of O-RAN's scope to define the "Internal" mechanism through which these services are exposed to specific SMO Framework implementations.

Note that each of the endpoint functional blocks of the SMO Framework's external interfaces have interface termination functions and functionalities associated with them to drive that respective interface. Thus, the Non-RT RIC Framework has associated with it the "A1 Functionalities", i.e., A1 Policy, A1 EI and A1 ML, required to drive the A1 interface, as well as the A1 interface logical termination functions. Examples of R1 Services exposure functionality would include what is needed to manage the runtime interactions of individual rApps with the Non-RT RIC Framework itself.

The Non-RT RIC architecture lends itself to be described by using two views, based on different approaches: a functional approach and a service-based approach. In the functional approach, the architectural view focuses on "packaging" functionalities into functions, i.e., logical entities with well-defined behaviour, connected by interfaces. In a service-based approach, the architectural view focuses on describing the functionalities as capabilities offered through services by service producer entities to service consumer entities, inspired by the service-based design principles defined in ETSI ZSM [12]. Service consumer and service producer entities are interconnected by means of communication support that enable adequate communication patterns e.g. 1-1, 1-many, pub-sub, routed etc. In general, a functional architectural view can be regarded as a deployment option, or specific realization, allowed by the service-based architectural view, and both provide the same functionalities. As described in the following, this principle is valid also for the Non-RT RIC case.

Following the principles above, Section 2.2 provides the functional view of the Non-RT RIC architecture, whereas Section 2.2a shows the service-based view. The functional decomposition of the Non-RT RIC shown in Section 2.2 facilitates analysis and description of the required functionality of the combination of Non-RT RIC and SMO framework. Based on that analysis, the service-based view of the Non-RT RIC architecture focuses on interoperability aspects – namely the interfaces with other O-RAN components and the R1 services that allow interworking between the Non-RT RIC and the rApps as the main multivendor integration point. To facilitate flexibility of deployment, the service-based architectural view defines SMO and Non-RT RIC capabilities, but abstracts from a detailed representation of functional blocks that realize the capabilities and produce or consume the R1 services. This architectural approach allows multiple different instantiations; the detailed functional view presented in Section 2.2 being one of these possible instantiations. All such instantiations will all be interoperable towards rApps as long as they support the R1 services, and towards other O-RAN system components as long as they support the interfaces defined towards these.

2.2 Non-RT RIC Architecture Functional View Diagram

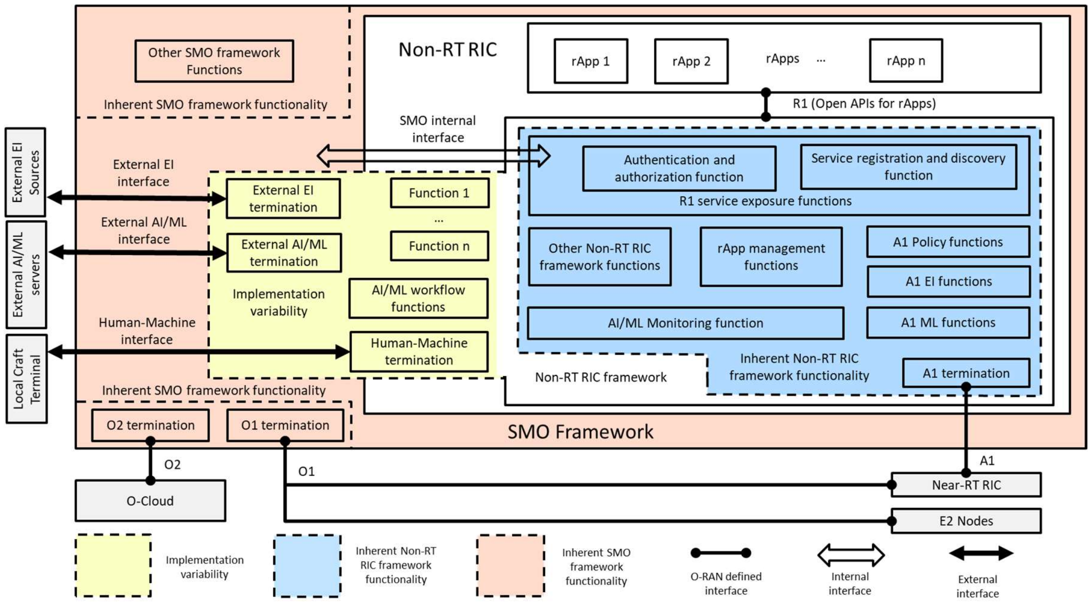

Other SMO framework Functions
Inherent SMO framework functionality
Non-RT RIC
rApp 1
rApp 2
rApps ...
rApp n
R1 (Open APIs for rApps)
SMO internal interface
External EI Sources
External EI interface
External AI/ML servers
External AI/ML interface
Local Craft Terminal
Human-Machine interface
External EI termination
External AI/ML termination
Function 1
...
Function n
Implementation variability
AI/ML workflow functions
Human-Machine termination
Authentication and authorization function
Service registration and discovery function
R1 service exposure functions
Other Non-RT RIC framework functions
rApp management functions
A1 Policy functions
A1 EI functions
AI/ML Monitoring function
A1 ML functions
Non-RT RIC framework
Inherent Non-RT RIC framework functionality
A1 termination
Inherent SMO framework functionality
O2 termination
O1 termination
SMO Framework
O2
O1
A1
O-Cloud
Near-RT RIC
E2 Nodes
Implementation variability
Inherent Non-RT RIC framework functionality
Inherent SMO framework functionality
O-RAN defined interface
Internal interface
External interface

Figure 2.2-1: Non-RT RIC architecture functional view diagram

Figure 2.2-1 illustrates the Non-RT RIC architecture functional view diagram. Non-RT RIC is an internal functionality of SMO framework, and the diagram shows three categories of components of Non-RT RIC: rApps, Non-RT RIC framework, and Open APIs for rApps.

- rApp is an application designed to run on the Non-RT RIC, and it is defined in Section 2.3.
- Non-RT RIC framework is a collection of Non-RT RIC framework functions, and it is described in Section 2.4.
- R1 interface (Open APIs for rApps) are Non-RT RIC internal interface between rApps and Non-RT RIC framework, and it is a collection of services, such as service registration and discovery services, AI/ML workflow services, and A1-related services. Note that whether the function that provides the services is in Non-RT RIC framework or in SMO framework is transparent to the open APIs. R1 interface general principles and R1 services are discussed in Section 3.4.

Note that one Non-RT RIC can connect to multiple Near-RT RICs.

A function is a logical entity that plays the roles of services producer and/or service consumer.

Inside the Non-RT RIC framework, there is a set of essential Non-RT RIC framework functions, which is illustrated as the blue area in the diagram. Because these functions are used to support A1 interface and rApps, it is nature to deploy these functions inside Non-RT RIC framework. Such functions are denoted as “inherent Non-RT RIC framework functionality”. As the termination point of A1 interface, “A1 functions” should be regarded as inherent Non-RT RIC functions. “A1 functions” includes “A1 logical termination” and functions to support A1 services defined in [7] [8], i.e., “A1 Policy functions”, “A1 EI functions”, and “A1 ML functions”.

To facilitate modular rApps, “rApp management functions” and “R1 service exposure functions” are required. “rApp management functions” includes rApp conflict mitigation. This example of rApp management is regarded as inherent to Non-RT RIC framework. Note that functions for rApp orchestration are not part of rApp management functions, and rApp orchestration functions can be part of SMO framework. In the diagram, “other Non-RT RIC framework functions” is used as a placeholder for any other functions inside Non-RT RIC, if identified later during the study. “R1 service exposure functions” includes “service registration and discovery function” and “authentication and authorization function", etc. The services provided by Non-RT RIC framework and SMO framework are discovered by rApps via "services registration and discovery function". An rApp needs to be authenticated and authorized by the authentication and authorization function before being able to access the required services.

As discussed in the previous section, functions can fall into the "implementation variability" area, shown as the "yellow" area in the diagram. The deployment of these "implementation variable" functions is left to implementation. If an "implementation variable" function is deployed in the Non-RT RIC in a particular implementation, then this function is regarded as a part of Non-RT RIC framework, and its service is discovered by rApps via "service registration and discovery function". On the other hand, if an "implementation variable" function is deployed in the SMO framework in one implementation, then this function is not a part of Non-RT RIC framework, and its service is also discovered by rApps via "service registration and discovery function".

In Figure 2.2-1, three terminations of external interfaces are demonstrated as examples of "implementation variable" functions: external EI termination, external AI/ML termination, and human-machine termination. External EI termination is connected to external EI sources to import enrichment information for Non-RT RIC applications. External AI/ML termination is connected to external AI/ML server for ML model importation. Human-machine termination is used to inject RAN intent manually. All three terminations can reside inside Non-RT RIC or in SMO (but outside of Non-RT RIC). Note that specification of these external interfaces is for further discussion. There are many more "implementation variable" functions, and examples for those "implementation variable" functions include: AI/ML model training, data analytics, data sharing, etc. In the diagram, "Function 1", …, "Function n" are used as placeholders for more "implementation variable" functions.

The functions depicted in the architecture in Figure 2.2-1 provide the functionalities and services defined in Section 3.

2.2a Non-RT RIC Architecture Service-based View

The service-based architectural approach is today widely adopted by telco industry. Service-based principles allow to describe an architecture not based on fixed functional components and interfaces between them but based on the definition of services as the central point of architecture consideration. Though there's no formal definition available, a service-based architecture follows principles of which some are listed below:

- **Modularity**: The functionality is modularized into a set of services at the appropriate granularity level.
- **Extensibility**: The architecture allows simple extensibility by adding new services and making them discoverable.
- **Functional abstraction**: Service-based architecture abstracts from the complexity and details of the underlying functions that produce the services.
- **Discoverability**: Services are searchable for their availability.
- **Composability**: Services can be composed, forming new services.
- **Reusability**: Since services are modularized, they can be re-used and can be invoked at multiple stages of the business process.
- **Loose coupling**: Services are loosely coupled and independent from one-another.

Designing an architecture in a service-based fashion allows a great degree of deployment *flexibility* and *future-proofedness*, as it leaves the choice of components that produce and/or consume certain services to the deployment, yet facilitating multi-vendor interoperability through the definition of standardized services respectively standardized service interfaces. Services are produced by service producers and consumed by service consumers through service endpoints. Service-based architecture allows *modularization* of the services rather than the components of a deployment. Service consumers need to be *authenticated* and *authorized* to access services at runtime by policies, rather than pre-assigning fixed consumer-producer relationships at the time the standard is developed. Produced services are *registered* in a registry from which service consumers can *discover* the available services, and the technical parameters how to consume them, such as the service endpoint, the data format and access protocol. For the service communication, different possibilities exist. Service consumers can access the service endpoints directly in a point-to-point fashion, after discovering them. Alternatively, service-related communication can be facilitated using an overarching communication framework such as a message bus or service mesh. Using a service-based design for the functionalities that are consumed and/or produced by rApps, these apps can run in any deployment that offers these services, without the need to document implementation variability. Also, future evolution of rApps into more generic smart network management apps is enabled, as such evolved apps would simply be able to discover and use more services.

2.2a.1 Non-RT RIC Architecture Service-based View Diagram

The diagram in Figure 2.2a-1 shows the decomposition of the Non-RT RIC into Non-RT RIC Framework and rApps, interacting with each other through the R1 services, described in Section 3.4.

A service is a set of capabilities offered by a service producer to service consumers for consumption through defined endpoints. The purpose of a service is to access, use and control a certain functionality (set of capabilities) available in the system. Production and consumption of services, which are specified, are roles of (software) entities in deployments, however, the exact software entities are not called out in the specification. A service may be standardized by O-RAN Alliance, by another organization, or be proprietary, but it can be made available in the system if it implements a common and standard way to register, to be accessed and controlled.

According to the above, the architecture in Figure 2.2a-1 depicts the R1 services and the related capabilities which are part of the SMO and Non-RT Framework. Such capabilities also include communication support to enable adequate communication patterns for the service requirements, e.g., 1-1, 1-many, pub-sub, routed etc., inspired by the service-based design principles defined in [12]. Furthermore, the diagram shows that the service-based approach describes consistently how services provide access to Non-RT RIC inherent functionalities, SMO inherent functionalities, implementation variable functionalities and also capabilities external to the SMO.

The Non-RT RIC service-based architectural view caters for different deployment options, including the one illustrated in Figure 2.2-1, by applying an appropriate configuration of the authorized producers-consumer pairs.

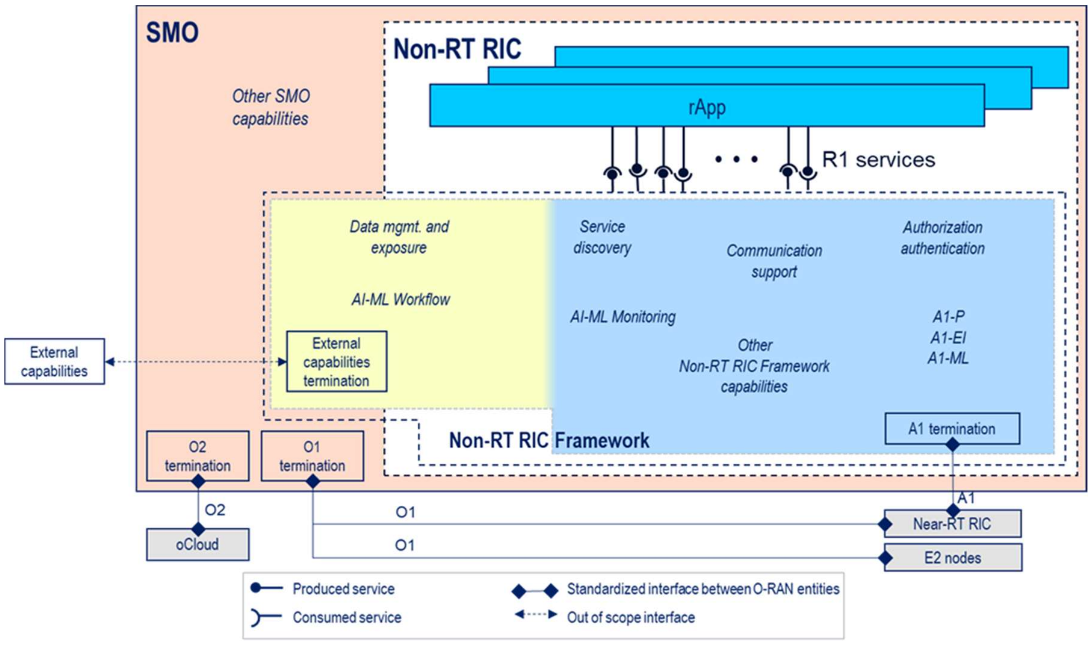


Figure 2.2a-1: Non-RT RIC architecture service-based view

2.3 rApp Definition

rApps are modular applications that leverage the functionality exposed by the Non-RT RIC to provide added value services relative to intelligent RAN optimization and operation. Examples of such added value services include:

- Providing policy-based guidance and enrichment information across A1 interface.
- Performing data analytics, AI/ML training, and inference for RAN optimization or for the use of other rApps.
- Recommending configuration management actions over O1 interface.

