Deliverable-5.2

Specification of the common elements of the management framework

Deliverable Editor: Sven van der Meer, LMI

Publication date:	30-December-2014
Deliverable Nature:	Report
Dissemination level (Confidentiality):	PU (Public)
Project acronym:	PRISTINE
Project full title:	PRogrammability In RINA for European Supremacy of virTualIsed NEtworks
Website:	www.ict-pristine.eu
Keywords:	DIF, management, system, RIB, elements, common
Synopsis:	This document outlines the implementation specification of the common elements of the management framework. It will capture a snapshot of the RIB, associated model objects, document the concepts and API to the management agent.

The research leading to these results has received funding from the European Community's Seventh Framework Programme for research, technological development and demonstration under Grant Agreement No. 619305.

List of Contributors

Deliverable Editor: Sven van der Meer, LMI
i2CAT: Eduard Grasa, Bernat Gaston
NXW: Francesco Salvestrini
TSSG: Micheal Crotty, Jason Barron
BISDN: Marc Sune, Victor Alvarez
TRT: Sarah Haines
ATOS: Miguel Angel
CN: Roberto Riggio

Disclaimer

Executive Summary

This deliverable is an update on the progress of the PRISTINE Distributed Management System (DMS). It is broken down into several sections which provide more detail on the strands that need to be woven together to make the Distributed Management System (DMS) a functioning reality. It begins with an overview of the DMS architecture, which outlines the major components, and how they work together to form the DMS.

This is followed by an update on the specification of the Resource Information Base (RIB). This section highlights the major changes since [D51]. This is followed by a more detailed description of the Managed Object (MO) templates. These templates are based on existing work in ITU.T [x722] however they have been simplified where possible [D51]. These "templates" are used to specify the Managed Objects (MO) that make up the RIB. The following section then gives an overview of the currently specified Managed Objects (MO).

The next sub-section then attempts to validate the defined RIB, by examining some typical Manager / Agent interactions. We introduce a validation methodology based on "typical" management activities that would be necessary in a deployed system. This is followed by the detailed description of the management actions between the Manager and Management Agent, or Management Agent and Manager, necessary to complete the management activity.

The next technical section outlines the design of the Management Agent in particular. This is followed by some of the agent workflows, and a description of the agent components. The last section describes the guiding principles that govern the design of the DMS RIB library. The RIB library provides the means to create, destroy and manipulate RIB objects. It uses the Common Distributed Application Protocol (CDAP) library, however it provides a higher level of abstraction for RIB users (developers).

The final section outline the future plans for progressing the RIB, the Management Agent and the Manager. Appendix A contains a formal type specification, for both the basic and complex data-types used in the MOs. Appendix B contains a snapshot of the formal specification of the MOs that form the RIB.

Table of Contents

1. Introduction

This deliverable describes the progress to develop a unified network management system for RINA. This unified management system will provide configuration, performance and security management functions for the PRISTINE project scenarios as described in [D21]. This is referred to as a Distributed Management System (DMS).

The Recursive Inter-Network Architecture (RINA) is an emerging clean-slate programmable networking approach, centring on the Inter-Process Communication (IPC) paradigm, which will support high scalability, multi-homing, built-in security, seamless access to real-time information and operation in dynamic environments. The heart of this networking structure is naturally formed and organised by blocks of containers called *"Distributed Information Facilities - DIFs"* where each block has programmable functions to be attributed to as required. A DIF is seen as an organising structure, grouping together application processes that provide IPC services and are configured under the same policies (see glossary term DIF).

1.1. Scope

The document begins with an overview of the DMS architecture, which outlines the major components, and how they work together to form the DMS. The DMS will include multi-function and multi-layer DIF management, within a single administrative domain. This administrative simplification allows some complications of inter-domain coordination to be ignored, for example, contradicting manager instructions from different peer administrative domains. The multi-function nature of the DMS is covered by Configuration, Performance and Security management functions.

This deliverable presents an update on the specification of the Resource Information Base (RIB). This includes a formal specification of the templates used to describe Managed Objects (MO). Typical Manager-Agent interactions are introduced as a means to verify the validity of the RIB objects. Appendix B contains the full specification of the Managed Objects that form the RIB.

The second focus is to outline the design of the Management Agent and the software components that make up the agent. As with the RIB section, agent work-flows are introduced to verify the design of the Management Agent. The final section outlines the future plans for progressing the RIB, the Management Agent and the Manager.

1.2. Terminology refinement

This section provides a number of clarifications on the **terminology** used within this document that was not described in deliverable D51.

In order to clearly define the difference between the various types of policies, a distinct name has been chosen based on where the policy can be applied. This distinction is necessary as some of these policies operate at a considerably higher level of abstraction. This is illustrated in the following diagram:

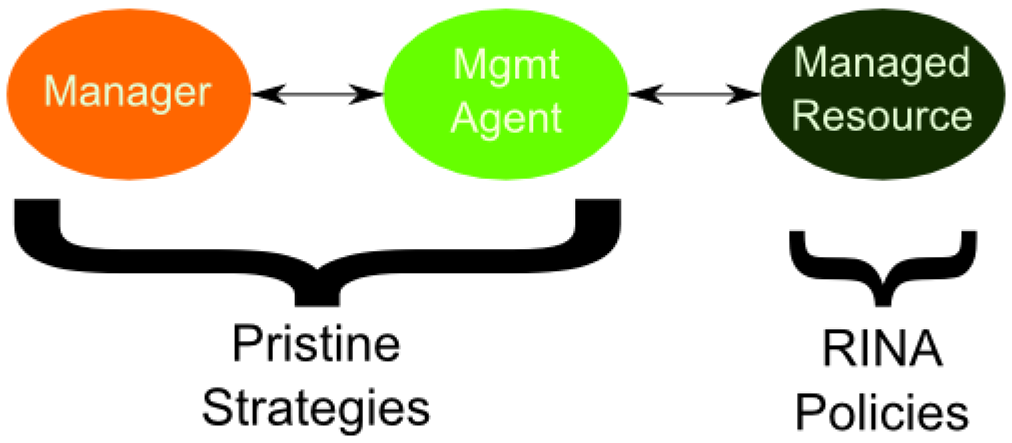


Figure 1. Policy - Strategy separation

1.3. Architecture overview

The following section gives an overview of the DMS. The first diagram shows how the DMS is split into entities performing manager and agent roles. The manager and the agent are members of a single management Distributed Application Facility (DAF), whose sole purpose is to monitor, measure and repair if necessary. Each entity within the DAF is an Application Process (AP). The first diagram shows how the managers and agent APs communicate over a "Management DIF".

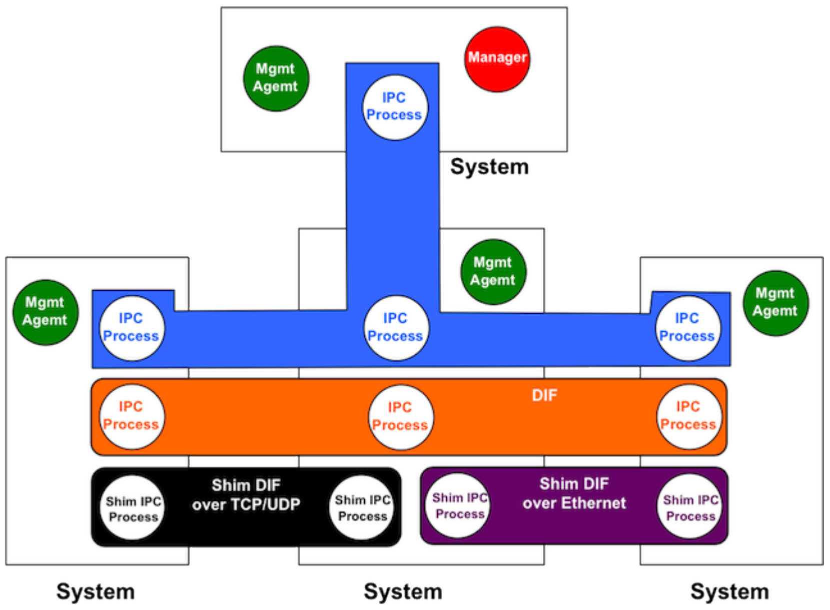


Figure 2. Graphical model of the RINA architecture

1.3.1. PRISTINE work

For the manager and agent to operate effectively, there must be an agreed RIB describing a view of the resources in the agent to the manager.

A draft version of the Managed Object Model used in the RIB was given as part of D5.1. This deliverable further refines it, and attempts some further validation (of the model) through applying typical management activities on the RIB objects. Each management activity is related to configuration, performance and security management functions and is elaborated in the use-cases.

An approach to encapsulate the extra functionality needed to manipulate the RIB objects is presented in the RIB library. This adds features like transactions, multi-version RIB objects, etc. that would be necessary in a running system. This design is described in Section 4, "RIB library design".

For the manager, design and prototyping work is ongoing, but not described in this deliverable. A key aspect of this work is to allow parts of the manager to be written in other programming languages. This reflects the polyglot environment in which these managers operate, where all the components are not written in a single programming language. Hence, special attention is being paid to the design of the RIB library in this regard.

For the agent, the design is presented in Section 5, “Management Agent design” and includes details of the components, and the workflows for agent operation.

1.3.2. Relation to other RINA work

There are three types of "application processes" in the DMS system (Manager, agent and IPC Process). From a review of the existing RINA prototypes and their implementations the following table is produced:

Component	**IRATI project** [a]	**Boston Uni** [b]	**TRIA** [c]	**Required**
Manager	None	None	None	Yes
Agent	Config file	Config file	Config file	Yes
IPC daemon	C++	Java	C	Yes

[a] http://irati.eu/
[b] http://csr.bu.edu/rina
[c] http://www.trianetworksystems.com/triawww/TRIANetworkSystems.html

2. Updated RIB

This section introduces updates to the Resource Information Base (RIB) model that have been made since D5.1. Specifically, the section provides details of changes to the notification model that includes the introduction of a discriminator object used to filter notifications according to policy in a bid to aggregate notifications. The section also provides examples of the sequence of messages required to support subscription and notification flows. Following on from this are the specification of some typical managed object templates that outline in detail an action template, a notification template and a name binding template and provide examples of their implementations.