2.4 Non-RT RIC Framework

2.4.1 Non-RT RIC Framework in Functional View

In the functional view of the Non-RT RIC architecture in Section 2.2, the Non-RT RIC framework is a collection of Non-RT RIC framework functions. It includes the set of inherent Non-RT RIC framework functions to support A1 interface and rApps, other Non-RT RIC framework functions if any, and "implementation variable" functions which are deployed in Non-RT RIC. Non-RT RIC framework functions provide services to rApps via the open APIs (also referred to as R1 services, described in Section 3.4). In summary, Non-RT RIC framework functions include

- "rApp supporting functions", e.g, rApp service exposure functions, rApp conflict mitigation, etc.
  - To support interoperability between Non-RT RIC and SMO, Non-RT RIC and SMO can expose their rApp-supporting functions to each other through SMO service exposure function, which is part of rApp service exposure functions. Non-RT RIC can leverage SMO services, for example, data collection and provisioning services, via SMO service exposure function.
- "A1 functions", e.g., A1 logical termination, A1-Policy coordination and catalog, A1-EI coordination and catalog, etc.
- "AI/ML Monitoring functions",
  - Providing online monitoring functions to AI/ML models in Non-RT RIC. For example, it can adopt a contract-based strategy for run-time monitoring of AI/ML models to support different types of AI/ML algorithm in multi-vendors scenarios.
- "AI/ML workflow functions", e.g., "AI/ML Model Management Functions", "AI/ML Data Preparation Functions", "AI/ML Modeling/Training Functions", "ML Model Repository". AI/ML workflow functions are implementation variability functions and can be flexibly deployed within Non-RT RIC, outside Non-RT RIC but within SMO, or even outside SMO.
- Other logical terminations, e.g., external EI termination, external AI/ML termination, human-machine termination, etc.
- Other "implementation variable" functions, e.g., data sharing.

2.4.2 Non-RT RIC Framework in a Service-based View

In the service-based view of the Non-RT RIC architecture of Section 2.2a, the Non-RT RIC framework is a collection of capabilities to support well defined functionalities. It includes the set of inherent Non-RT RIC framework functionalities for the A1 interface, the support for integration services, other Non-RT RIC framework functionalities if any, and "implementation variable" functionalities which are deployed in Non-RT RIC. Non-RT RIC framework functionalities provide services to rApps via the open APIs (also referred to as R1 services, described in Clause 3.4). In summary, Non-RT RIC framework functionalities include:

- *Integration services*, providing interoperability between rApps and Non-RT RIC/SMO. The Non-RT RIC Framework, the SMO, and potentially the rApps too can expose their services through a set of capabilities in the Non-RT RIC Framework that enable rApps to register and discover service endpoints, and also to obtain notifications whenever a change in the service availability occurs. In addition, integration services include capabilities to authenticate and authorize the service producers and the service consumers.
- *Data management and exposure* enable the collection and production of data, as well as fetching data from a database. These functionalities cater for having different sources of heterogeneous data which can be delivered directly to a data consumer or through a distribution system. More details are elaborated in Section 3.4.3. Data management and exposure are implementation variability functionalities and can be flexibly deployed within Non-RT RIC, outside Non-RT RIC but within SMO.
- *A1 interface support*, e.g., A1 logical termination, A1-Policy coordination and catalogue, A1-EI coordination and catalogue, etc.
- *AI/ML monitoring*, providing online monitoring to AI/ML models in Non-RT RIC. For example, it can adopt a contract-based strategy for run-time monitoring of AI/ML models to support different types of AI/ML algorithm in multi-vendors scenarios.
- *AI/ML workflow support*, e.g., AI/ML model management and storage, AI/ML data preparation, AI/ML modelling/training. AI/ML workflow functionalities are implementation variability functionalities and can be flexibly deployed within Non-RT RIC, outside Non-RT RIC but within SMO, or even outside SMO.
- Other logical terminations, e.g., external EI termination, external AI/ML termination, human-machine termination, etc.
- Other "implementation variable" functionalities, e.g., data sharing.

2.5 External Interfaces

2.5.1 A1 Interface

O-RAN-WG2.A1.GA&P [7] specifies A1 interface general aspects and principles.

O-RAN-WG2.A1.AP [8] specifies A1 interface application protocols.

O-RAN-WG2.A1.TP [9] specifies A1 interface transport protocols.

2.5.2 External EI interface

2.5.2.1 External EI interface general principles

The general principles for the functionalities of an external EI interface are as follows:

- Support registration of an external source of EI.
- Support transfer of EI from an external source to Non-RT RIC that requests it.
- Support transfer of handshaking messages (e.g. EI request and response messages) between the Non-RT RIC and an external source of EI.

2.5.2.2 External EI interface objectives

The external EI interface functionalities facilitate the following:

- Secure delivery of EI from an external source of EI to the Non-RT RIC.

- Recognition of EI type by the Non-RT RIC.
- Transfer of EI request messages from the Non-RT RIC to an external source of EI.
- Transfer of EI response messages from an external source of EI to the Non-RT RIC.

2.5.3 External AI/ML interface

2.5.3.1 External AI/ML interface general principles

The general principles for the functionalities of an external AI/ML interface are as follows:

- Support transfer of a trained ML model (and its metadata) from an external AI/ML server to SMO/Non-RT RIC that requests it.
- Support transfer of handshaking messages (e.g. training request and response messages) between the SMO/Non-RT RIC and an external AI/ML server.

Note: Transfer of training data to an external AI/ML server is FFS.

2.5.3.2 External AI/ML interface objectives

The external AI/ML interface functionalities facilitate the following:

- Secure delivery of trained ML models (and their metadata) from an external AI/ML server to the SMO/Non-RT RIC.
- Transfer of training request messages from the SMO/Non-RT RIC to an external AI/ML server.
- Transfer of training response messages from an external AI/ML server to the SMO/Non-RT RIC.

2.5.4 Human-Machine interface

The interface is provided by Non-RT RIC for technicians to manually deploy RAN intent and operate rApp .

As many systems provide Human-Machine interface (e.g., web based UI), Non-RT RIC also provides such functions to technicians for their manual operations. Examples of operations include RAN intent injection, rApp maintenance, and human interaction/intervention for AI/ML training in the SMO/Non-RT RIC.

This document doesn't limit the implementation of the interface. The interface function could also be integrated into other interfaces to form a unified interface.

Note: RAN intent may be also imported to SMO/Non-RT RIC by other ways, e.g., from the business support system.

Note: The details of RAN intent are FFS.

2.5.4.1 Human-Machine Interface general principles

The general principles for the functionalities of the Human-Machine interface are as follows:

- Support for RAN intent injection.
- Support for checking an rApp information.
- Support for suspending a running rApp.
- Support for resuming a pending rApp.

- Support for human interaction/intervention for AI/ML training.

Note: The rApp information may include it's running status, etc.

2.5.4.2 Human-Machine interface objectives

The Human-Machine interface functionalities facilitate the following:

- Transfer the RAN intent.
- Provide the information of deployed rApps on the Non-RT RIC.
- Transfer the command of suspending a specified running rApp.
- Transfer the command of resuming a specified pending rApp.
- Transfer the AI/ML training configurations.
- Transfer the AI/ML training status requests and reports.
- Transfer the commands of initiating/suspending/resuming/terminating a training process.

3 Study on Functionalities, Functions and Services

3.1 Interface termination functions

3.1.1 A1 interface termination

The A1 interface termination enables the Non-RT RIC framework to send and receive messages to and from the Near-RT RIC via the A1 interface. The A1 interface termination secures the A1 interface via TLS in A1 protocol stack [9]. A1 messages sent to, and received from, the Near-RT RIC currently include

- A1 policy creation/update request based on the policy received from the A1-P functions (see Section 3.2.1).
- A1 policy feedback, which is routed to the A1-P functions.
- A1 enrichment information from the A1-EI functions (see Section 3.2.2).
- A1 enrichment information subscription request, which is routed to the A1-EI functions.

Possible future A1 messages may include

- A1 machine learning service requests/responses and subscriptions/notifications, to and from the A1-ML functions (see Section 3.2.3).

Refer to [7, 8, 9] for details on A1 interface specifications.

3.1.2 External EI termination

The external EI interface termination enables the SMO/Non-RT RIC to receive and send messages via the external EI interface. These include, e.g., handshaking messages sent between the SMO/Non-RT RIC and an external source of EI. The delivery of EI over the external EI interface can be leveraged by rApps to enhance RAN operations. An xApp can also leverage EI that is initially delivered over the external EI interface and then to the Near-RT RIC over the A1 interface.

The external EI interface termination secures the external EI interface.

The external EI interface termination routes all incoming messages from an external source of EI to the SMO/Non-RT RIC. The external EI interface termination may decode incoming messages (e.g. ASN.1-encoded messages) from an external source of EI.

The external EI interface termination performs conversion of external data models (e.g. 5GAA standard-compliant messages) from an external source of EI to facilitate internal messaging for the SMO/Non-RT RIC.

Figure 3.1-1 shows the connection between the external EI interface termination and an external source of EI via the external EI interface.

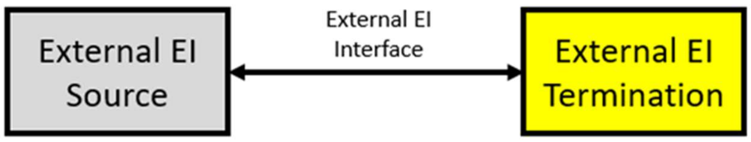


Figure 3.1-1: External EI termination

3.1.3 External AI/ML termination

The external AI/ML interface termination enables the SMO/Non-RT RIC to receive and send messages via the external AI/ML interface. These include, e.g., handshaking messages sent between the SMO/Non-RT RIC and an external AI/ML server.

The external AI/ML interface termination secures the external AI/ML interface.

The external AI/ML interface termination routes all incoming messages from an external AI/ML server to their intended destinations. For example, it can route a trained ML model to an ML model repository.

The external AI/ML interface termination may decode incoming messages (e.g. JSON-encoded messages) from an external AI/ML server.

The external AI/ML interface termination performs conversion of trained ML models (e.g. ONNX standard-compliant model formats) from an external AI/ML server to a model format that is supported by the SMO/Non-RT RIC.

Figure 3.1-2 shows the connection between the external AI/ML interface termination and an external AI/ML server via the external AI/ML interface.

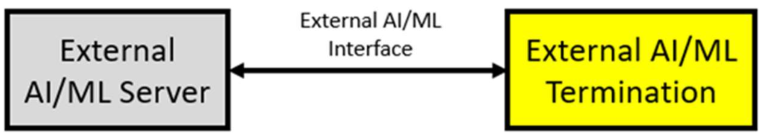


Figure 3.1-2: External AI/ML termination

3.1.4 Human-Machine termination

The Human-Machine interface enhances the closed-loop mechanism of intelligent end user service experience guarantee and network performance improvement. The interface function includes RAN intent injection, rApp maintenance, and AI/ML training interaction/intervention.

The Human-Machine interface termination secures the Human-Machine interface.

The Human-Machine termination receives information of deployed rApps from rApp management function and forwards to the technician via Human-Machine interface. The Human-Machine interface termination routes commands of suspending/resuming an rApp from a technician (e.g., with web based UI) to rApp management function.The Human-Machine interface termination may decode incoming messages (e.g. JSON/XML/YANG-encoded messages) from a technician using the portal, e.g. web based UI.

The Human-Machine termination routes AI/ML training configurations and reporting configurations/requests from technicians to rApps and/or the training host in the SMO/Non-RT RIC. It forwards commands (e.g., initiating, suspending, resuming, terminating, etc.) from a technician to allow human intervention of a specific training process. The Human-Machine termination receives AI/ML training status reports from rApps and/or the training host, and it forwards the status report to technicians.

A technician can configure a specific training process. The configuration message can contain:

- The ID for the AI/ML training process.
- The ID for the selected base model.
- The associated App ID (xApp or rApp).
- Training location (e.g., the internal/external training host, or within the application.).
- Training/validation/testing dataset description.
- Instructions for data pre-processing (e.g. normalization).
- Configuration of hyperparameters, etc.

For example, the technician can configure the training host in the SMO/Non-RT RIC to use datasets collected from dense urban area for AI/ML training, to obtain a specialized AI/ML application (or model) for metropolitan area deployment.

A technician can request and/or subscribe status reporting of a specific training process. The reporting configuration message can contain:

- The ID for the AI/ML training process.
- Reporting periodicity.
- Reporting time period.
- Reporting objects (e.g., training progress indicator, training performance indicator, etc.).
- Event-trigger conditions.
- Maximum number of reports, etc.

Based on the training status reports, a technician can send commands to intervene the training process. For example, a technician can terminate a training process, if the reporting shows the training is diverging or the training of another candidate App/model is completed. A technician can initiate a new training process after hyperparameter re-configuration.

The Human-Machine termination converts external messages from technicians to internal message formats used by the training host and/or rApps. External messages include AI/ML training configuration, reporting configuration/request, and human commands on AI/ML training operation (e.g., terminating/initiating a training process, publishing a trained model, etc.). The Human-Machine termination converts the internal training status reports from the training host and/or rApps to outgoing messages to a technician.

Note: The exact contents of AI/ML training configuration, status requests/reports, and commands are FFS.

Figure 3.1-3 shows the connection between the Human-Machine interface termination and the technician

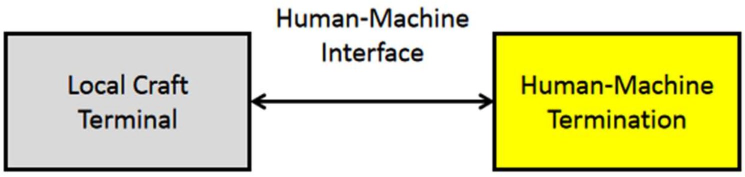


Figure 3.1-3: External Human-Machine termination

3.2 A1 Functionalities

3.2.1 A1-P functionalities

A1-P functionalities include:

- Hosting A1 policy and policy type repository.
- Processing queries from rApps on Near-RT RIC supported A1 policy types.
- Maintaining A1 policy enforcement status.
- Notifying rApps about an A1 policy enforcement status.

3.2.2 A1-EI functionalities

Some of the A1-EI functionalities include:

- Processing subscription requests for EI from the Near-RT RIC.
- Routing EI to the near-RT RIC in response to subscription requests for that EI.
- Performing EI Job Control.

3.2.3 A1-ML functionalities

Potential A1-ML functionalities may include:

- Enabling an AI/ML workflow function to register for the A1-ML service and exchange messages with the Near-RT RIC.
- Routing A1-ML service requests/responses and subscriptions/notifications between the Near-RT RIC and registered AI/ML workflow functions.

Note: A1-ML service details are FFS.

3.3 AI/ML Functionalities

AI/ML Functionalities include Non-RT RIC Framework inherent or implementation variability functionalities to support the whole AI/ML lifecycle.

3.3.1 ML Training

The ML training functionality allows the training of AI/ML models [11] of 3rd party applications (for example, rApps or xApps) within the SMO/Non-RT RIC. The inputs consist of training data and ML model. The training data can be composed of the data collected from Near-RT RIC and O-CU/O-DU over O1 interface, external enrichment information over external EI interface, and registered data output from rApps over R1 interface. The input ML model is described by the AI/ML model descriptor/metadata, which can be included in an Application package. This functionality also allows the update/re-training of ML models stored in the ML model repository functionality.

Note: Details of an AI/ML model descriptor/metadata (format, content, etc.) is FFS. Refer to Annex A.2 in [11] for examples of ML model descriptors.

The output is a trained, validated, and tested ML model, which is ready to be deployed within an rApp or xApp and which can be stored by the SMO or by the Non-RT RIC Framework using the ML model repository functionality.

The ML training in the SMO/Non-RT RIC offers offline training. Note: using the ML training in the SMO/Non-RT RIC for online training, e.g., to support online reinforcement learning, is FFS.

Note: it does not preclude the option that a 3rd party application (for example, rApp or xApp) performs online learning by itself (e.g., a reinforcement learning application).

The ML training is an “implementation variable” functionality. In the functional view of the Non-RT RIC architecture, it can be realized as an ML Training host function as part of the AI/ML Workflow functions.

3.3.2 ML Model Repository Functionality

The ML Model Repository functionality allows to store trained ML models and offers the following services to manage them:

- ML model registration: A service consumer sends a trained ML model, which can be identified by a model identifier (e.g., ModelId), and it is stored with an initial version number.
- ML model update: A service consumer updates a stored ML model. Multiple versions of a model can be stored with version numbers, e.g., for tracking purposes.
- ML model retrieval: A service consumer retrieves a ML model, providing the model identifier and the version number.
- ML model deletion: A service consumer deletes a version of a stored ML model.
- ML model deregistration: A service consumer deregisters a stored ML model, i.e., all stored versions of the model are removed.

A service consumer can provide additional supplemental information for a ML model version when it registers or updates a ML model. For example, supplemental information can include the performance of the ML model, which helps consumers to select the right model version to be retrieved. Details of the supplemental information (format, content, etc.) is FFS. Note that a service consumer needs SMO/Non-RT RIC’s authorization to use the services, including reading/writing the stored ML models and associated supplemental information. SMO/Non-RT RIC may decide to only partially expose supplemental information of stored models to a service consumer.

Examples of service consumer include: the ML training host in the SMO/Non-RT RIC, ML training host in the Near-RT RIC, the external AI/ML server (over external AI/ML interface), etc.

Note: The interface to be used for ML model transfer between the ML model repository functionality and the Near-RT RIC is FFS.

A ML model can be continuously and incrementally updated/re-trained with online information, however, there is no guarantee that the updated model always leads to better RAN performance. Therefore, a previously stored ML model, which was proved to be well-performing, can serve as a backup if the running model evolves in the wrong direction.

The ML model repository is an "implementation variable" functionality. In the functional view of the Non-RT RIC architecture, it can be realized as an ML model repository function as part of the AI/ML Workflow functions. Note: ML model repository does not train ML models, it stores trained models.

3.3.3 AI/ML Monitoring

AI/ML Monitoring is an essential Non-RT RIC Framework functionality to enable run-time monitoring of AI/ML models which are deployed in Non-RT RIC. These AI/ML models can be provided by multi-vendors. The AI/ML Monitoring as key Non-RT RIC Framework functionality ensures these deployed AI/ML models performing normally as they declared in their agreements or contracts signed between model providers and platform operators.

Contract based strategy can be applied between model providers with application or service providers. Model providers need sign contracts with the service provider when they onboard their models to the platform provided by the service provider. The contract schema can include following types of information:

- Model Specification.
- Data Specification.
- Monitoring metrics.
- Policies.
- Feedback mechanism.

AI/ML monitoring is based on the information defined contract schema to support different models from different vendors.

The monitoring metrics should be defined based on different types of algorithms. Table 3.3-1 shows some monitoring metrics examples based on different types of algorithms.

Table 3.3-1: AI/ML model monitoring metrics examples

Binary classification problems	*Multiclass classification problems*	*Regression classification problems* *Forecasting problems*
Area under ROC	Accuracy	R squared
True positive rate (TPR)	Weighted True Positive Rate (wTPR)	Proportion explained variance
Precision	Weighted False Positive Rate (wFPR)	Root of mean squared error(RMSE)
F1-Measure	Weighted recall	Mean absolute error (MAE)
Logarithmic loss	Weighted precision	Mean squared error(MSE)
False positive rate (FPR)	Weighted F1-Measure	
Area under PR	Logarithmic loss	
Recall		

Contracts are signed between model providers and service providers, which defines the key information of the model, data, monitoring metrics, etc. A contract example is defined in JSON format below.

```
{
  "contract": [
    {
      "model_spec": {
        "algorithm_type": "binary_classification",
        "training_data_label": "risk"
      },
      "data_spec": {
        "data_type": "numeric"
      },
      "monitoring_metrics": {
        "Area_under_ROC": true,
        "Precision": true,
        "Logarithmic_loss": true,
        "Recall": true
      },
      "policy": {
        "policy_id": "hge7kahf84asdf44",
        "condition": "fault",
        "allow_action": [
          "terminate",
          "deploy"
        ]
      },
      "feedback_mech": {
        "feedback_data_location": "object_storage_spec",
        "feedback_interval": "12h"
      },
      "data_schema": {
        "fields": [
          {
            "metadata": {
            },
            "name": "bandwidth",
            "nullable": false,
            "type": "string"
          },
          {
            "metadata": {
            },
            "name": "throughput",
            "nullable": false,
            "type": "floating"
          },
          {
            "metadata": {
            },
            "name": "delay",
            "nullable": false,
            "type": "floating"
          },
          {
            "metadata": {
              },
            "name": "jitter",
            "nullable": true,
            "type": "floating"
          }
        ]
      }
    }
   ]
  }
```

In the functional view of the Non-RT RIC architecture, the AI/ML Monitoring can be realized as an AI/ML Monitoring function within the Non-RT RIC Framework.

3.4 R1 Services

The R1 interface (depicted in Figure 3.4-1) is comprised of a collection of services, hereafter referred to as R1 services, that facilitate the interactions between rApps and the Non-RT RIC framework.

3.4.1 General Description

As described earlier, the Non-RT RIC is comprised of Non-RT RIC Framework and Non-RT RIC Application (rApp) functional layers which capabilities are offered for consumption and usage through the R1 services. Such services include, but are not limited to:

- Integration services, including service registration and discovery services, service authentication and authorization services and communication support services.
- Data management and exposure services.
- A1-related services.
- AI/ML workflow services.
- O1-related services.
- O2-related services.

NOTE: The services listed above may be used on other O-RAN-defined interfaces as well.

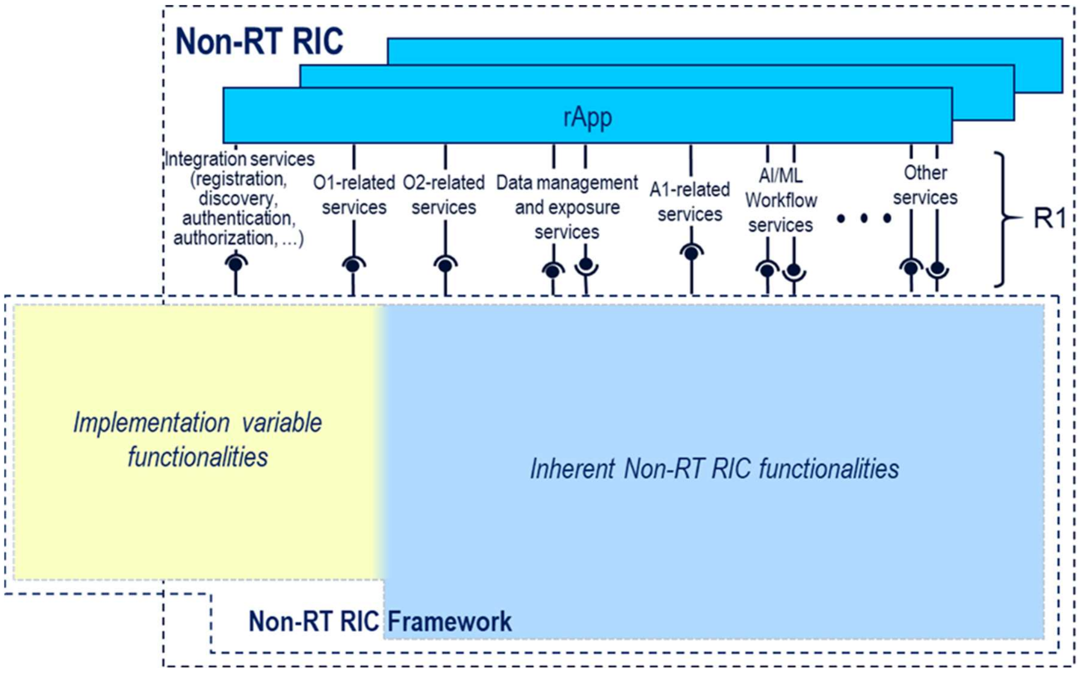

Non-RT RIC
rApp
Integration services (registration, discovery, authentication, authorization, ...)
O1-related services
O2-related services
Data management and exposure services
A1-related services
AI/ML Workflow services
Other services
R1
Implementation variable functionalities
Inherent Non-RT RIC functionalities
Non-RT RIC Framework

Figure 3.4-1: R1 interface as a collection of services between the Non-RT RIC Framework and rApps

3.4.2 General Principles

As described in section 5.2 of [11], at least some ML applications may profitably be deployed either to the Non-RT RIC, to the Near-RT RIC, or to both. This is certainly not true for all ML applications, as the same interfaces are not available in both platforms (e.g., E2, A1 downstream) nor do each have the same real-time requirements. However, for the types of interactions needed by the application types for which such portability would be beneficial, the design choices of the R1 interface should not unnecessarily become an obstacle for such portability.

- Leverage Existing Work: Because rApps will share some common functionality with xApps (startup, messaging, data sharing, etc.), the prior work done for xApps should be analyzed for applicability to the R1 use cases and leveraged where appropriate.
- Aligned integration services: Because integration or enablement services (such as registration, discovery, authorization, authentication, etc.) are needed to integrate rApps with the Non-RT RIC as well as to integrate xApps with the Near-RT RIC, the work done for xApp integration/enablement should be analyzed for applicability to rApp integration/enablement and leveraged where appropriate.
- Behavioral Alignment Where Practical: Because at least some ML applications will be deployable to the Non-RT RIC, Near-RT RIC, or both, the design choices made for R1 interactions should strive to minimize the refactoring needed of an rApp to enable deployment into a Near-RT RIC environment.
- Decoupled Interface: Although rApps will share some common functionality with xApps, they will also have many unique functionalities and needs that are not applicable to the Near-RT RIC. Therefore, the analysis and design choices made for the R1 interface should be done in parallel and not considered dependent on that of xApps.
- The R1 interface is the sole interface between an rApp and the functionality of the Non-RT RIC and SMO. Therefore, the R1 interface should be defined to meet all functional needs of rApps, with appropriate interface extensibility capabilities as needed.