2.1. Notification specification

In Section 6.1 of [D51] the notification architecture was defined. Due to various reasons explained below, we have decided to change the notifications modelling. In the previous model, each event was encapsulated in an object called EventSource. The object EventSource was the parent (in terms of inheritance relation) of the EventSubscription object which, in addition to the Event, contained a filter.

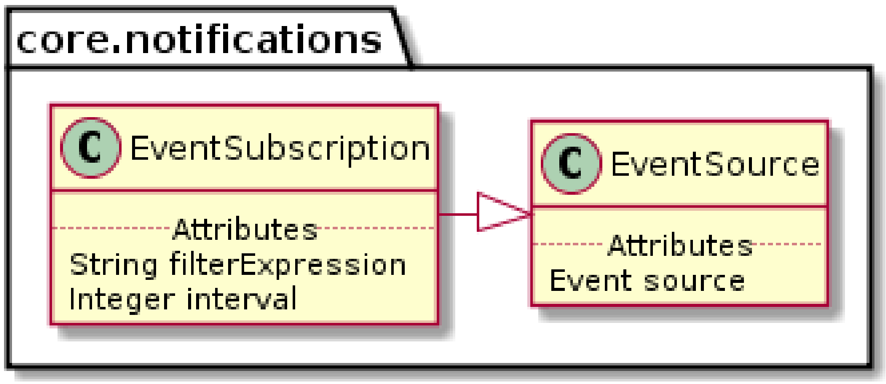


Figure 3. Subscription managed object

The flow chart of this model is shown in Figure 4, "Subscribing for notifications with filtering". The receiver creates a Subscription object on the Managed Object (MO) of interest. This MO then creates a child in its containment tree, with the attributes supplied in the create request. On receipt of the acknowledgement, the Receiver sends a read request for the source attribute on the Subscription managed object. This is acknowledged, and as the notifications are generated, they are passed to the receiver as multiple read responses. A receiver can unsubscribe by sending a delete request to the Subscription object.

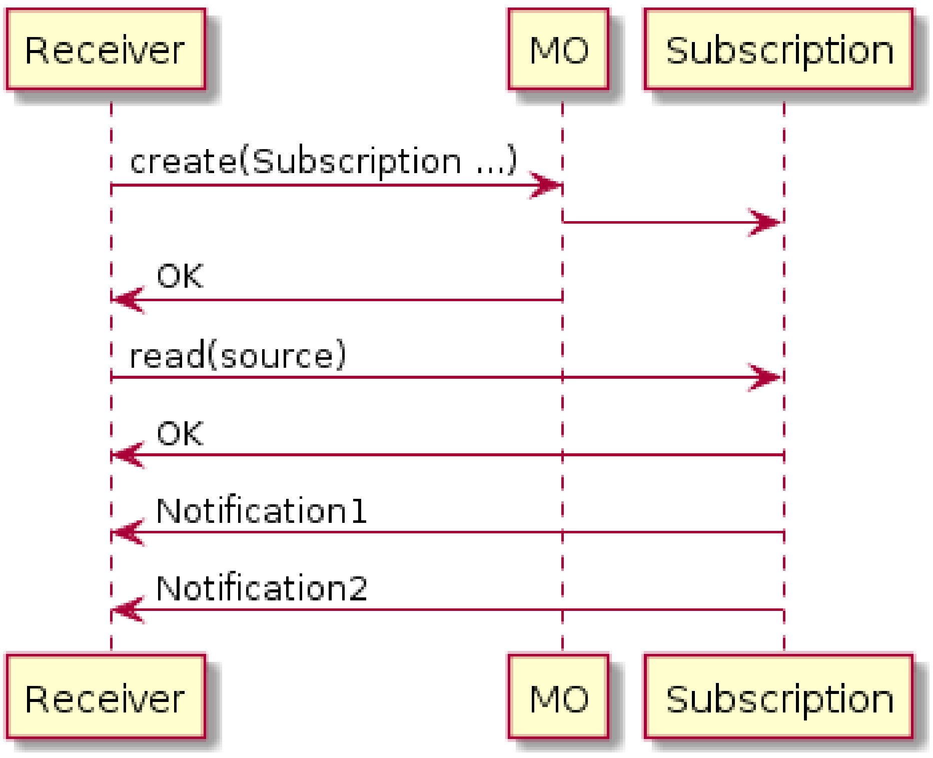


Figure 4. Subscribing for notifications with filtering

However, a couple of important requirements are not addressed with this model. These requirements are:

1. Filtering of various notifications coming from different objects. This is an important requirement that the model must be able to address. Some systems will not use complex filtering (e.g. sensors) but others can, producing many benefits (simplicity in the management and decreasing of the management traffic).
2. Simplicity of the containment model. Subscription to an object means adding an object in the containment tree (The EventSubscription). Many objects emit notifications, so many objects would be added in the containment tree making it more complex.

Following, we will present the new notification model. In order to avoid confusion with the concepts of event, notification, subscription, discriminator and report, we want to define these concepts and use them accordingly from now on.

- **An event** is the specific state change that happens in a system.
- **A notification** is the representation of an event in the MO model.
- **A subscription** to a notification is the observation of a notification, which means that the observer will be noticed about such a notification.
- **A discriminator** is the entity responsible of filtering notifications according to a policy to create an aggregation of data representing this filtering (the report).
- **A report** is a record that represents one or more notifications that have happened in the system. The report can be as simple as only one notification or can be more complex such as the aggregation of various notifications happening in a specific hour of the day.

2.1.1. The Notification Model

This model is based on some important assumptions:

- The subscriber sends a message to the Management Agent (MA) to subscribe to notifications.
- Reports are created through filtering one or more notifications.
- Reports are the entities sent to the Manager.
- Reports will not be stored.
- Notifications will be stored in the log system.
- There will only be one Manager at a time.

It must also fulfil the following requirements:

- System must scale.
- It is possible to see which subscribers are subscribing to each notification.
- Filtering and reporting must scale from trivial filtering and reporting to complex ones.

The inheritance tree

Each notification contains at least two attributes:

i. a list of discriminators, which are the subscribers as explained in the section called "The containment tree" below and

ii. a boolean which determines if the notification is active.

Each object emitting a notification must inherit from the notification class that models this type of notification. This means that all possible notifications (object creation, object deletion, attribute change value, etc.) must be defined in advance. Moreover, the RIB objects inherit all the attributes present in their parents.

As we will see in the next section, notifications are abstract classes present only in the inheritance tree as shown in Figure 5, "Notifcation inheritance".

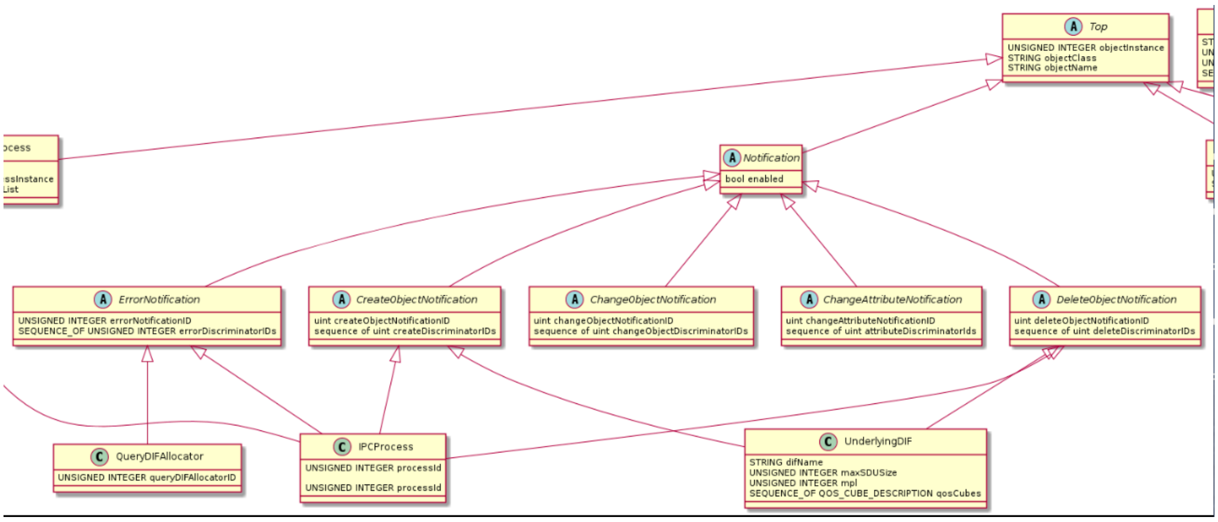

A Top
UNSIGNED INTEGER objectInstance
STRING objectClass
STRING objectName
A Notification
bool enabled
A ErrorNotification
UNSIGNED INTEGER errorNotificationID
SEQUENCE_OF UNSIGNED INTEGER errorDiscriminatorIDs
A CreateObjectNotification
uint createObjectNotificationID
sequence of uint createDiscriminatorIDs
A ChangeObjectNotification
uint changeObjectNotificationID
sequence of uint changeObjectDiscriminatorIDs
A ChangeAttributeNotification
uint changeAttributeNotificationID
sequence of uint attributeDiscriminatorIds
A DeleteObjectNotification
uint deleteObjectNotificationID
sequence of uint deleteDiscriminatorIDs
C QueryDIFAllocator
UNSIGNED INTEGER queryDIFAllocatorID
C IPCProcess
UNSIGNED INTEGER processId
UNSIGNED INTEGER processId
C UnderlyingDIF
STRING difName
UNSIGNED INTEGER maxSDUSize
UNSIGNED INTEGER mpl
SEQUENCE_OF QOS_CUBE_DESCRIPTION qosCubes

Figure 5. Notifcation inheritance

The containment tree

The ForwardingDiscriminator is the object that encapsulates the information about the subscriber and the information about how to create the reports and which information they have to contain. There can be as many ForwardingDiscriminators as needed and they are placed under the Discriminator container under the RIB daemon of the MA object. This allows discriminators to have a higher scope and to be able to subscribe and filter notifications coming from different application processes.

A part from the destination name and instance of the subscriber, each ForwardingDiscriminator has one policy that determines the behaviour of the reports. A Filtering policy determines under which circumstances a report will be created. It can be as simple as one notification $\Rightarrow$ one report or it can depend on a combination of various notifications and other circumstances such as the hour of the day. It also determines the format of the report (attributes, notifications included, etc..)

The entire issue of discriminator persistence can be complex because of the synchronization of saved discriminators through restarts of both ends and/ or connection loss, dynamically establishing connections for reports (and necessarily going through authentication, before sending the report), etc. In PRISTINE we will keep a connection application between the manager and the MA. If this connection is lost, we will remove all the discriminators created by this Manager, but not the reports.

A partial snapshot of the DAF containment diagram is shown in Figure 6, "ForwardingDiscriminator containment" below.

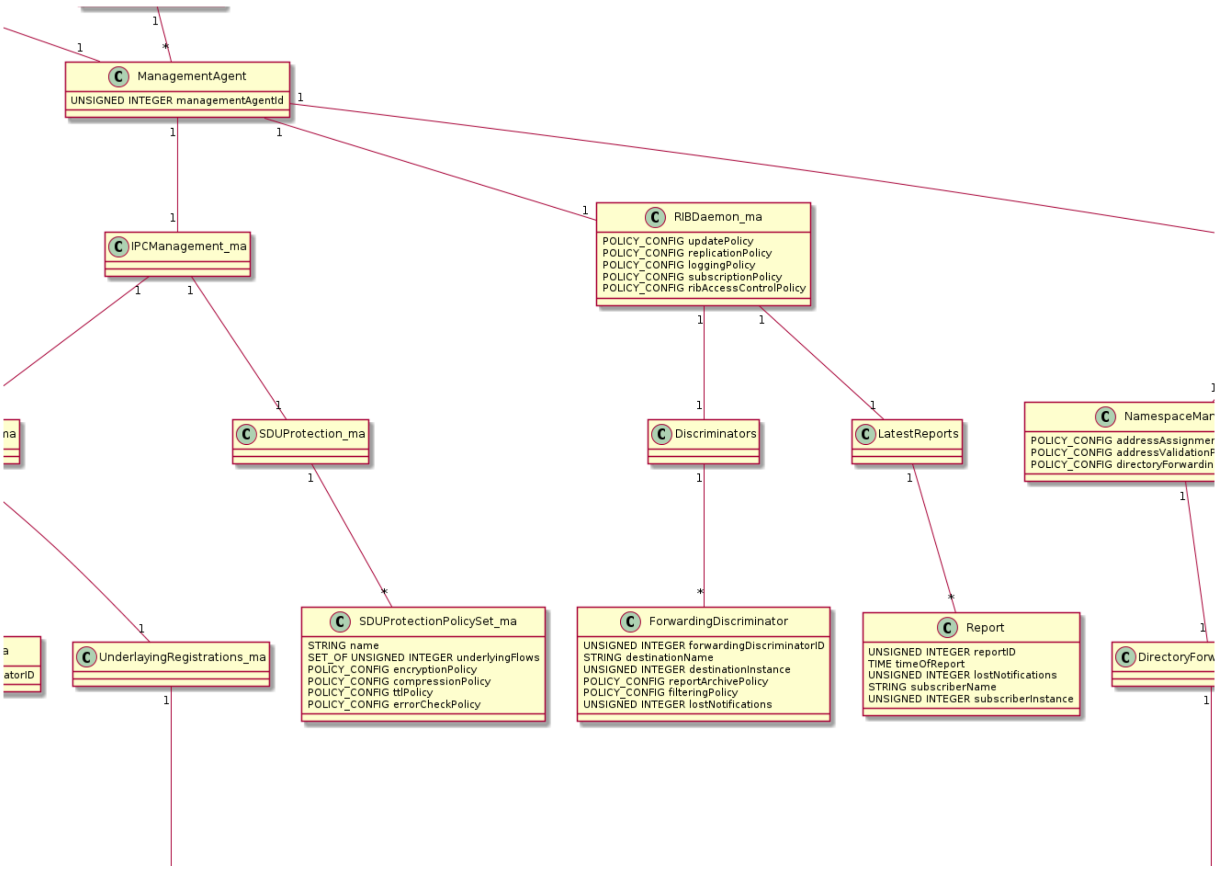

ManagementAgent
UNSIGNED INTEGER managementAgentId
IPCManagement_ma
RIBDaemon_ma
POLICY_CONFIG updatePolicy
POLICY_CONFIG replicationPolicy
POLICY_CONFIG loggingPolicy
POLICY_CONFIG subscriptionPolicy
POLICY_CONFIG ribAccessControlPolicy
SDUProtection_ma
Discriminators
LatestReports
NamespaceMan
POLICY_CONFIG addressAssignmen
POLICY_CONFIG addressValidationP
POLICY_CONFIG directoryForwardin
UnderlayingRegistrations_ma
SDUProtectionPolicySet_ma
STRING name
SET_OF UNSIGNED INTEGER underlyingFlows
POLICY_CONFIG encryptionPolicy
POLICY_CONFIG compressionPolicy
POLICY_CONFIG ttlPolicy
POLICY_CONFIG errorCheckPolicy
ForwardingDiscriminator
UNSIGNED INTEGER forwardingDiscriminatorID
STRING destinationName
UNSIGNED INTEGER destinationInstance
POLICY_CONFIG reportArchivePolicy
POLICY_CONFIG filteringPolicy
UNSIGNED INTEGER lostNotifications
Report
UNSIGNED INTEGER reportID
TIME timeOfReport
UNSIGNED INTEGER lostNotifications
STRING subscriberName
UNSIGNED INTEGER subscriberInstance
DirectoryForw
atorID

Figure 6. ForwardingDiscriminator containment

Subscribing for notifications

The subscription flow is shown in Figure 7, "Subscription sequence chart". The manager sends a create message to the MA which creates a ForwardingDiscriminator object in the appropriate place of the RIB. The input parameters provided determine the notifications where the

ForwardingDiscriminator will be subscribed to and the policy used to filter notifications and send reports. The Discriminator is added to the list of subscribed discriminators (per notification type) of the MO it is subscribed to.

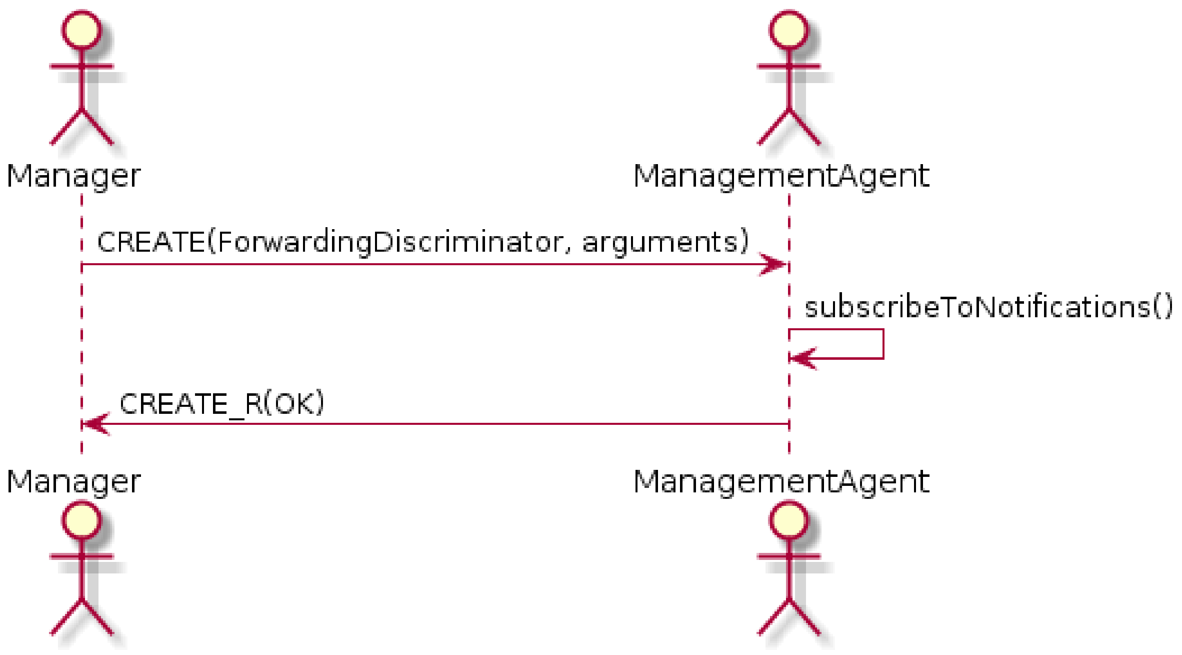

Manager
ManagementAgent
CREATE(ForwardingDiscriminator, arguments)
subscribeToNotifications()
CREATE_R(OK)
Manager
ManagementAgent

Figure 7. Subscription sequence chart

Reporting a notification

The notification flow is shown in Figure 8, “Notification sequence chart”. The MO produces a notification which reaches the Discriminator(s) associated to it. Then, the discriminators filter the notification and decide if they have to create a report. If they create a report, they send it to the Manager via a WRITE with reply requested on a specific object name (e.g. “notificationReport”). It is important to request a reply in the WRITE CDAP message since otherwise it is not possible to control the flow, this is why all the WRITE CDAP messages containing reports will ask for a reply.