The R1 interface will provide a level of abstraction between rApps that are consumers of data and their data sources:

- The R1 interface will provide functionality such that an rApp that is a consumer of data will not need to know the data needs or data capabilities of the producer of that data service.
- The R1 interface will provide functionality such that an rApp that is a producer of data will not need to know whether there exists one or multiple consumer rApps for that data, or the nature of that consumer (e.g., an rApp versus an entity external to the Non-RT RIC or SMO).
- The R1 interface will provide functionality such that an rApp that is a consumer of data will not need to know whether the data source is local (e.g., an rApp) or remote (e.g., an external data source such as O1).
- The R1 interface will provide functionality such that an rApp that is a consumer of data will not need to know whether the data was the product of a single entity (e.g., rApp), or the combined output of a complex chain of entities (e.g., a chain of rApps each consuming the value-added product of another).
- The Non-RT RIC Framework will not need to understand the semantic meaning of data or the internal structure thereof in order to provide its R1 data registration and data subscription services.

For example, the data registration and data subscription processing of the Non-RT RIC Framework should be the same irrespective of whether the Non-RT RIC Framework understands the semantic meaning and the internal structure of the data type (as perhaps it might for standardized data types, such as O1) or not (e.g., in the case of rApp-produced data that is not standardized).

3.4.3 Data Management and Exposure Services

3.4.3.1 R1 Security considerations

Void

3.4.3.2 Data Registration Services

This set of services is used in the setup of data routing paths between rApps and their data sources.

“Data Registration” functionality allows an rApp to communicate information about the data types that it consumes and produces to the Non-RT RIC Framework. Data registration interactions occur when an rApp is deployed on the Non-RT RIC Framework, and can be re-initiated by either entity thereafter as conditions change in the Non-RT RIC Framework.

Note that “data registration” interactions are different than “data subscription” interactions in that “data registration” involves only data types and their periodicity, whereas “data subscription” involves specific sets of data within a given “scope”.[1] Thus, the purpose of “data registration” interactions is to enable the Non-RT RIC Framework to facilitate data type level value-added services, such as:

- Authorizing an rApp to produce data of a certain data type. For example, a service provider may want to enforce a policy that two separate rApp instances should not produce the same data type within the same Non-RT RIC Framework instance.
- Authorizing an rApp to consume data of a certain data type. For example, a service provider may want to restrict the availability of certain data types, such as data types that relate to sensitive data.
- Accounting for the number of rApp consumers that exist for a given data type. For example, licensing of certain data-producing rApps may be sensitive to the number of consumer instances of that data type.

As part of registration processing, the Non-RT RIC Framework will match the needs of data type consumers against the capabilities of data type producers. A valid data type producer for matching purposes would include previously registered

[1] See “Data Subscription Services” section for more information.

rApp instances that have communicated during their own registration processing that they produce the data type desired for consumption, as well as other data sources, such as O1. For shorthand purposes, a valid data source that the Non-RT RIC Framework is aware of will be referred to in this document as a "known" data source.

Note: Whether deployment-time data registration interactions precede or follow rApp application configuration and/or activation is TBD.

As part of data registration interactions, the rApp will first communicate to the Non-RT RIC Framework the data types that it consumes, and then communicate the data types that it produces. This sequence is important to maintain, because the data types that an rApp is capable of producing in some cases will depend on the availability of data types that it desires to consume. The Non-RT RIC Framework will match rApp data type consumption needs against known data sources. In many cases the data type consumed will be standard RAN O1 data.

As part of its data registration functionality, the Non-RT RIC Framework might also perform some added value functionality such as:

- Keeping track of the number of rApp data consumers per data type and data source (e.g., for licensing or accounting purposes).
- Ensuring that there is no more than one producer, within a Non-RT RIC instance, of a given data type, or if there are multiple producers, verifying at data subscription-time that the two entities produce data corresponding to different "scopes" (e.g., geography or time).

The Non-RT RIC Framework will inform the rApp if no corresponding source can be matched to an rApp's "consumed data" requirements.

Only the rApp itself knows whether a data source is critical to the proper functioning of that application. Thus, it would be inappropriate for the Non-RT RIC Framework to fail registration processing due to a lack of data source. However, it is useful to have the Non-RT RIC Framework inform the rApp that a source cannot be found for one of its data type inputs, because the lack of a data input could, as noted above, affect the data type(s) the rApp is capable of producing. The rApp would determine how to react given that the data source is unknown: continue with the registration interactions or fail the registration.

Note that a source of missing data could be found later, such as when an rApp later initiates data registration interactions declaring its ability to produce that data type. To handle such a case, the design of the R1 interface can avoid requiring an rApp to re-declare its desire to consume that data type by having the Non-RT RIC Framework use an anonymous pub/sub paradigm to set up a routing rule for a data consumer rApp, even though no known source of this data currently exists. With such an approach, the Non-RT RIC Framework need only inform the previously registered rApp that the data type with a previously unknown data source now has a known data source. This interaction would inform the rApp that data subscription requests can now be initiated for that data type.

Of course, the presence of a "new" data source may enable the previously registered rApp to produce a data type that it previously could not due to lack of critical input data. Thus, it is necessary that an rApp be allowed to initiate R1 data registration interactions with the Non-RT RIC Framework even after initial deployment.

Just as rApp data registration can result in new data-consumption potential for rApps that have previously registered, so too can the deletion of an rApp result in the loss of data-consumption potential for previously registered rApps. This possibility requires that the Non-RT RIC Framework also be able to inform a previously registered rApp that a data type with a previously known data source now has an unknown data source. Because that data source may be a critical data source for the previously registered rApp in producing its own data type, it is necessary that an rApp be able to initiate R1 data registration interactions, even after initial deployment, to notify the Non-RT RIC Framework that it will no longer be producing a particular data type.

3.4.3.3 Data Subscription Services

This set of services is used in the request of specific data instances, and the fulfilment thereof by the Non-RT RIC Framework. The subscriber indicates the data type and periodicity thereof being subscribed to (which must be a previously registered “consumed” data type) and a “scope”. A “scope” declaration specifies the filter criteria applied to the “subject” of the data, such as a particular set of RAN network function instances and/or UEs being reported on.

The Non-RT RIC Framework determines whether the requested data is already locally available, or if (as in the case of RAN data) it needs to interact with an external data source (perhaps via the SMO) to obtain that data. The Non-RT RIC Framework relieves the data source of the burden of having to keep track of the number of data subscribers and mediating multiple subscribers to the same data stream.

3.4.4 Integration Services

The Non-RT RIC Framework is responsible for producing a collection of Integration Services to assist the service producers and consumers, including, but not limited to:

- Authentication of service producers and consumers.
- Authorization of service producers and consumers.
- Means for service producers to register/deregister the produced services, and to update the information in case of changes of the service availability.
- Means to notify the changes of the service availability to subscribed consumers.
- Discovery of available services.
- Information about available communication mechanisms (aka transports).

The Non-RT RIC Framework also comprises a communication support for the service consumers and producers to interact with each other. Nonetheless, a service producer may incorporate its own communication mechanism, so that a producer and a consumer can interact directly via that custom transport mechanism.

3.5 rApp Supporting Functions

Void

3.6 Other “Implementation Variable” Functions

Void

4 Requirements

4.1 R1 Interface Requirements

4.1.1 Non-RT RIC Framework Requirements

The following requirements pertain to the Non-RT RIC Framework functionality in support of the R1 interface.

4.1.1.1 Non-RT RIC Framework Data Registration Services Requirements

This section captures requirements on data registration interactions between the rApp and the Non-RT RIC Framework.

[REQ-nRTRfW-R1r-10]	The Non-RT RIC Framework shall support the capability to engage in 'data registration" interactions, via the R1 interface, with an rApp instance that has been newly deployed to that Non-RT RIC Framework.

[REQ-nRTRfW-R1r-30]	The Non-RT RIC Framework shall have the capability to match rApp data consumption needs against known rApp data sources, setting up routing between a data source rApp and data type consumer rApp.

Ref: AIML requirement REQ-Non-RT RIC-FUN7.

[REQ-nRTRfW-R1r-40]	The Non-RT RIC Framework shall recognize certain standard data types as having O1 as their known data source, that data source being provided as functionality inherent to the SMO. The Non-RT RIC Framework shall interact as necessary with the SMO to determine whether to consider a particular O1 data type as being available in a specific SMO implementation. A data type that was previously communicated as 'produced' by an rApp in a prior data registration interaction shall be considered a 'known' data type to the Non-RT RIC Framework.

Note: For example, a particular SMO implementation may not yet support a particular O1 data type, or a particular Service Provider may have disabled a particular O1 data type. In order for an O1 data type to be considered "known" it must be available within the context of the given SMO implementation and Service Provider environment.

Note that there is no expectation that data produced by rApps is in any way standardized, nor that the Non-RT RIC Framework must understand the semantic meaning or the internal structure of the data produced by rApps in order for that data type to be considered "known".

[REQ-nRTRfW-R1r-60]	In response to receipt of a consuming rApp's R1 data type request, the Non-RT RIC Framework shall set up the consuming routing rule for that data type to the rApp even if no "known" data source can be found.

[REQ-nRTRfW-R1r-70]	The Non-RT RIC Framework shall be capable of communicating to the R1 consumer rApp whether that rApp's data consumption needs can or cannot be fulfilled by a 'known' data source.

Ref: AIML requirement REQ-Non-RT RIC-FUN7.

rApps may add to the set of data that they consume or produce, for example due to environmental changes (e.g., notification of new produced data type availability).

[REQ-nRTRfW-R1r-90]	The Non-RT RIC Framework shall allow an rApp to initiate registration interactions declaring new data types consumed or produced, even after initial deployment.

[REQ-nRTRfW-R1r-100]	When an rApp interacts with the Non-RT RIC Framework to communicate a "produced" data type, the Non-RT RIC Framework shall determine whether there is already a data type routing rule for a registered consumer of that data type. If so, the Non-RT RIC Framework shall send a notification

	to that consuming rApp informing it that a data type with a previously unknown data source now has a known data source.

The Non-RT RIC Framework may need to delete from the "known" data sources for a given data type, for example due to removal of a producing rApp.

[REQ-nRTRfW-R1r-110]	When the Non-RT RIC Framework detects that an rApp is being undeployed, it shall conclude that any data types being produced by that rApp must be rescinded. It shall determine whether the data source of that data type must now be considered "unknown".

When a produced data type's source is rescinded, it may have ripple effects on any consuming rApps. The Non-RT RIC Framework will communicate to the consuming rApps of a data type when its source is being rescinded.

[REQ-nRTRfW-R1r-120]	When the data source for a data type transitions from "known" to "unknown", the Non-RT RIC Framework shall determine whether there are any registered consumers of that data type. If so, the Non-RT RIC Framework shall send a notification to that consuming rApp informing it that a currently known data source is being removed.

4.1.2 rApp Requirements

4.1.2.1 rApp Data Registration Service Requirements

The following requirements pertain to the rApp functionality in support of the R1 interface.

[REQ-nRTRApp-R1r-20]	Deployed rApp instances shall be able to communicate via the R1 interface information relating the data type(s) and periodicity thereof that the rApp consumes and produces, in that sequence.

Ref: AIML requirement REQ-Non-RT RIC-FUN6.

[REQ-nRTRApp-R1r-30]	An rApp that receives, as part of its data registration interactions, an indication that no corresponding data source can be found for a "consumed" data type shall be responsible for determining whether it can continue functioning without that data type or not.

Ref: AIML requirement REQ-Non-RT RIC-FUN7.

[REQ-nRTRApp-R1r-40]	An rApp shall be able to interrupt data registration interactions, including after having received an indication that its data consumption needs cannot be fulfilled for some data type. An rApp that has done so shall not continue with data registration processing.

Note: If an rApp does not interrupt its data registration processing after having been informed that no corresponding data source can be found for a "consumed" data type, it is understood that the rApp has functionality to perform runtime processing, perhaps with reduced functionality, in the absence of that data source.

When a data type with a known data source is rescinded, it may have ripple effects on any consuming rApps. The same is true when a new data source becomes available for a data type with a previously unknown data source. It is the responsibility of each affected rApp to determine how the lack of a data source, or the addition of a data source, impacts that rApp’s processing.

[REQ-nRTRApp-R1r-50]	When an rApp is informed that the ability to fulfill that rApp’s data consumption needs has changed, it is the responsibility of that rApp to determine how to accommodate that change.

Note: For example, any need for graceful termination of current execution threads will be the responsibility of the rApp.

Annex A (Informative): Non-RT RIC Processing Examples

Because examples can be useful in promoting understanding, this Annex provides examples to illustrate the concepts described in the main document.

A.1 Introduction to Illustrative Example

We will make use of the following fictitious rApps in some of the subsequent sections of this Annex. In this illustration, there are three separate rApps that execute in a given Non-RT RIC Framework instance.

- rApp "R" – An RF signal predictor. This rApp consumes RAN (O1) measurements of the actual RF signal experienced by a UE for its serving and neighbor cells ($RFa_{n,s}$) as well as other measurements useful in determining that UE's actual location (La). This rApp's role is to output a future time (e.g., t = 10 seconds) prediction of the location of that UE (Lp^t) as well as a prediction of the RF signal that will at that location be experienced by that UE for its serving and neighbor cells ($RFp^t_{n,s}$).
- rApp "U" – A cell utilization predictor. This rApp consumes RAN (O1) cell utilization measurements regarding the actual capacity utilization for a given cell site (CUa) over time. This rApp's role is to trend that input so as to allow it to output a future time (e.g., t = 60 minutes) prediction of that cell site's utilization (CUp^t).
- rApp "Q" – A UE QoE predictor. This rApp consumes measurements regarding a UE's RF signal (either an actual RAN measurement or a future time prediction) as well as a measurement of a cell site's capacity utilization (either an actual RAN measurement or a future time prediction). This rApp's role is to calculate the QoE experienced by that UE. Thus, this rApp can:
    - Estimate the actual QoE currently being experienced by a UE (Qa_s), by consuming measurements of the actual RF signal currently experienced by that UE relative to its serving cell (RFa_s) along with the actual cell utilization of that serving cell (CUa).
    - Estimate the QoE that a UE would experience were it currently connected instead to a neighbor cell (Qp_n), by consuming measurements of the actual RF signal currently experienced by that UE relative to that neighbor cell (RFa_n) along with the actual cell utilization of that neighbor cell (CUa).
    - Estimate the QoE that will be experienced by a UE at a future time (t) assuming that UE will remain connected to its serving cell (Qp^t_s), by consuming future time (t) predicted measurements of the RF signal that will be experienced by that UE relative to its serving cell (RFp^t_s), along with the future time (t) predicted cell utilization of that serving cell (CUp^t).
    - Estimate the QoE that would be experienced by a UE at that future time if that UE were to be connected instead to a neighbor cell (Qp^t_n), by consuming future time (t) predicted measurements of the RF signal that would be experienced by that UE relative to that neighbor cell (RFp^t_n), along with the future time (t) predicted cell utilization of that neighbor cell (CUp^t).