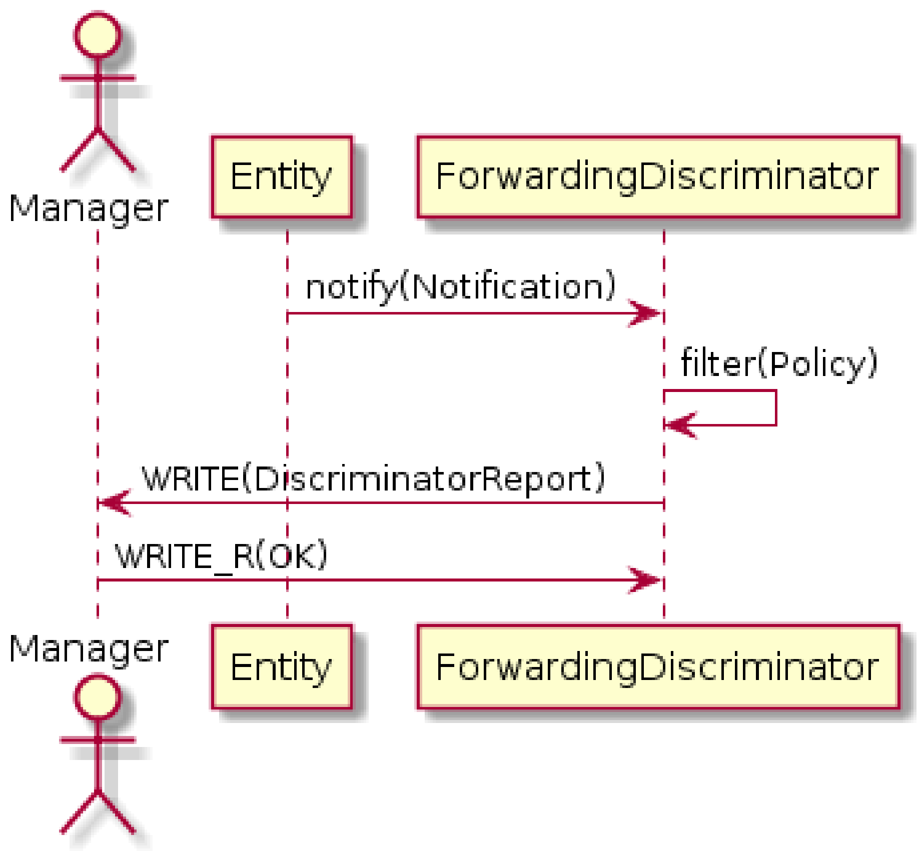


Figure 8. Notification sequence chart

Stopping notification reporting

To stop reporting, one can either stop all the discriminators or stop all the notifications in one or more objects. It is clear that stopping every discriminator will cause the stop of every report. However, conceptually, to stop a discriminator means to stop the catching, filtering and reporting of notifications, it does not mean to stop the notifications. In other words, if no discriminators are present notification will still be launched in response to events. For very complicated situations, for example when the system is collapsing, it may be useful to stop all the notifications meaning that the system is not catching any event.

Even the stop of the notifications or the stop of the discriminators can be done with a single message with the appropriate scope. In the case of the discriminators, the message will target one or more (up to all) ForwardingDiscriminators. For notifications, the message will write a false on the enabled attribute of one or more (up to all) objects emitting notifications. The object can emit more than one notification, however, the granularity of the stop action is at the MO level, meaning that if the enabled attribute of the MO is false, it will not throw any notification.

It is important to note that as it is explained in this section, if the MA loses the connection to the Manager, it will remove all the discriminators corresponding to that manager and hence, no more reports will be created.

Reports and the log system

Reports are designed to minimize the amount of traffic in the network by aggregating notifications in a single report. Moreover, this system allows the manager to design complex filtering procedures in order to combine the triggering of various notifications.

We have designed the report system in a way that reports can also be stored in the RIB allowing an easier access by the manager. Otherwise, if the manager has to access information outside of these reports, it has to access the log system and manually filter the notifications.

However, allowing this storage of reports in addition to the log system is a complex task that can have impacts on system memory. Because of the extra complexity of the whole system, we will not allow the storage of reports in PRISTINE, despite we foresee it as a future development of the RIB system.

In order to allow diagnostic procedures, every notification will be logged in the appropriate system log (as is the norm in such devices).

2.1.2. Notification examples using CDAP

The following section outlines a concrete example of two management flows outlined above (subscribing for and reporting a notifcation).

Manager subscribing to a new neighbor notification

See ForwardingDiscriminator MOC definition.

- *Operation*: CREATE
- *invokeID*: Automatic
- *objClass*: ForwardingDiscriminator

- *objInst*: 1
- *objName*: {ProcessingSystem = 13, Discriminator}
- *objValue*: DISCRIMINATOR_POLICY_CONFIG
  - SET_OF NOTIFICATION notifications :
    - STRING object : neighbour.
    - STRING notification : createObjectNotification.
  - POLICY_CONFIG filteringPolicy :
    - STRING name: 1-to-1
    - STRING version: 1
    - SET_OF POLICY_CONFIG_PARAMETER parameters: -.
- *opCode*: Automatic

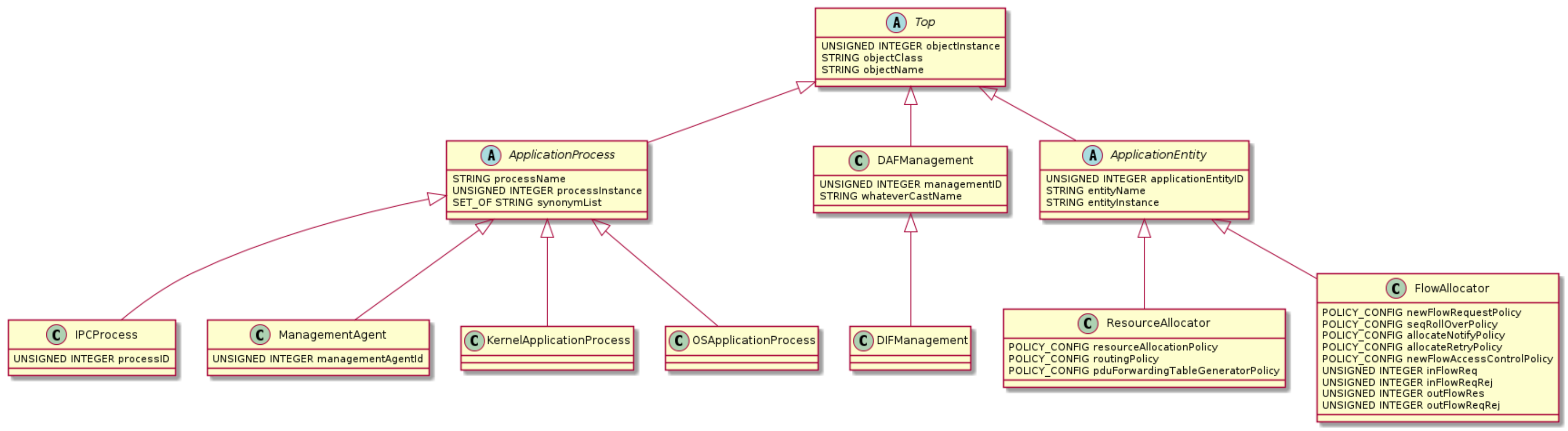
MA sending a report to the manager

- *Operation*: WRITE-with-reply
- *invokeID*: Automatic
- *objClass*: DiscriminatorReport
- *objInst*: 1
- *objName*: {<Manager RIB path>}
- *objValue*: SEQUENCE newNeighborParameters :
  - STRING processName : <newNeighbourName>
  - UNSIGNED INTEGER processInstance : <newNeighborInstance>
  - SET_OF STRINGS synonymList : {<synonym 1>, <synonym 2>}
  - SET_OF UNSIGNED INTEGERS suportingFlows : {<suportingFlow 1>, <suportingFlow 2>}.
- *opCode*: Automatic

2.2. RIB Inheritance tree

In Section 6.2 of [D51] the inheritance diagram is shown. Due to various updates in the concepts involved in the diagram, it has been modified. The most important updates are the following ones:

1. According to the new notification model, the inheritance diagram has included the foreseen common notifications and the RIB objects that inherit from them.
2. There are various components that are in fact application entities. They have been included under ApplicationEntity object.



Top
UNSIGNED INTEGER objectInstance
STRING objectClass
STRING objectName
ApplicationProcess
STRING processName
UNSIGNED INTEGER processInstance
SET_OF STRING synonymList
DAFManagement
UNSIGNED INTEGER managementID
STRING whateverCastName
ApplicationEntity
UNSIGNED INTEGER applicationEntityID
STRING entityName
STRING entityInstance
IPCProcess
UNSIGNED INTEGER processID
ManagementAgent
UNSIGNED INTEGER managementAgentId
KernelApplicationProcess
OSApplicationProcess
DIFManagement
ResourceAllocator
POLICY_CONFIG resourceAllocationPolicy
POLICY_CONFIG routingPolicy
POLICY_CONFIG pduForwardingTableGeneratorPolicy
FlowAllocator
POLICY_CONFIG newFlowRequestPolicy
POLICY_CONFIG seqRollOverPolicy
POLICY_CONFIG allocateNotifyPolicy
POLICY_CONFIG allocateRetryPolicy
POLICY_CONFIG newFlowAccessControlPolicy
UNSIGNED INTEGER inFlowReq
UNSIGNED INTEGER inFlowReqRej
UNSIGNED INTEGER outFlowRes
UNSIGNED INTEGER outFlowReqRej

Figure 9. Top Managed Object inheritance tree

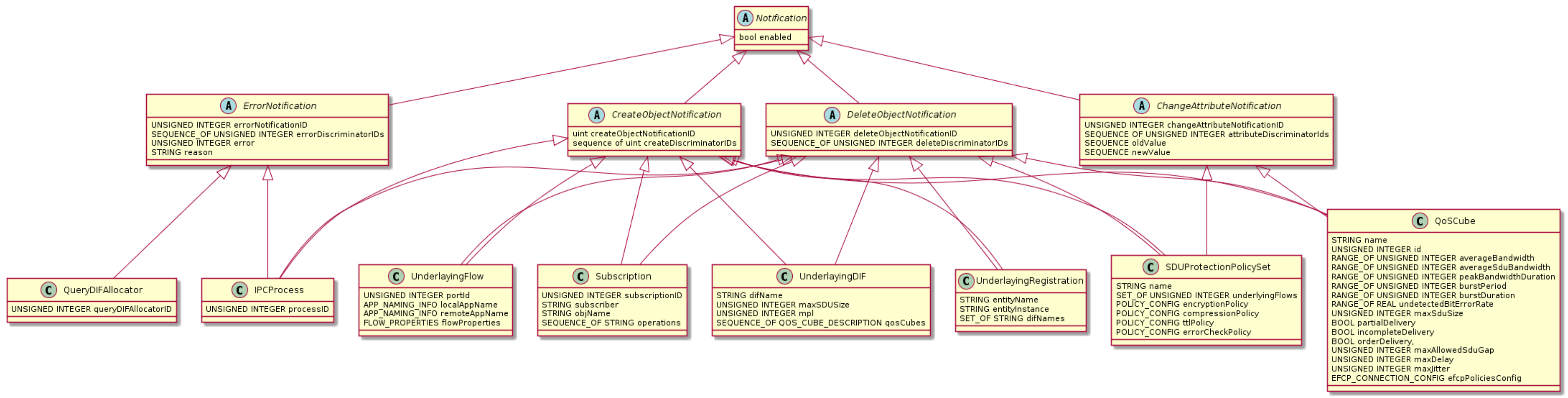

Notification
bool enabled
ErrorNotification
UNSIGNED INTEGER errorNotificationID
SEQUENCE_OF UNSIGNED INTEGER errorDiscriminatorIDs
UNSIGNED INTEGER error
STRING reason
CreateObjectNotification
uint createObjectNotificationID
sequence of uint createDiscriminatorIDs
DeleteObjectNotification
UNSIGNED INTEGER deleteObjectNotificationID
SEQUENCE_OF UNSIGNED INTEGER deleteDiscriminatorIDs
ChangeAttributeNotification
UNSIGNED INTEGER changeAttributeNotificationID
SEQUENCE OF UNSIGNED INTEGER attributeDiscriminatorIds
SEQUENCE oldValue
SEQUENCE newValue
QueryDIFAllocator
UNSIGNED INTEGER queryDIFAllocatorID
IPCProcess
UNSIGNED INTEGER processID
UnderlayingFlow
UNSIGNED INTEGER portId
APP_NAMING_INFO localAppName
APP_NAMING_INFO remoteAppName
FLOW_PROPERTIES flowProperties
Subscription
UNSIGNED INTEGER subscriptionID
STRING subscriber
STRING objName
SEQUENCE_OF STRING operations
UnderlayingDIF
STRING difName
UNSIGNED INTEGER maxSDUSize
UNSIGNED INTEGER mpl
SEQUENCE_OF QOS_CUBE_DESCRIPTION qosCubes
UnderlayingRegistration
STRING entityName
STRING entityInstance
SET_OF STRING difNames
SDUProtectionPolicySet
STRING name
SET_OF UNSIGNED INTEGER underlyingFlows
POLICY_CONFIG encryptionPolicy
POLICY_CONFIG compressionPolicy
POLICY_CONFIG ttlPolicy
POLICY_CONFIG errorCheckPolicy
QoSCube
STRING name
UNSIGNED INTEGER id
RANGE_OF UNSIGNED INTEGER averageBandwidth
RANGE_OF UNSIGNED INTEGER averageSduBandwidth
RANGE_OF UNSIGNED INTEGER peakBandwidthDuration
RANGE_OF UNSIGNED INTEGER burstPeriod
RANGE_OF UNSIGNED INTEGER burstDuration
RANGE_OF REAL undetectedBitErrorRate
UNSIGNED INTEGER maxSduSize
BOOL partialDelivery
BOOL incompleteDelivery
BOOL orderDelivery.
UNSIGNED INTEGER maxAllowedSduGap
UNSIGNED INTEGER maxDelay
UNSIGNED INTEGER maxJitter
EFCP_CONNECTION_CONFIG efcpPoliciesConfig

Figure 10. Notification Managed Object inheritance tree 1

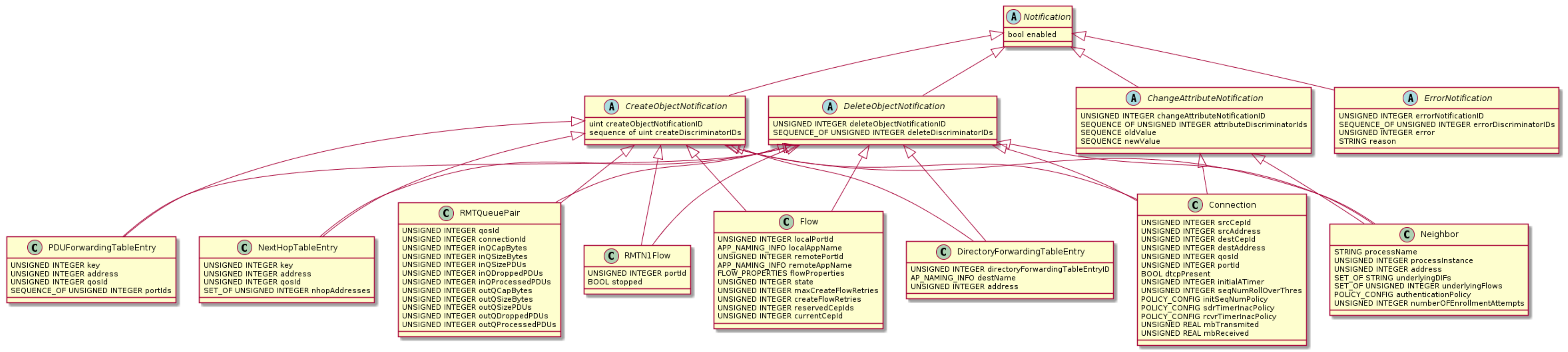

Notification
bool enabled
CreateObjectNotification
uint createObjectNotificationID
sequence of uint createDiscriminatorIDs
DeleteObjectNotification
UNSIGNED INTEGER deleteObjectNotificationID
SEQUENCE_OF UNSIGNED INTEGER deleteDiscriminatorIDs
ChangeAttributeNotification
UNSIGNED INTEGER changeAttributeNotificationID
SEQUENCE OF UNSIGNED INTEGER attributeDiscriminatorIds
SEQUENCE oldValue
SEQUENCE newValue
ErrorNotification
UNSIGNED INTEGER errorNotificationID
SEQUENCE_OF UNSIGNED INTEGER errorDiscriminatorIDs
UNSIGNED INTEGER error
STRING reason
PDUForwardingTableEntry
UNSIGNED INTEGER key
UNSIGNED INTEGER address
UNSIGNED INTEGER qosId
SEQUENCE_OF UNSIGNED INTEGER portIds
NextHopTableEntry
UNSIGNED INTEGER key
UNSIGNED INTEGER address
UNSIGNED INTEGER qosId
SET_OF UNSIGNED INTEGER nhopAddresses
RMTQueuePair
UNSIGNED INTEGER qosId
UNSIGNED INTEGER connectionId
UNSIGNED INTEGER inQCapBytes
UNSIGNED INTEGER inQSizeBytes
UNSIGNED INTEGER inQSizePDUs
UNSIGNED INTEGER inQDroppedPDUs
UNSIGNED INTEGER inQProcessedPDUs
UNSIGNED INTEGER outQCapBytes
UNSIGNED INTEGER outQSizeBytes
UNSIGNED INTEGER outQSizePDUs
UNSIGNED INTEGER outQDroppedPDUs
UNSIGNED INTEGER outQProcessedPDUs
RMTN1Flow
UNSIGNED INTEGER portId
BOOL stopped
Flow
UNSIGNED INTEGER localPortId
APP_NAMING_INFO localAppName
UNSIGNED INTEGER remotePortId
APP_NAMING_INFO remoteAppName
FLOW_PROPERTIES flowProperties
UNSIGNED INTEGER state
UNSIGNED INTEGER maxCreateFlowRetries
UNSIGNED INTEGER createFlowRetries
UNSIGNED INTEGER reservedCepIds
UNSIGNED INTEGER currentCepId
DirectoryForwardingTableEntry
UNSIGNED INTEGER directoryForwardingTableEntryID
AP_NAMING_INFO destName
UNSIGNED INTEGER address
Connection
UNSIGNED INTEGER srcCepId
UNSIGNED INTEGER srcAddress
UNSIGNED INTEGER destCepId
UNSIGNED INTEGER destAddress
UNSIGNED INTEGER qosId
UNSIGNED INTEGER portId
BOOL dtcpPresent
UNSIGNED INTEGER initialATimer
UNSIGNED INTEGER seqNumRollOverThres
POLICY_CONFIG initSeqNumPolicy
POLICY_CONFIG sdrTimerInacPolicy
POLICY_CONFIG rcvrTimerInacPolicy
UNSIGNED REAL mbTransmited
UNSIGNED REAL mbReceived
Neighbor
STRING processName
UNSIGNED INTEGER processInstance
UNSIGNED INTEGER address
SET_OF STRING underlyingDIFs
SET_OF UNSIGNED INTEGER underlyingFlows
POLICY_CONFIG authenticationPolicy
UNSIGNED INTEGER numberOFEnrollmentAttempts

Figure 11. Notification Managed Object inheritance tree 2

2.3. RIB containment tree

In Section 6.3 of [D51] the containment diagram is shown. Due to various updates in the concepts involved in the diagram, it has been modified, and presented below (Figure 12, "Containment tree with upper Managed Objects" - Figure 15, "Containment sub-tree from the IPC Process Managed Object (2 of 2)."). A summary of the updates follows:

1. The policies that where RIB objects by their own, have been moved as attributes of their container objects. This reduces the overall number of RIB objects.
2. The Management Agent RIB has been added. Their contained objects are very similar to those contained by the IPC process, since a Management Agent is an application process and thus, it shares many components with an IPC process which is also an application process. To avoid confusions, we have renamed the shared components with a _ma if they are part of the Management Agent, however, conceptually, they are the same object and hence, they are the same template. This means that the RIBDaemon_ma is the RIBDaemon of the management agent and its behaviour is determined by the RIBDaemon template.
3. The Discriminator model explained above has been included under the RIBDaemon object of the management agent.
4. The ApplicationEntity has been removed from the containment tree since we have realized that in fact it had no sense as a named object. The FlowAllocator and the ResourceAllocator have been foreseen as ApplicationEntities and this relation has been exposed in the inheritance and in the containment.
5. All the objects that where in a 1 to N relation have been moved under a "container" object, e.g. UnderlyingDIF is under UnderlyingDIFs container object. This change has been introduced to allow an easier CDAP targetting of the contained objects. Now, to get all the underlying DIFs, the Manager can do it with a simple CDAP message to the UnderlyingDIFs with scope one.
6. There have been changes in the upper layers of the containment tree. To accommodate the RINA reference model, we have included the kernel and the OS as application processes and also the DIF and the DAF objects. Moreover, in order for the Manager to use the SDK that is being

defined in WP2, we have added a software and a hardware objects in the RIB.