These data source relationships can be seen in Figure A.1-1 below. For the purposes of this section we need not specify what will be done with QoE measurements output by rApp Q, rather leaving it unstated for other sections of this document to build on this example. For now we merely point out that multiple options exist, both local (e.g., another rApp consuming the QoE measurement to determine an action to take across A1 or O1) and remote (e.g., the Non-RT RIC Framework forwarding the QoE measurement to a Near-RT RIC via A1-EI for use by some xApp).

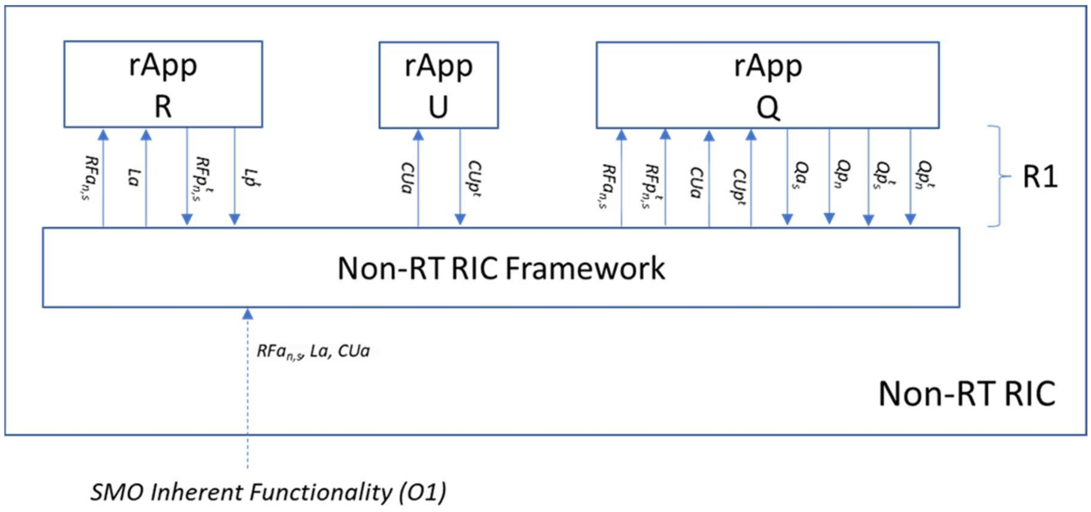


Figure A.1-1: Example rApp Data Sources

We will use sequence diagrams to illustrate the interactions using this example. In so doing, we will use the following actors:

- Non-RT RIC Inherent Functions: This actor will be chosen for the termination of an interaction when the associated functionality is inherent to the Non-RT RIC Framework.

- SMO Inherent Functions: This actor will be chosen for the termination of an interaction when the associated functionality is decidedly not part of the Non-RT RIC functional space, i.e., neither part of the Non-RT RIC Framework nor part of an rApp.

- Implementation Variable Functions: This actor will be chosen for the termination of an interaction when the associated functionality can be optionally considered part of the Non-RT RIC Framework, or not, in any given implementation.

A.2 R1 Processing Examples

A.2.1 R1 Data Registration Services Examples

R1 “data registration” refers to the set of functionality through which the Non-RT RIC Framework provides a set of data mediation services to the rApps, and the rApps provide a set of data production services to the Non-RT RIC Framework. The rApp will first communicate to the Non-RT RIC Framework the data that it consumes, and then communicate the data that it produces.

```
@startuml
Autonumber
skinparam defaultFontSize 30
Autonumber

Participant SMO as “SMO Inherent\nFunctions”
Participant ImpVar as “Implementation\nVariable\nFunctions”
Participant NRTR as “Non-RT RIC\nInherent\nFunctions”

Group Data Registration

  Group Interactions: rApp as Data Consumer

    Note over SMO, NRTR
```

```
      The rApp communicates over the R1 interface the data types that it consumes.
    End Note

  End

  Group Interactions: rApp as Data Producer

    Note over SMO, NRTR
      The rApp communicates over the R1 interface the data types that it produces.
    End Note

  End

End

@enduml
```

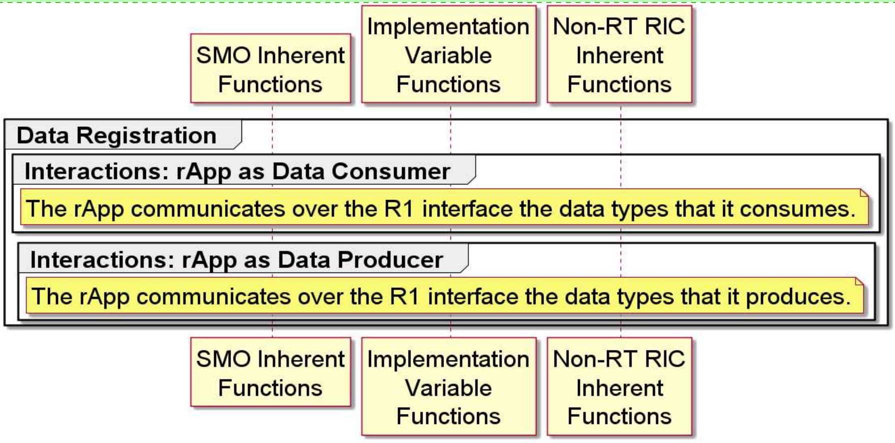

SMO Inherent Functions
Implementation Variable Functions
Non-RT RIC Inherent Functions
Data Registration
Interactions: rApp as Data Consumer
The rApp communicates over the R1 interface the data types that it consumes.
Interactions: rApp as Data Producer
The rApp communicates over the R1 interface the data types that it produces.
SMO Inherent Functions
Implementation Variable Functions
Non-RT RIC Inherent Functions

Figure A.2-1: Data Registration Interaction Summary

As described earlier, the Non-RT RIC Framework will match rApp data consumption needs against known data sources. In many cases the data consumed will be standard RAN O1 data.

To illustrate this, assume that rApp “R” is deployed first into the Non-RT RIC execution environment. This rApp would initiate R1 data registration interactions to declare that it consumes as input the RAN data types $RFa_{n,s}$ and La. Because these are O1 data types, the Non-RT RIC Framework would know that it must look to the SMO (inherent functionality) as their data source. Thus, the Non-RT RIC Framework will reply to rApp “R” that the source for these data types is known.

At some point after the data registration interactions, rApp “R” would initiate a formal “subscription” request with specific “scope” information (including the specific UEs or a class thereof from which to collect that input). Only then would the Non-RT RIC Framework (working with the SMO as necessary) start providing actual data measure instances. This “subscription” interaction will be discussed in a subsequent section.

As part of its continued data registration processing, rApp “R” would declare that it produces measure Lp^t, a future time prediction of the location of a UE, as well as produce data type $RFp^t_{n,s}$ a future time prediction of RF signal at a UE’s serving and neighbor cells.[2]

[2] Alternately the data type could be defined such that the neighbor and serving cell discriminators are understood as “scope” information and left out of the registration interaction. With this approach, the data type would be represented as RFp^t, and a “scope” parameter at data subscription time would indicate the cell type, “serving” or “neighbor”, of interest.

The above can be seen represented in the sequence diagrams below.

```
@startuml
skinparam defaultFontSize 30
Autonumber

Participant NRTR as “Non-RT RIC\nInherent\nFunctions”
Participant rAppR as “rApp R”

Group Data Consumer Interactions

Group In Parallel: Consumed Data Type 1

rAppR -> NRTR: Register Data Type RFa(n,s) as Consumed.
NRTR -> NRTR: Determine SMO as data source for\ndata type RFa(n,s).
NRTR -> NRTR: Set up anonymous routing rule for SMO\nas producer for data type RFa(n,s).
NRTR -> NRTR: Set up anonymous routing rule to rApp R\nas consumer for data type
RFa(n,s).

End

Group In Parallel: Consumed Data Type 2

rAppR -> NRTR: Register Data Type La as Consumed.
NRTR -> NRTR: Determine SMO as data source for\ndata type La.
NRTR -> NRTR: Set up anonymous routing rule for SMO\nas producer for data type La.
NRTR -> NRTR: Set up anonymous routing rule to rApp R\nas consumer for data type La.

End

End

Note over NRTR, rAppR
  Note that all “consumed” data type interactions must complete prior to initiating
  “produced” data type interactions.  This is because an rApp only knows what it
  can produce once it knows what data is available to it.
End Note

Group Data Producer Interactions

Group In Parallel: Produced Data Type 1

rAppR -> NRTR: Register Data Type Lp(t) as Produced.
NRTR -> NRTR: Set up anonymous routing rule from rApp R\nas producer for data type Lp(t).

End

Group In Parallel: Produced Data Type 2

rAppR -> NRTR: Register Data Type RFp(t)(n,s) as Produced.
NRTR -> NRTR: Set up anonymous routing rule from rApp R\nas producer for data type
RFp(t)(n,s).

End

Note over NRTR, rAppR
  Note that at this time there are producer routing rules for
  both La(t) and RFp(t)(n,s) with no corresponding consumer.
End Note

End

@enduml
```

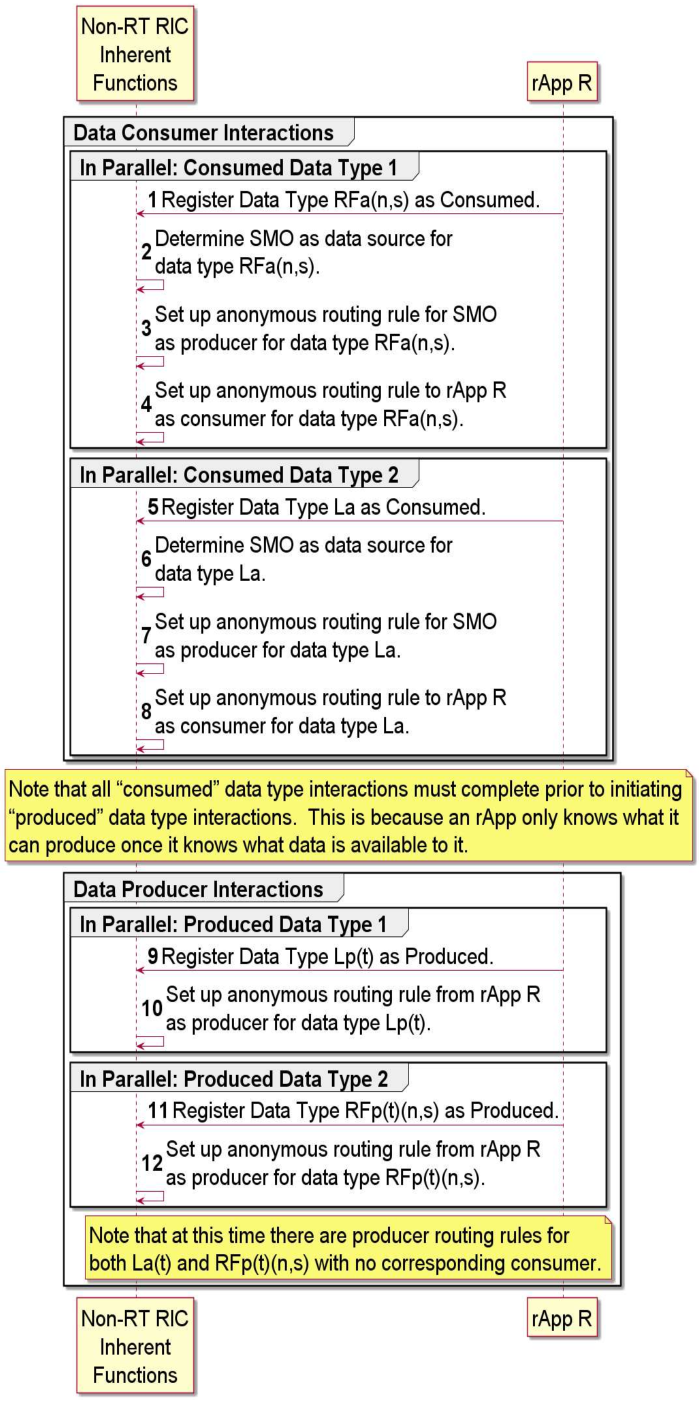

Non-RT RIC Inherent Functions
rApp R
Data Consumer Interactions
In Parallel: Consumed Data Type 1
1 Register Data Type RFa(n,s) as Consumed.
2 Determine SMO as data source for data type RFa(n,s).
3 Set up anonymous routing rule for SMO as producer for data type RFa(n,s).
4 Set up anonymous routing rule to rApp R as consumer for data type RFa(n,s).
In Parallel: Consumed Data Type 2
5 Register Data Type La as Consumed.
6 Determine SMO as data source for data type La.
7 Set up anonymous routing rule for SMO as producer for data type La.
8 Set up anonymous routing rule to rApp R as consumer for data type La.
Note that all "consumed" data type interactions must complete prior to initiating "produced" data type interactions. This is because an rApp only knows what it can produce once it knows what data is available to it.
Data Producer Interactions
In Parallel: Produced Data Type 1
9 Register Data Type Lp(t) as Produced.
10 Set up anonymous routing rule from rApp R as producer for data type Lp(t).
In Parallel: Produced Data Type 2
11 Register Data Type RFp(t)(n,s) as Produced.
12 Set up anonymous routing rule from rApp R as producer for data type RFp(t)(n,s).
Note that at this time there are producer routing rules for both La(t) and RFp(t)(n,s) with no corresponding consumer.
Non-RT RIC Inherent Functions
rApp R

Figure A.2-2: First Illustrative Example - rApp R Data Registration Interactions

The Non-RT RIC Framework would return a registration-time warning to the rApp if no corresponding source can be matched to an rApp's "consumed data" requirements. The rApp would determine how to react given that the data source is unknown: continue with the registration interactions or fail the registration.