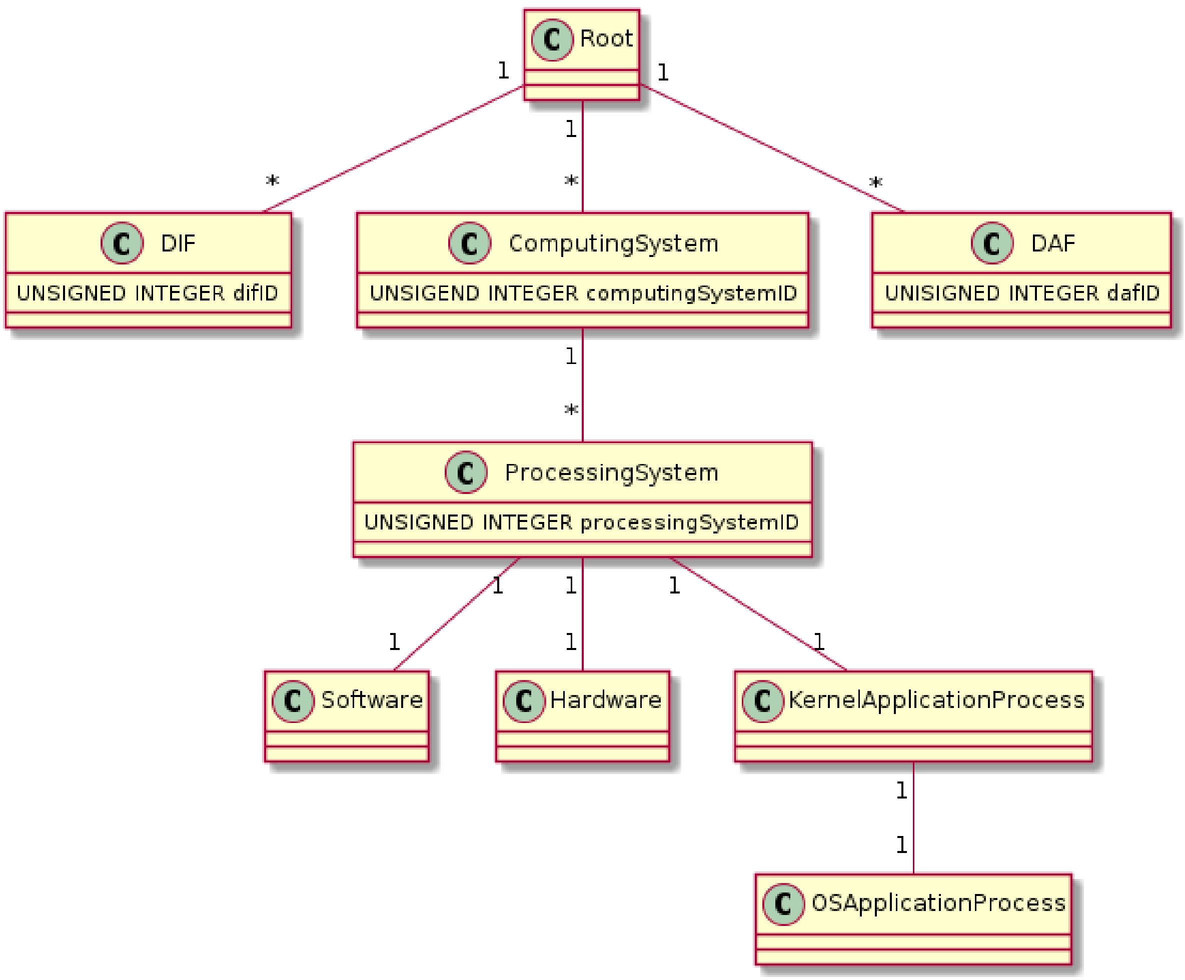


Figure 12. Containment tree with upper Managed Objects

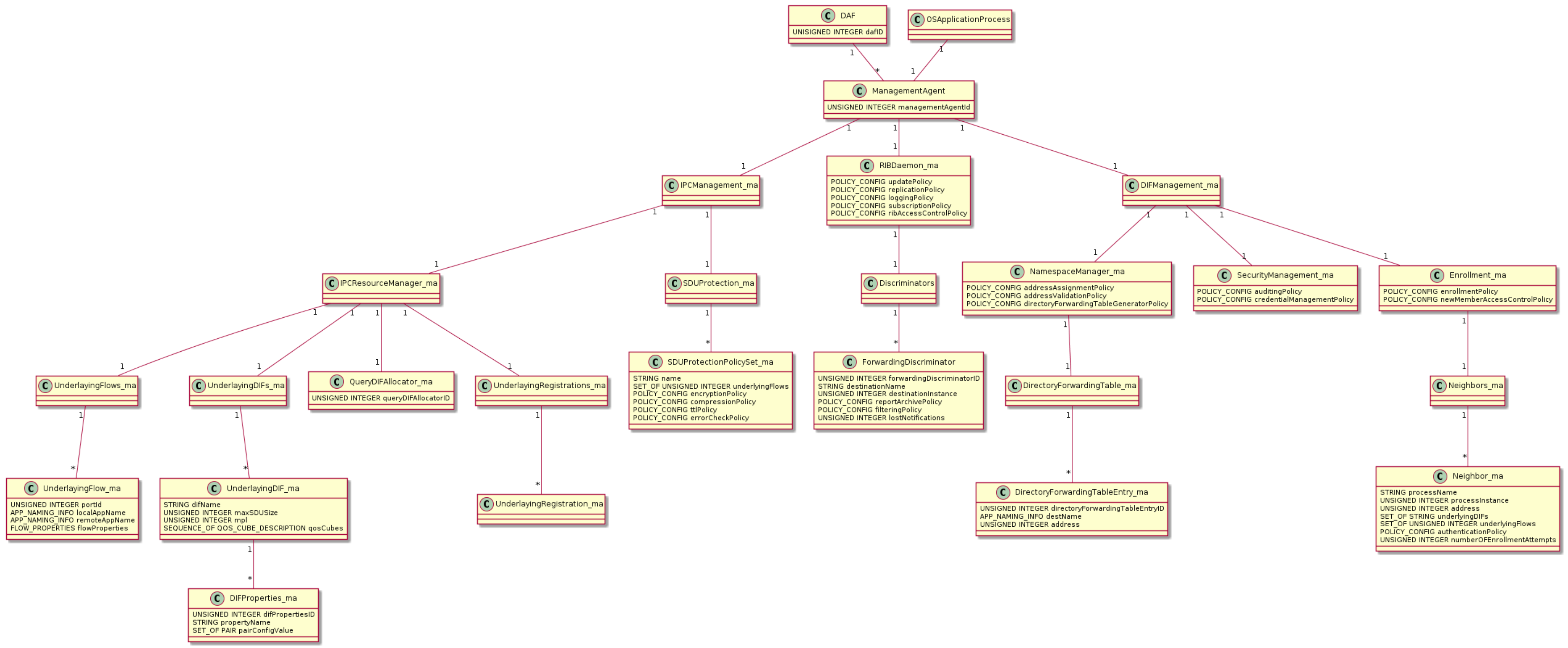

DAF
UNISIGNED INTEGER dafID
OSApplicationProcess
ManagementAgent
UNSIGNED INTEGER managementAgentId
IPCManagement_ma
RIBDaemon_ma
POLICY_CONFIG updatePolicy
POLICY_CONFIG replicationPolicy
POLICY_CONFIG loggingPolicy
POLICY_CONFIG subscriptionPolicy
POLICY_CONFIG ribAccessControlPolicy
DIFManagement_ma
IPCResourceManager_ma
SDUProtection_ma
Discriminators
NamespaceManager_ma
POLICY_CONFIG addressAssignmentPolicy
POLICY_CONFIG addressValidationPolicy
POLICY_CONFIG directoryForwardingTableGeneratorPolicy
SecurityManagement_ma
POLICY_CONFIG auditingPolicy
POLICY_CONFIG credentialManagementPolicy
Enrollment_ma
POLICY_CONFIG enrollmentPolicy
POLICY_CONFIG newMemberAccessControlPolicy
UnderlayingFlows_ma
UnderlayingDIFs_ma
QueryDIFAllocator_ma
UNSIGNED INTEGER queryDIFAllocatorID
UnderlayingRegistrations_ma
SDUProtectionPolicySet_ma
STRING name
SET_OF UNSIGNED INTEGER underlyingFlows
POLICY_CONFIG encryptionPolicy
POLICY_CONFIG compressionPolicy
POLICY_CONFIG ttlPolicy
POLICY_CONFIG errorCheckPolicy
ForwardingDiscriminator
UNSIGNED INTEGER forwardingDiscriminatorID
STRING destinationName
UNSIGNED INTEGER destinationInstance
POLICY_CONFIG reportArchivePolicy
POLICY_CONFIG filteringPolicy
UNSIGNED INTEGER lostNotifications
DirectoryForwardingTable_ma
Neighbors_ma
UnderlayingFlow_ma
UNSIGNED INTEGER portId
APP_NAMING_INFO localAppName
APP_NAMING_INFO remoteAppName
FLOW_PROPERTIES flowProperties
UnderlayingDIF_ma
STRING difName
UNSIGNED INTEGER maxSDUSize
UNSIGNED INTEGER mpl
SEQUENCE_OF QOS_CUBE_DESCRIPTION qosCubes
UnderlayingRegistration_ma
DirectoryForwardingTableEntry_ma
UNSIGNED INTEGER directoryForwardingTableEntryID
APP_NAMING_INFO destName
UNSIGNED INTEGER address
Neighbor_ma
STRING processName
UNSIGNED INTEGER processInstance
UNSIGNED INTEGER address
SET_OF STRING underlyingDIFs
SET_OF UNSIGNED INTEGER underlyingFlows
POLICY_CONFIG authenticationPolicy
UNSIGNED INTEGER numberOFEnrollmentAttempts
DIFProperties_ma
UNSIGNED INTEGER difPropertiesID
STRING propertyName
SET_OF PAIR pairConfigValue