The Non-RT RIC Framework might also perform some registration-time added value validations such as:

- Keeping track of the number of rApp data consumers per data type and data source (e.g., for licensing or accounting purposes).
- Ensuring that no other entity also produces data type $RFp^t_{n,s}$, or if they do verifying at data subscription-time that the two entities produce data corresponding to different "scopes" (e.g., geography or time).

We will illustrate the above considerations by extending our example, assuming that rApp "Q" is next deployed to the execution environment. This rApp would initiate R1 data registration interactions to declare that it consumes as input data sources $RFa_{n,s}$, $RFp^t_{n,s}$ (alternately RFp^t, see footnote above), CUa and CUp^t. Because the data source of $RFa_{n,s}$ and CUa is assumed to be standard O1, these data types would be treated as described above. The Non-RT RIC Framework would also respond that there is a known data source (i.e., rApp "R") for data type $RFp^t_{n,s}$. However, because rApp "U" had not yet been deployed and registered, the Non-RT RIC Framework would find no source for data type CUp^t.

Note that, as described above, rApp "Q" can produce meaningful output in the absence of CUp^t input: $Qa_{n,}$ and Qa_s, though it cannot produce Qp^t_n or Qp^t_s. Only the rApp itself knows whether a data source is critical to the proper functioning of that application. Thus, it would be inappropriate for the Non-RT RIC Framework to fail registration processing due to a lack of data source CUp^t. However, it is useful to have the Non-RT RIC Framework inform the rApp that a source cannot be found for one of its data type inputs, because the lack of a data input could, as in this case, affect the data type(s) the rApp is capable of producing. Because a source of missing data could be found later, the Non-RT RIC Framework would use an anonymous pub/sub paradigm to set up a routing rule for rApp "Q" as a consumer of data type CUp^t, even though no known source of this data currently exists.

These interactions can be seen in the sequence diagram below.

```
@startuml
skinparam defaultFontSize 30
Autonumber

Participant NRTR as "Non-RT RIC\nInherent\nFunctions"
Participant rAppQ as "rApp Q"

Group Data Consumer Interactions

Group In Parallel: Consumed Data Type 1

rAppQ -> NRTR: Register Data Type RFa(n,s) as Consumed.
NRTR -> NRTR: Determine SMO as data source for\ndata type RFa(n,s)
NRTR -> NRTR: Determine that there already exists a routing rule for\nSMO as producer for
data type RFa(n,s).
NRTR -> NRTR: Set up anonymous routing rule to rApp Q\nas consumer for data type
RFa(n,s).

End

Group In Parallel: Consumed Data Type 2

rAppQ -> NRTR: Register Data Type RFP(t)(n,s) as Consumed.
NRTR -> NRTR: Determine rApp R as data source for data type RFp(t)(n,s).
NRTR -> NRTR: Determine that there already exists a routing rule for\nrApp R as producer
for data type RFp(t)(n,s).
NRTR -> NRTR: Set up anonymous routing rule to rApp Q\nas consumer for data type
RFp(t)(n,s).

End

Group In Parallel: Consumed Data Type 3

rAppQ -> NRTR: Register Data Type CUa as Consumed.
NRTR -> NRTR: Determine SMO as data source for\ndata type CUa
NRTR -> NRTR: Set up anonymous routing rule from SMO\nas producer for data type CUa.
NRTR -> NRTR: Set up anonymous routing rule to rApp Q\nas consumer for data type CUa.
```

```
End

Group In Parallel: Consumed Data Type 4

rAppQ -> NRTR: Register Data Type CUp(t) as Consumed.
NRTR -> NRTR: Determine no known data source for data type CUp(t).
NRTR -> rAppQ: Notification that CUp(t) has no known data source.
rAppQ -> rAppQ: Decision to proceed with\nregistration processing.

Note over NRTR, rAppQ
Because rApp "Q" can perform predictions based on "actuals" and not just future
time predicted values, we will assume that lack of CUp(t) data is no reason to stop
registration processing.  Alternately rApp Q could throw an error.
The implication is that the Non-RT RIC Framework will reject any subsequent
Subscription requests for data type CUp(t).
End Note

NRTR -> NRTR: Set up anonymous routing rule to rApp Q\nas consumer for data type CUp(t).

End

Note over NRTR, rAppQ
  Note that at this time there is a consumer routing rule for CUp(t) with no
corresponding producer.
End Note

End

Group Data Producer Interactions

Group In Parallel: Produced Data Types

rAppQ -> NRTR: Register Data Type Qa(s) as Produced.
NRTR -> NRTR: Set up anonymous routing rule from rApp Q\nas producer for data type Qa(s).

rAppQ -> NRTR: Register Data Type Qp(n) as Produced.
NRTR -> NRTR: Set up anonymous routing rule from rApp Q\nas producer for data type Qp(n).

End

Note over NRTR, rAppQ
  Because there is no known data source for CUp(t), rApp Q cannot produce data
  types Qp(t)(s) or Qp(t)(n).  This exemplifies why it is necessary for data
  consumer interactions to complete prior to data producer interactions begin.
End Note

End

@enduml
```

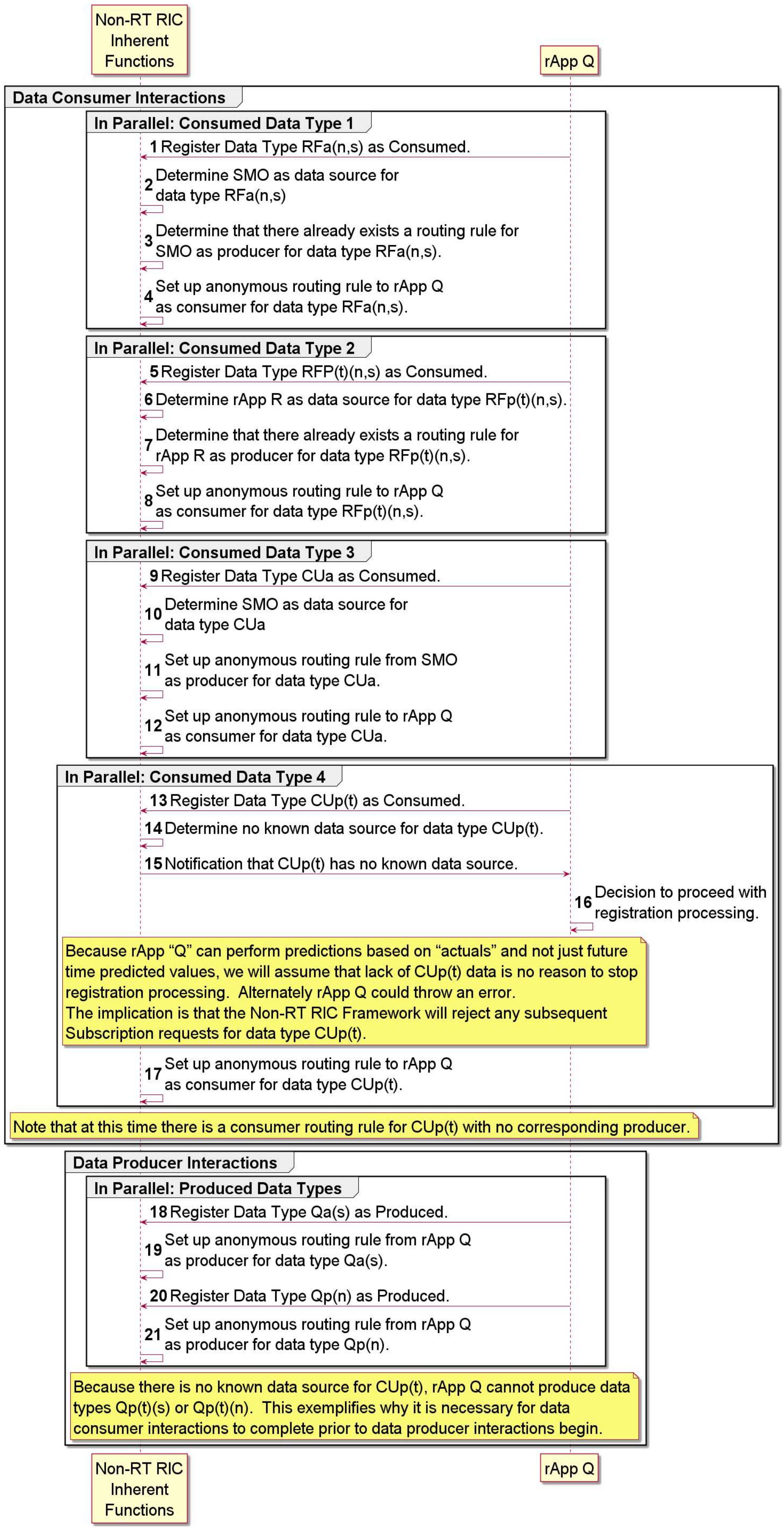

Non-RT RIC Inherent Functions
rApp Q
Data Consumer Interactions
In Parallel: Consumed Data Type 1
1 Register Data Type RFa(n,s) as Consumed.
2 Determine SMO as data source for data type RFa(n,s)
3 Determine that there already exists a routing rule for SMO as producer for data type RFa(n,s).
4 Set up anonymous routing rule to rApp Q as consumer for data type RFa(n,s).
In Parallel: Consumed Data Type 2
5 Register Data Type RFP(t)(n,s) as Consumed.
6 Determine rApp R as data source for data type RFp(t)(n,s).
7 Determine that there already exists a routing rule for rApp R as producer for data type RFp(t)(n,s).
8 Set up anonymous routing rule to rApp Q as consumer for data type RFp(t)(n,s).
In Parallel: Consumed Data Type 3
9 Register Data Type CUa as Consumed.
10 Determine SMO as data source for data type CUa
11 Set up anonymous routing rule from SMO as producer for data type CUa.
12 Set up anonymous routing rule to rApp Q as consumer for data type CUa.
In Parallel: Consumed Data Type 4
13 Register Data Type CUp(t) as Consumed.
14 Determine no known data source for data type CUp(t).
15 Notification that CUp(t) has no known data source.
16 Decision to proceed with registration processing.
Because rApp "Q" can perform predictions based on "actuals" and not just future time predicted values, we will assume that lack of CUp(t) data is no reason to stop registration processing. Alternately rApp Q could throw an error.
The implication is that the Non-RT RIC Framework will reject any subsequent Subscription requests for data type CUp(t).
17 Set up anonymous routing rule to rApp Q as consumer for data type CUp(t).
Note that at this time there is a consumer routing rule for CUp(t) with no corresponding producer.
Data Producer Interactions
In Parallel: Produced Data Types
18 Register Data Type Qa(s) as Produced.
19 Set up anonymous routing rule from rApp Q as producer for data type Qa(s).
20 Register Data Type Qp(n) as Produced.
21 Set up anonymous routing rule from rApp Q as producer for data type Qp(n).
Because there is no known data source for CUp(t), rApp Q cannot produce data types Qp(t)(s) or Qp(t)(n). This exemplifies why it is necessary for data consumer interactions to complete prior to data producer interactions begin.
Non-RT RIC Inherent Functions
rApp Q

Figure A.2-3: First Illustrative Example - rApp Q Data Registration Interactions

Continuing this example, if rApp "U" is later deployed to the execution environment of this Non-RT RIC Framework, it would initiate R1 data registration interactions to declare that it consumes as input data type *CUa*. Because a routing rule already exists for the SMO as the producer of *CUa*, all that remains is to set up a routing rule for rApp "U" as a consumer of this data type.

rApp "U" would also declare that it produces data type CUp^t. The Non-RT RIC Framework will set up the producing routing rule for rApp "U" as the producer of CUp^t data type. In addition, having previously set up the anonymous consuming routing rule of CUp^t data to rApp "Q", the Non-RT RIC Framework will notify rApp "Q" that a known data source now exists. It is up to rApp "Q" to decide how to proceed given this information.

The following sequence diagram illustrates these interactions. As it is beyond the scope of this section to go into the full set of R1 interactions involved in the deployment of an rApp, the following sequence diagram will only highlight that aspect of rApp U deployment processing that involves the communication of its produced data type CUp^t .

```
@startuml
skinparam defaultFontSize 30
Autonumber

Participant ImpVar as "Implementation\nVariable\nFunctions"
Participant NRTR as "Non-RT RIC\nInherent\nFunctions"
Participant rAppU as "rApp U"

Group rApp U Deployment

  Create rAppU
  ImpVar -> rAppU:

  Note over NRTR, rAppU
    rApp U is deployed to the Non-RT RIC Framework execution environment.
  End Note

… <i>Full deployment processing TBD</i> …

End

Group Data Consumer Interactions

rAppU -> NRTR: Register Data Type CUa as Consumed.
NRTR -> NRTR: Determine SMO as data source for\ndata type CUa.
NRTR -> NRTR: Determine that there already exists a routing rule for\nSMO as producer for
data type CUa.
NRTR -> NRTR: Set up anonymous routing rule to rApp U\nas consumer for data type CUa.

End

Group Data Producer Interactions

rAppU -> NRTR: Register Data Type CUp(t) as Produced.
NRTR -> NRTR: Set up anonymous routing rule from rApp U\nas producer for data type
CUp(t).
NRTR -> NRTR: Determine that there already exists a routing rule for\nrApp Q as consumer
of data type CUp(t).

End

Group Extended Data Consumer Interactions

NRTR -> rAppQ: Notification that CUp(t) now has a known data source.

Note over NRTR, rAppQ
```

```
rApp Q now knows that the Non-RT RIC Framework will no longer reject subsequent
Subscription
requests for data type CUp(t).  It must decide how to proceed.
rApp Q determines that it now has all the data dependencies needed to produce
additional data types: Qp(t)(s) and Qp(t)(n).
End Note

rAppQ -> rAppQ: Decision how to proceed

End

Group Extended Data Producer Interactions: In Parallel

rAppQ -> NRTR: Register Data Type Qp(t)(s) as Produced.
NRTR -> NRTR: Set up anonymous routing rule from rApp Q\nas producer for data type
Qp(t)(s).
rAppQ -> NRTR: Register Data Type Qp(t)(n) as Produced.
NRTR -> NRTR: Set up anonymous routing rule from rApp Q\nas producer for data type
Qp(t)(n).

NRTR -> NRTR: Determine if any consumer routing rules for\ndata types Qp(t)(s) or
Qp(t)(n).

Note over NRTR, rAppQ
If there were such consumer routing rule(s),the Non-RT RIC Framework
would send a notification to each corresponding consuming rApp notifying
that Qp(t)(s) and Qp(t)(n) now have known data sources.  This interaction
would be equivalent to step 9 shown above.
End Note

End

@enduml
```

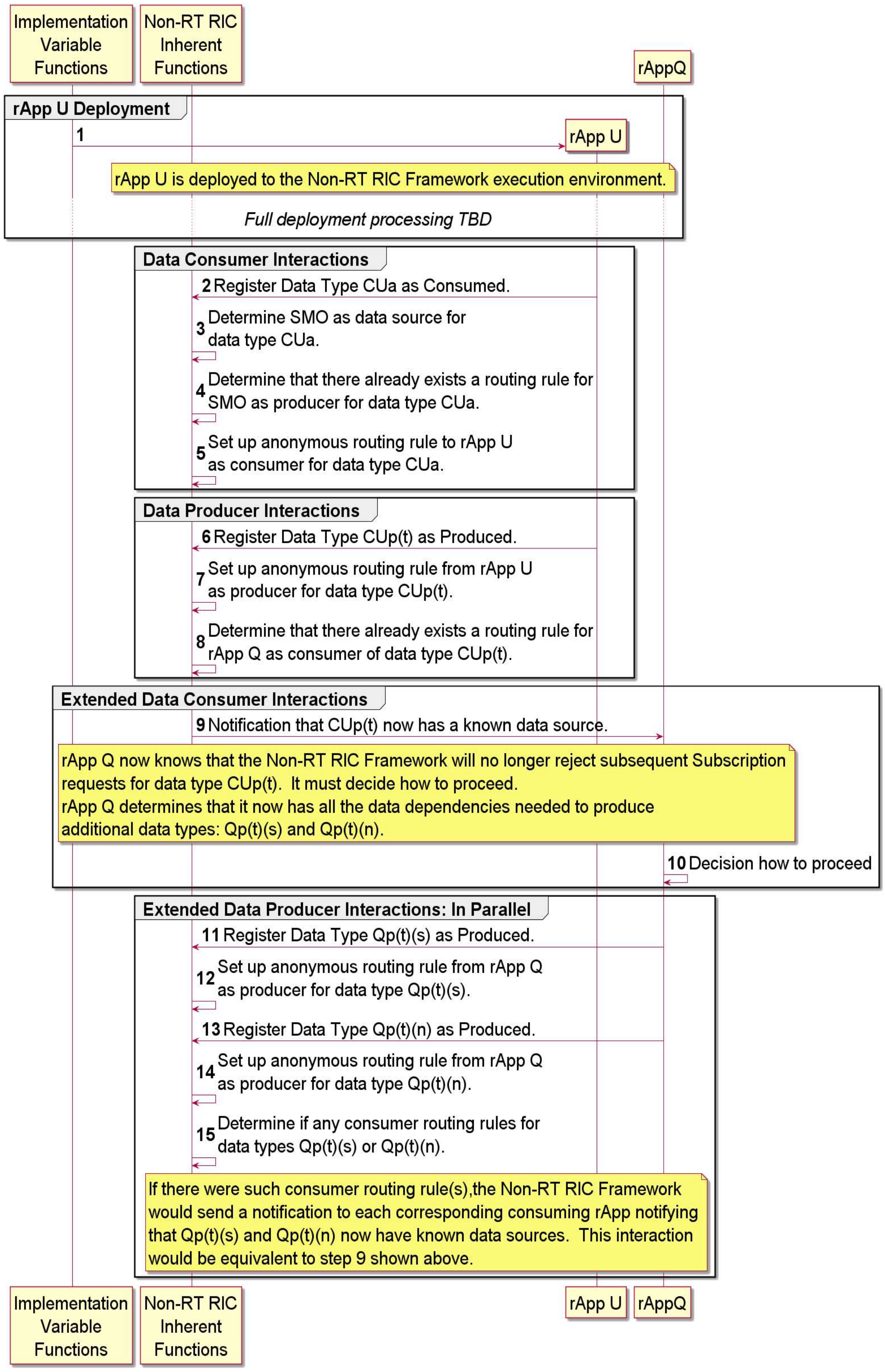

Implementation Variable Functions
Non-RT RIC Inherent Functions
rAppQ
rApp U Deployment
1
rApp U
rApp U is deployed to the Non-RT RIC Framework execution environment.
Full deployment processing TBD
Data Consumer Interactions
2 Register Data Type CUa as Consumed.
3 Determine SMO as data source for data type CUa.
4 Determine that there already exists a routing rule for SMO as producer for data type CUa.
5 Set up anonymous routing rule to rApp U as consumer for data type CUa.
Data Producer Interactions
6 Register Data Type CUp(t) as Produced.
7 Set up anonymous routing rule from rApp U as producer for data type CUp(t).
8 Determine that there already exists a routing rule for rApp Q as consumer of data type CUp(t).
Extended Data Consumer Interactions
9 Notification that CUp(t) now has a known data source.
rApp Q now knows that the Non-RT RIC Framework will no longer reject subsequent Subscription requests for data type CUp(t). It must decide how to proceed.
rApp Q determines that it now has all the data dependencies needed to produce additional data types: Qp(t)(s) and Qp(t)(n).
10 Decision how to proceed
Extended Data Producer Interactions: In Parallel
11 Register Data Type Qp(t)(s) as Produced.
12 Set up anonymous routing rule from rApp Q as producer for data type Qp(t)(s).
13 Register Data Type Qp(t)(n) as Produced.
14 Set up anonymous routing rule from rApp Q as producer for data type Qp(t)(n).
15 Determine if any consumer routing rules for data types Qp(t)(s) or Qp(t)(n).
If there were such consumer routing rule(s),the Non-RT RIC Framework would send a notification to each corresponding consuming rApp notifying that Qp(t)(s) and Qp(t)(n) now have known data sources. This interaction would be equivalent to step 9 shown above.
Implementation Variable Functions
Non-RT RIC Inherent Functions
rApp U
rAppQ

Figure A.2-4: First Illustrative Example - rApp U Registration Interactions

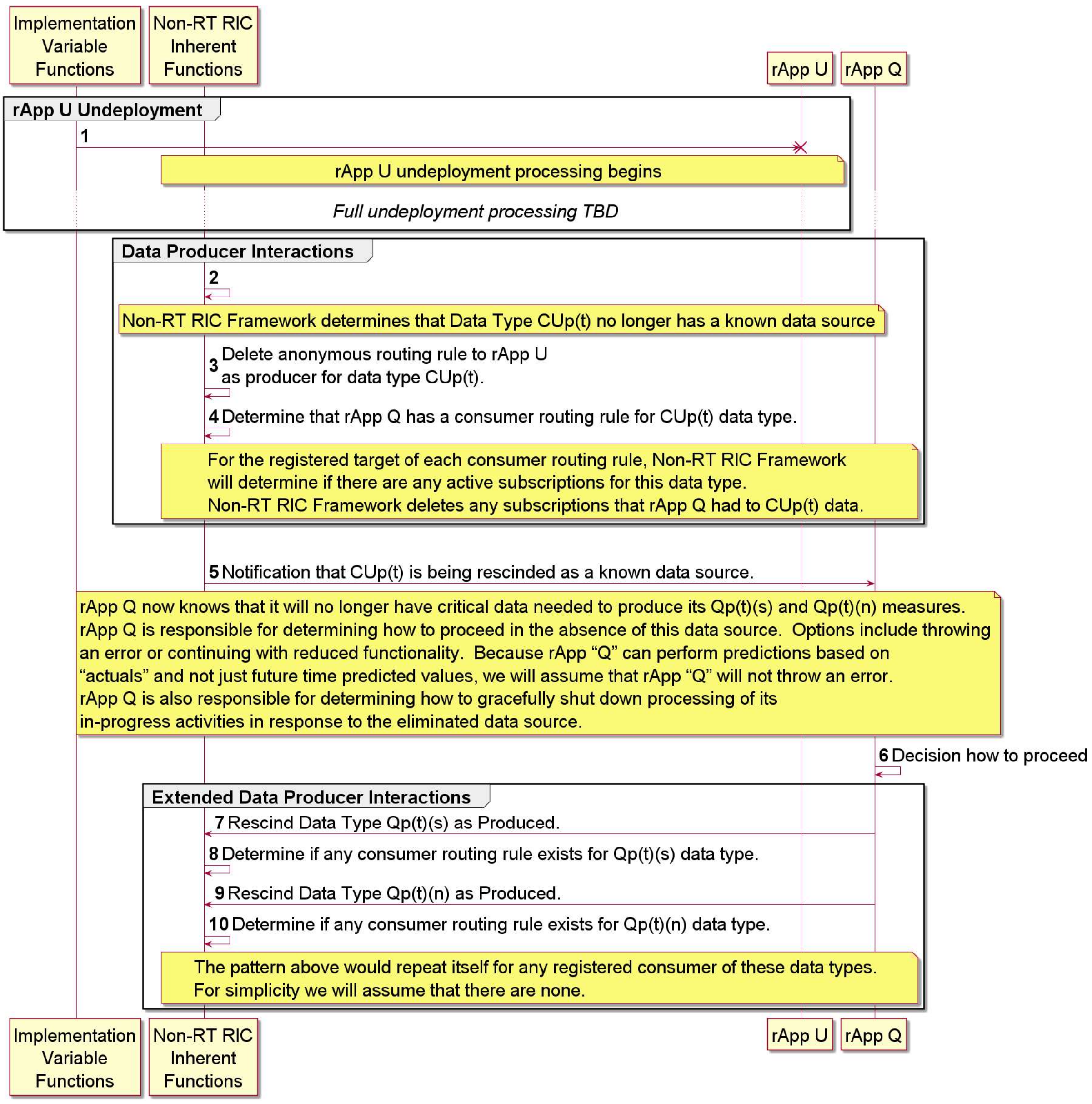
Just as the deployment of an rApp can result in new data-producing potential for previously deployed rApps, the deletion of an rApp can result in the loss of data-producing potential for previously deployed rApps.

As an example, if rApp “U” were next to be undeployed from the Non-RT RIC Framework, then rApp “Q” would lose its data source for data type CUp^t. This would result in rApp “Q” no longer being able to produce data types Qp^t_s and Qp^t_n.

As it is beyond the scope of this section to go into the full set of R1 interactions involved in the undeployment of an rApp, the following sequence diagram will only highlight that aspect of rApp “U” undeployment processing that involves the rescinding of its produced data type CUp^t.

```
@startuml
skinparam defaultFontSize 30
Autonumber

Participant ImpVar as "Implementation\nVariable\nFunctions"
Participant NRTR as "Non-RT RIC\nInherent\nFunctions"
Participant rAppU as "rApp U"
Participant rAppQ as "rApp Q"

Group rApp U Undeployment

  ImpVar -> rAppU !! :

  Note over NRTR, rAppU
    rApp U undeployment processing begins
  End Note

… <i>Full undeployment processing TBD</i> …

End

Group Data Producer Interactions

NRTR -> NRTR:

Note over NRTR, rAppU
  Non-RT RIC Framework determines that Data Type CUp(t) no longer has a known data source
End Note

NRTR -> NRTR: Delete anonymous routing rule to rApp U\nas producer for data type CUp(t).

NRTR -> NRTR: Determine that rApp Q has a consumer routing rule for CUp(t) data type.

Note over NRTR, rAppQ
For the registered target of each consumer routing rule, Non-RT RIC Framework
will determine if there are any active subscriptions for this data type.
Non-RT RIC Framework deletes any subscriptions that rApp Q had to CUp(t) data.
End Note

End

Group Extended Data Consumer Interactions

NRTR -> rAppQ: Notification that CUp(t) is being rescinded as a known data source.

Note over NRTR, rAppQ
rApp Q now knows that it will no longer have critical data needed to produce its Qp(t)(s)
and Qp(t)(n) measures.
rApp Q is responsible for determining how to proceed in the absence of this data source.
Options include throwing
an error or continuing with reduced functionality.  Because rApp "Q" can perform
predictions based on
```

```
"actuals" and not just future time predicted values, we will assume that rApp "Q" will
not throw an error.
rApp Q is also responsible for determining how to gracefully shut down processing of its
in-progress activities in response to the eliminated data source.
End Note

rAppQ -> rAppQ: Decision how to proceed

Group Extended Data Producer Interactions

rAppQ -> NRTR: Rescind Data Type Qp(t)(s) as Produced.
NRTR -> NRTR: Determine if any consumer routing rule exists for Qp(t)(s) data type.

rAppQ -> NRTR: Rescind Data Type Qp(t)(n) as Produced.
NRTR -> NRTR: Determine if any consumer routing rule exists for Qp(t)(n) data type.

Note over NRTR, rAppQ
The pattern above would repeat itself for any registered consumer of these data types.
For simplicity we will assume that there are none.
End Note

End

@enduml
```