Figure 13. Containment sub-tree from the Management Agent Managed Object

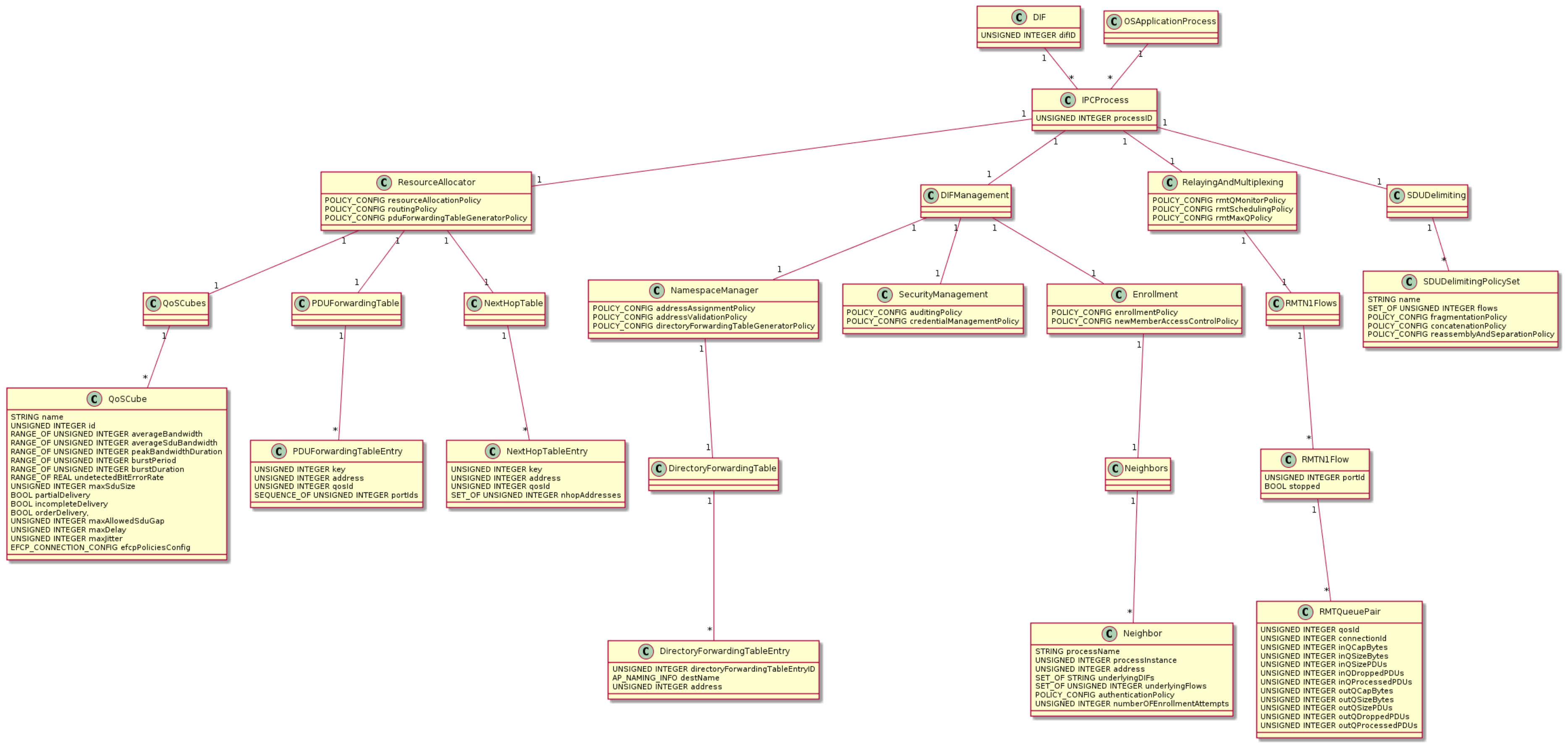

DIF
UNSIGNED INTEGER difID
OSApplicationProcess
IPCProcess
UNSIGNED INTEGER processID
ResourceAllocator
POLICY_CONFIG resourceAllocationPolicy
POLICY_CONFIG routingPolicy
POLICY_CONFIG pduForwardingTableGeneratorPolicy
DIFManagement
RelayingAndMultiplexing
POLICY_CONFIG rmtQMonitorPolicy
POLICY_CONFIG rmtSchedulingPolicy
POLICY_CONFIG rmtMaxQPolicy
SDUDelimiting
SDUDelimitingPolicySet
STRING name
SET_OF UNSIGNED INTEGER flows
POLICY_CONFIG fragmentationPolicy
POLICY_CONFIG concatenationPolicy
POLICY_CONFIG reassemblyAndSeparationPolicy
QoSCubes
PDUForwardingTable
NextHopTable
NamespaceManager
POLICY_CONFIG addressAssignmentPolicy
POLICY_CONFIG addressValidationPolicy
POLICY_CONFIG directoryForwardingTableGeneratorPolicy
SecurityManagement
POLICY_CONFIG auditingPolicy
POLICY_CONFIG credentialManagementPolicy
Enrollment
POLICY_CONFIG enrollmentPolicy
POLICY_CONFIG newMemberAccessControlPolicy
RMTN1Flows
QoSCube
STRING name
UNSIGNED INTEGER id
RANGE_OF UNSIGNED INTEGER averageBandwidth
RANGE_OF UNSIGNED INTEGER averageSduBandwidth
RANGE_OF UNSIGNED INTEGER peakBandwidthDuration
RANGE_OF UNSIGNED INTEGER burstPeriod
RANGE_OF UNSIGNED INTEGER burstDuration
RANGE_OF REAL undetectedBitErrorRate
UNSIGNED INTEGER maxSduSize
BOOL partialDelivery
BOOL incompleteDelivery
BOOL orderDelivery,
UNSIGNED INTEGER maxAllowedSduGap
UNSIGNED INTEGER maxDelay
UNSIGNED INTEGER maxJitter
EFCP_CONNECTION_CONFIG efcpPoliciesConfig
PDUForwardingTableEntry
UNSIGNED INTEGER key
UNSIGNED INTEGER address
UNSIGNED INTEGER qosId
SEQUENCE_OF UNSIGNED INTEGER portIds
NextHopTableEntry
UNSIGNED INTEGER key
UNSIGNED INTEGER address
UNSIGNED INTEGER qosId
SET_OF UNSIGNED INTEGER nhopAddresses
DirectoryForwardingTable
Neighbors
RMTN1Flow
UNSIGNED INTEGER portId
BOOL stopped
DirectoryForwardingTableEntry
UNSIGNED INTEGER directoryForwardingTableEntryID
AP_NAMING_INFO destName
UNSIGNED INTEGER address
Neighbor
STRING processName
UNSIGNED INTEGER processInstance
UNSIGNED INTEGER address
SET_OF STRING underlyingDIFs
SET_OF UNSIGNED INTEGER underlyingFlows
POLICY_CONFIG authenticationPolicy
UNSIGNED INTEGER numberOFEnrollmentAttempts
RMTQueuePair
UNSIGNED INTEGER qosId
UNSIGNED INTEGER connectionId
UNSIGNED INTEGER inQCapBytes
UNSIGNED INTEGER inQSizeBytes
UNSIGNED INTEGER inQSizePDUs
UNSIGNED INTEGER inQDroppedPDUs
UNSIGNED INTEGER inQProcessedPDUs
UNSIGNED INTEGER outQCapBytes
UNSIGNED INTEGER outQSizeBytes
UNSIGNED INTEGER outQSizePDUs
UNSIGNED INTEGER outQDroppedPDUs
UNSIGNED INTEGER outQProcessedPDUs

Figure 14. Containment sub-tree from the IPC Process Managed Object (1 of 2).