Implementation Variable Functions
Non-RT RIC Inherent Functions
rApp U
rApp Q
rApp U Undeployment
1
rApp U undeployment processing begins
Full undeployment processing TBD
Data Producer Interactions
2
Non-RT RIC Framework determines that Data Type CUp(t) no longer has a known data source
3 Delete anonymous routing rule to rApp U as producer for data type CUp(t).
4 Determine that rApp Q has a consumer routing rule for CUp(t) data type.
For the registered target of each consumer routing rule, Non-RT RIC Framework will determine if there are any active subscriptions for this data type. Non-RT RIC Framework deletes any subscriptions that rApp Q had to CUp(t) data.
5 Notification that CUp(t) is being rescinded as a known data source.
rApp Q now knows that it will no longer have critical data needed to produce its Qp(t)(s) and Qp(t)(n) measures. rApp Q is responsible for determining how to proceed in the absence of this data source. Options include throwing an error or continuing with reduced functionality. Because rApp “Q” can perform predictions based on “actuals” and not just future time predicted values, we will assume that rApp “Q” will not throw an error. rApp Q is also responsible for determining how to gracefully shut down processing of its in-progress activities in response to the eliminated data source.
6 Decision how to proceed
Extended Data Producer Interactions
7 Rescind Data Type Qp(t)(s) as Produced.
8 Determine if any consumer routing rule exists for Qp(t)(s) data type.
9 Rescind Data Type Qp(t)(n) as Produced.
10 Determine if any consumer routing rule exists for Qp(t)(n) data type.
The pattern above would repeat itself for any registered consumer of these data types. For simplicity we will assume that there are none.
Implementation Variable Functions
Non-RT RIC Inherent Functions
rApp U
rApp Q

Figure A.2-5: Second Illustrative Example - rApp Q Rescinds Produced Data

Annex ZZZ: O-RAN Adopter License Agreement

BY DOWNLOADING, USING OR OTHERWISE ACCESSING ANY O-RAN SPECIFICATION, ADOPTER AGREES TO THE TERMS OF THIS AGREEMENT.

This O-RAN Adopter License Agreement (the “Agreement”) is made by and between the O-RAN Alliance and the entity that downloads, uses or otherwise accesses any O-RAN Specification, including its Affiliates (the “Adopter”).

This is a license agreement for entities who wish to adopt any O-RAN Specification.

Section 1: DEFINITIONS

1.1 “Affiliate” means an entity that directly or indirectly controls, is controlled by, or is under common control with another entity, so long as such control exists. For the purpose of this Section, “Control” means beneficial ownership of fifty (50%) percent or more of the voting stock or equity in an entity.

1.2 “Compliant Implementation” means any system, device, method or operation (whether implemented in hardware, software or combinations thereof) that fully conforms to a Final Specification.

1.3 “Adopter(s)” means all entities, who are not Members, Contributors or Academic Contributors, including their Affiliates, who wish to download, use or otherwise access O-RAN Specifications.

1.4 “Minor Update” means an update or revision to an O-RAN Specification published by O-RAN Alliance that does not add any significant new features or functionality and remains interoperable with the prior version of an O-RAN Specification. The term “O-RAN Specifications” includes Minor Updates.

1.5 “Necessary Claims” means those claims of all present and future patents and patent applications, other than design patents and design registrations, throughout the world, which (i) are owned or otherwise licensable by a Member, Contributor or Academic Contributor during the term of its Member, Contributor or Academic Contributorship; (ii) such Member, Contributor or Academic Contributor has the right to grant a license without the payment of consideration to a third party; and (iii) are necessarily infringed by a Compliant Implementation (without considering any Contributions not included in the Final Specification). A claim is necessarily infringed only when it is not possible on technical (but not commercial) grounds, taking into account normal technical practice and the state of the art generally available at the date any Final Specification was published by the O-RAN Alliance or the date the patent claim first came into existence, whichever last occurred, to make, sell, lease, otherwise dispose of, repair, use or operate a Compliant Implementation without infringing that claim. For the avoidance of doubt in exceptional cases where a Final Specification can only be implemented by technical solutions, all of which infringe patent claims, all such patent claims shall be considered Necessary Claims.

1.6 “Defensive Suspension” means for the purposes of any license grant pursuant to Section 3, Member, Contributor, Academic Contributor, Adopter, or any of their Affiliates, may have the discretion to include in their license a term allowing the licensor to suspend the license against a licensee who brings a patent infringement suit against the licensing Member, Contributor, Academic Contributor, Adopter, or any of their Affiliates.

Section 2: COPYRIGHT LICENSE

2.1 Subject to the terms and conditions of this Agreement, O-RAN Alliance hereby grants to Adopter a nonexclusive, nontransferable, irrevocable, non-sublicensable, worldwide copyright license to obtain, use and modify O-RAN Specifications, but not to further distribute such O-RAN Specification in any modified or unmodified way, solely in furtherance of implementations of an O-RAN

Specification.

2.2 Adopter shall not use O-RAN Specifications except as expressly set forth in this Agreement or in a separate written agreement with O-RAN Alliance.

Section 3: FRAND LICENSE

3.1 Members, Contributors and Academic Contributors and their Affiliates are prepared to grant based on a separate Patent License Agreement to each Adopter under Fair Reasonable And Non- Discriminatory (FRAND) terms and conditions with or without compensation (royalties) a nonexclusive, non-transferable, irrevocable (but subject to Defensive Suspension), non-sublicensable, worldwide patent license under their Necessary Claims to make, have made, use, import, offer to sell, lease, sell and otherwise distribute Compliant Implementations; provided, however, that such license shall not extend: (a) to any part or function of a product in which a Compliant Implementation is incorporated that is not itself part of the Compliant Implementation; or (b) to any Adopter if that Adopter is not making a reciprocal grant to Members, Contributors and Academic Contributors, as set forth in Section 3.3. For the avoidance of doubt, the foregoing licensing commitment includes the distribution by the Adopter's distributors and the use by the Adopter's customers of such licensed Compliant Implementations.

3.2 Notwithstanding the above, if any Member, Contributor or Academic Contributor, Adopter or their Affiliates has reserved the right to charge a FRAND royalty or other fee for its license of Necessary Claims to Adopter, then Adopter is entitled to charge a FRAND royalty or other fee to such Member, Contributor or Academic Contributor, Adopter and its Affiliates for its license of Necessary Claims to its licensees.

3.3 Adopter, on behalf of itself and its Affiliates, shall be prepared to grant based on a separate Patent License Agreement to each Members, Contributors, Academic Contributors, Adopters and their Affiliates under Fair Reasonable And Non-Discriminatory (FRAND) terms and conditions with or without compensation (royalties) a nonexclusive, non-transferable, irrevocable (but subject to Defensive Suspension), non-sublicensable, worldwide patent license under their Necessary Claims to make, have made, use, import, offer to sell, lease, sell and otherwise distribute Compliant Implementations; provided, however, that such license will not extend: (a) to any part or function of a product in which a Compliant Implementation is incorporated that is not itself part of the Compliant Implementation; or (b) to any Members, Contributors, Academic Contributors, Adopters and their Affiliates that is not making a reciprocal grant to Adopter, as set forth in Section 3.1. For the avoidance of doubt, the foregoing licensing commitment includes the distribution by the Members', Contributors', Academic Contributors', Adopters' and their Affiliates' distributors and the use by the Members', Contributors', Academic Contributors', Adopters' and their Affiliates' customers of such licensed Compliant Implementations.

Section 4: TERM AND TERMINATION

4.1 This Agreement shall remain in force, unless early terminated according to this Section 4.

4.2 O-RAN Alliance on behalf of its Members, Contributors and Academic Contributors may terminate this Agreement if Adopter materially breaches this Agreement and does not cure or is not capable of curing such breach within thirty (30) days after being given notice specifying the breach.

4.3 Sections 1, 3, 5 - 11 of this Agreement shall survive any termination of this Agreement. Under surviving Section 3, after termination of this Agreement, Adopter will continue to grant licenses (a) to entities who become Adopters after the date of termination; and (b) for future versions of O-RAN Specifications that are backwards compatible with the version that was current as of the date of termination.

Section 5: CONFIDENTIALITY

Adopter will use the same care and discretion to avoid disclosure, publication, and dissemination of O-RAN Specifications to third parties, as Adopter employs with its own confidential information, but no less than reasonable care. Any disclosure by Adopter to its Affiliates, contractors and consultants should be subject to an obligation of confidentiality at least as restrictive as those contained in this Section. The foregoing obligation shall not apply to any information which is: (1) rightfully known by Adopter without any limitation on use or disclosure prior to disclosure; (2) publicly available through no fault of Adopter; (3) rightfully received without a duty of confidentiality; (4) disclosed by O-RAN Alliance or a Member, Contributor or Academic Contributor to a third party without a duty of confidentiality on such third party; (5) independently developed by Adopter; (6) disclosed pursuant to the order of a court or other authorized governmental body, or as required by law, provided that Adopter provides reasonable prior written notice to O-RAN Alliance, and cooperates with O-RAN Alliance and/or the applicable Member, Contributor or Academic Contributor to have the opportunity to oppose any such order; or (7) disclosed by Adopter with O-RAN Alliance's prior written approval.

Section 6: INDEMNIFICATION

Adopter shall indemnify, defend, and hold harmless the O-RAN Alliance, its Members, Contributors or Academic Contributors, and their employees, and agents and their respective successors, heirs and assigns (the "Indemnitees"), against any liability, damage, loss, or expense (including reasonable attorneys' fees and expenses) incurred by or imposed upon any of the Indemnitees in connection with any claims, suits, investigations, actions, demands or judgments arising out of Adopter's use of the licensed O-RAN Specifications or Adopter's commercialization of products that comply with O-RAN Specifications.

Section 7: LIMITATIONS ON LIABILITY; NO WARRANTY

EXCEPT FOR BREACH OF CONFIDENTIALITY, ADOPTER'S BREACH OF SECTION 3, AND ADOPTER'S INDEMNIFICATION OBLIGATIONS, IN NO EVENT SHALL ANY PARTY BE LIABLE TO ANY OTHER PARTY OR THIRD PARTY FOR ANY INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE OR CONSEQUENTIAL DAMAGES RESULTING FROM ITS PERFORMANCE OR NON-PERFORMANCE UNDER THIS AGREEMENT, IN EACH CASE WHETHER UNDER CONTRACT, TORT, WARRANTY, OR OTHERWISE, AND WHETHER OR NOT SUCH PARTY HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES. O-RAN SPECIFICATIONS ARE PROVIDED "AS IS" WITH NO WARRANTIES OR CONDITIONS WHATSOEVER, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE. THE O-RAN ALLIANCE AND THE MEMBERS, CONTRIBUTORS OR ACADEMIC CONTRIBUTORS EXPRESSLY DISCLAIM ANY WARRANTY OR CONDITION OF MERCHANTABILITY, SECURITY, SATISFACTORY QUALITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, ERROR-FREE OPERATION, OR ANY WARRANTY OR CONDITION FOR O-RAN SPECIFICATIONS.

Section 8: ASSIGNMENT

Adopter may not assign the Agreement or any of its rights or obligations under this Agreement or make any grants or other sublicenses to this Agreement, except as expressly authorized hereunder, without having first received the prior, written consent of the O-RAN Alliance, which consent may be withheld in O-RAN Alliance's sole discretion. O-RAN Alliance may freely assign this Agreement.

Section 9: THIRD-PARTY BENEFICIARY RIGHTS

Adopter acknowledges and agrees that Members, Contributors and Academic Contributors (including future Members, Contributors and Academic Contributors) are entitled to rights as a third-party beneficiary under this Agreement, including as licensees under Section 3.

Section 10: BINDING ON AFFILIATES

Execution of this Agreement by Adopter in its capacity as a legal entity or association constitutes that legal entity's or association's agreement that its Affiliates are likewise bound to the obligations that are applicable to Adopter hereunder and are also entitled to the benefits of the rights of Adopter hereunder.

Section 11: GENERAL

This Agreement is governed by the laws of Germany without regard to its conflict or choice of law provisions.

This Agreement constitutes the entire agreement between the parties as to its express subject matter and expressly supersedes and replaces any prior or contemporaneous agreements between the parties, whether written or oral, relating to the subject matter of this Agreement.

Adopter, on behalf of itself and its Affiliates, agrees to comply at all times with all applicable laws, rules and regulations with respect to its and its Affiliates' performance under this Agreement, including without limitation, export control and antitrust laws. Without limiting the generality of the foregoing, Adopter acknowledges that this Agreement prohibits any communication that would violate the antitrust laws.

By execution hereof, no form of any partnership, joint venture or other special relationship is created between Adopter, or O-RAN Alliance or its Members, Contributors or Academic Contributors. Except as expressly set forth in this Agreement, no party is authorized to make any commitment on behalf of Adopter, or O-RAN Alliance or its Members, Contributors or Academic Contributors.

In the event that any provision of this Agreement conflicts with governing law or if any provision is held to be null, void or otherwise ineffective or invalid by a court of competent jurisdiction, (i) such provisions will be deemed stricken from the contract, and (ii) the remaining terms, provisions, covenants and restrictions of this Agreement will remain in full force and effect.

Any failure by a party or third party beneficiary to insist upon or enforce performance by another party of any of the provisions of this Agreement or to exercise any rights or remedies under this Agreement or otherwise by law shall not be construed as a waiver or relinquishment to any extent of the other parties' or third party beneficiary's right to assert or rely upon any such provision, right or remedy in that or any other instance; rather the same shall be and remain in full force and effect.