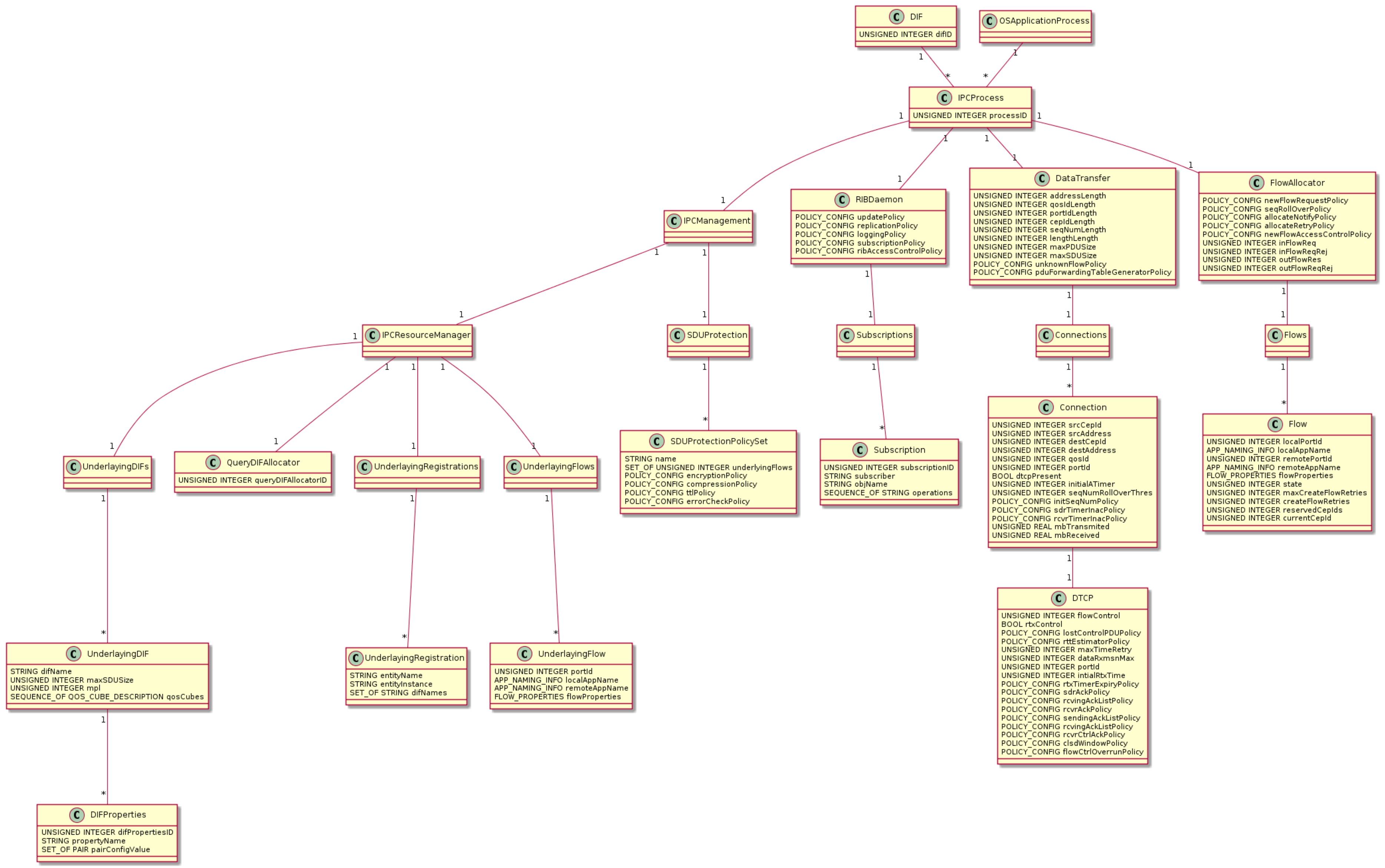

DIF
UNSIGNED INTEGER difID
OSApplicationProcess
IPCProcess
UNSIGNED INTEGER processID
IPCManagement
RIBDaemon
POLICY_CONFIG updatePolicy
POLICY_CONFIG replicationPolicy
POLICY_CONFIG loggingPolicy
POLICY_CONFIG subscriptionPolicy
POLICY_CONFIG ribAccessControlPolicy
DataTransfer
UNSIGNED INTEGER addressLength
UNSIGNED INTEGER qosIdLength
UNSIGNED INTEGER portIdLength
UNSIGNED INTEGER cepIdLength
UNSIGNED INTEGER seqNumLength
UNSIGNED INTEGER lengthLength
UNSIGNED INTEGER maxPDUSize
UNSIGNED INTEGER maxSDUSize
POLICY_CONFIG unknownFlowPolicy
POLICY_CONFIG pduForwardingTableGeneratorPolicy
FlowAllocator
POLICY_CONFIG newFlowRequestPolicy
POLICY_CONFIG seqRollOverPolicy
POLICY_CONFIG allocateNotifyPolicy
POLICY_CONFIG allocateRetryPolicy
POLICY_CONFIG newFlowAccessControlPolicy
UNSIGNED INTEGER inFlowReq
UNSIGNED INTEGER inFlowReqRej
UNSIGNED INTEGER outFlowRes
UNSIGNED INTEGER outFlowReqRej
IPCResourceManager
SDUProtection
Subscriptions
Connections
Flows
Connection
UNSIGNED INTEGER srcCepId
UNSIGNED INTEGER srcAddress
UNSIGNED INTEGER destCepId
UNSIGNED INTEGER destAddress
UNSIGNED INTEGER qosId
UNSIGNED INTEGER portId
BOOL dtcpPresent
UNSIGNED INTEGER initialATimer
UNSIGNED INTEGER seqNumRollOverThres
POLICY_CONFIG initSeqNumPolicy
POLICY_CONFIG sdrTimerInacPolicy
POLICY_CONFIG rcvrTimerInacPolicy
UNSIGNED REAL mbTransmited
UNSIGNED REAL mbReceived
Flow
UNSIGNED INTEGER localPortId
APP_NAMING_INFO localAppName
UNSIGNED INTEGER remotePortId
APP_NAMING_INFO remoteAppName
FLOW_PROPERTIES flowProperties
UNSIGNED INTEGER state
UNSIGNED INTEGER maxCreateFlowRetries
UNSIGNED INTEGER createFlowRetries
UNSIGNED INTEGER reservedCepIds
UNSIGNED INTEGER currentCepId
UnderlayingDIFs
QueryDIFAllocator
UNSIGNED INTEGER queryDIFAllocatorID
UnderlayingRegistrations
UnderlayingFlows
SDUProtectionPolicySet
STRING name
SET_OF UNSIGNED INTEGER underlyingFlows
POLICY_CONFIG encryptionPolicy
POLICY_CONFIG compressionPolicy
POLICY_CONFIG ttlPolicy
POLICY_CONFIG errorCheckPolicy
Subscription
UNSIGNED INTEGER subscriptionID
STRING subscriber
STRING objName
SEQUENCE_OF STRING operations
DTCP
UNSIGNED INTEGER flowControl
BOOL rtxControl
POLICY_CONFIG lostControlPDUPolicy
POLICY_CONFIG rttEstimatorPolicy
UNSIGNED INTEGER maxTimeRetry
UNSIGNED INTEGER dataRxmsnMax
UNSIGNED INTEGER portId
UNSIGNED INTEGER intialRtxTime
POLICY_CONFIG rtxTimerExpiryPolicy
POLICY_CONFIG sdrAckPolicy
POLICY_CONFIG rcvingAckListPolicy
POLICY_CONFIG rcvrAckPolicy
POLICY_CONFIG sendingAckListPolicy
POLICY_CONFIG rcvingAckListPolicy
POLICY_CONFIG rcvrCtrlAckPolicy
POLICY_CONFIG clsdWindowPolicy
POLICY_CONFIG flowCtrlOverrunPolicy
UnderlayingDIF
STRING difName
UNSIGNED INTEGER maxSDUSize
UNSIGNED INTEGER mpl
SEQUENCE_OF QOS_CUBE_DESCRIPTION qosCubes
UnderlayingRegistration
STRING entityName
STRING entityInstance
SET_OF STRING difNames
UnderlayingFlow
UNSIGNED INTEGER portId
APP_NAMING_INFO localAppName
APP_NAMING_INFO remoteAppName
FLOW_PROPERTIES flowProperties
DIFProperties
UNSIGNED INTEGER difPropertiesID
STRING propertyName
SET_OF PAIR pairConfigValue

Figure 15. Containment sub-tree from the IPC Process Managed Object (2 of 2).

2.4. Managed object templates

This section presents the formal update to the templates used to specify Managed Objects.

2.4.1. Managed object class template

As defined in [D51], the definition of a managed object class involves the specification of the attributes it possesses, operations that may be performed upon it, notifications that it may issue and its relationships with other managed objects. An attribute of a managed object has an associated value, represented by a simple or a complex data type. Some attributes may be "public" i.e. directly accessible at the CDAP protocol level. Public attributes are individually-addressable attributes of the object, modelled as contained object classes, which only support basic operations with little or no side effects.

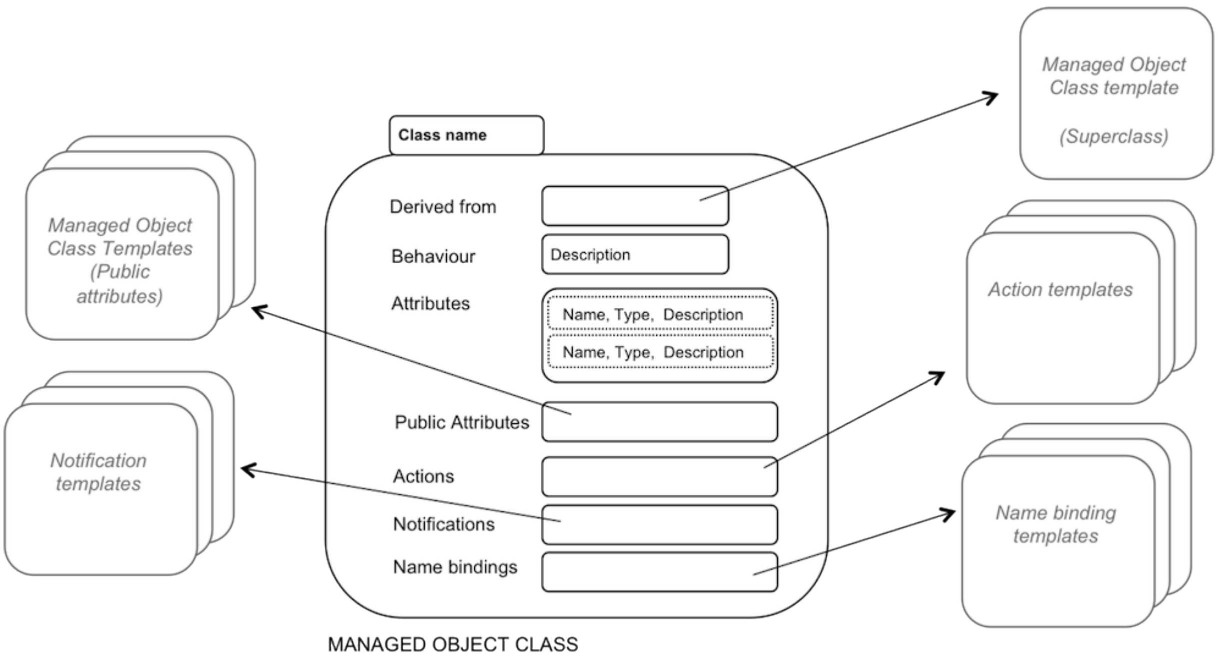


Figure 16. Managed object class template

The managed object class template needs to capture the information defined in the following bullet points.

- **Class name** (mandatory): The fully qualified name of the managed object class.
- **Derived from** (mandatory): Name of the class from which this managed object class is derived (points to the definition of the template).

- **Behaviour** (optional): Description of any behaviour that is not specific to a certain action, attribute or notification of the class. Examples of such behaviours are interrelationships between the values of attributes or relationships between elements of the managed object class and the underlying resource that the managed object models.
- **Attributes** (mandatory): Definition of the attributes of the object that are not individually-addressable. That is, only the object defined by the managed object class can be the target of an operation, but not its individual attributes. For each attribute the template must specify a *name*, a *type* and a *description*. The basic attribute types and the rules for creating complex attribute types are defined in D5.1.
- **Public attributes** (optional): Public attributes are individually-addressable attributes of the object, modelled as contained object classes, which only support basic operations with little or no side effects. For each public attribute of the managed object class the template has to provide the name of the public attribute and the name of the class of the contained object that models the public attribute (which serves as a pointer to the template defining that managed object class).
- **Actions** (mandatory): Pointers to the templates defining the actions that can be applied to this type of managed objects. Only the following actions are allowed: read, cancel_read, write, start and stop. The create and delete behaviour may depend on where objects of this particular class are instantiated in the containment tree, and therefore modelled by each particular "Name binding" template. Actions that are not specified are assumed to be forbidden for this MO.
- **Notifications** (optional): Name of the Notifications that this managed object class generates.
- **Name bindings**. Points to the different templates that define the name bindings for this managed object class. Each name binding defines a MO class that can contain instances of this object, the attribute of this object that will uniquely identify it within the scope of the container managed object and the description of the create and delete operations.

2.4.2. Action template

An action defines an operation on a managed object. Only the create, delete, write, read, cancel_read, start and stop operations are allowed.

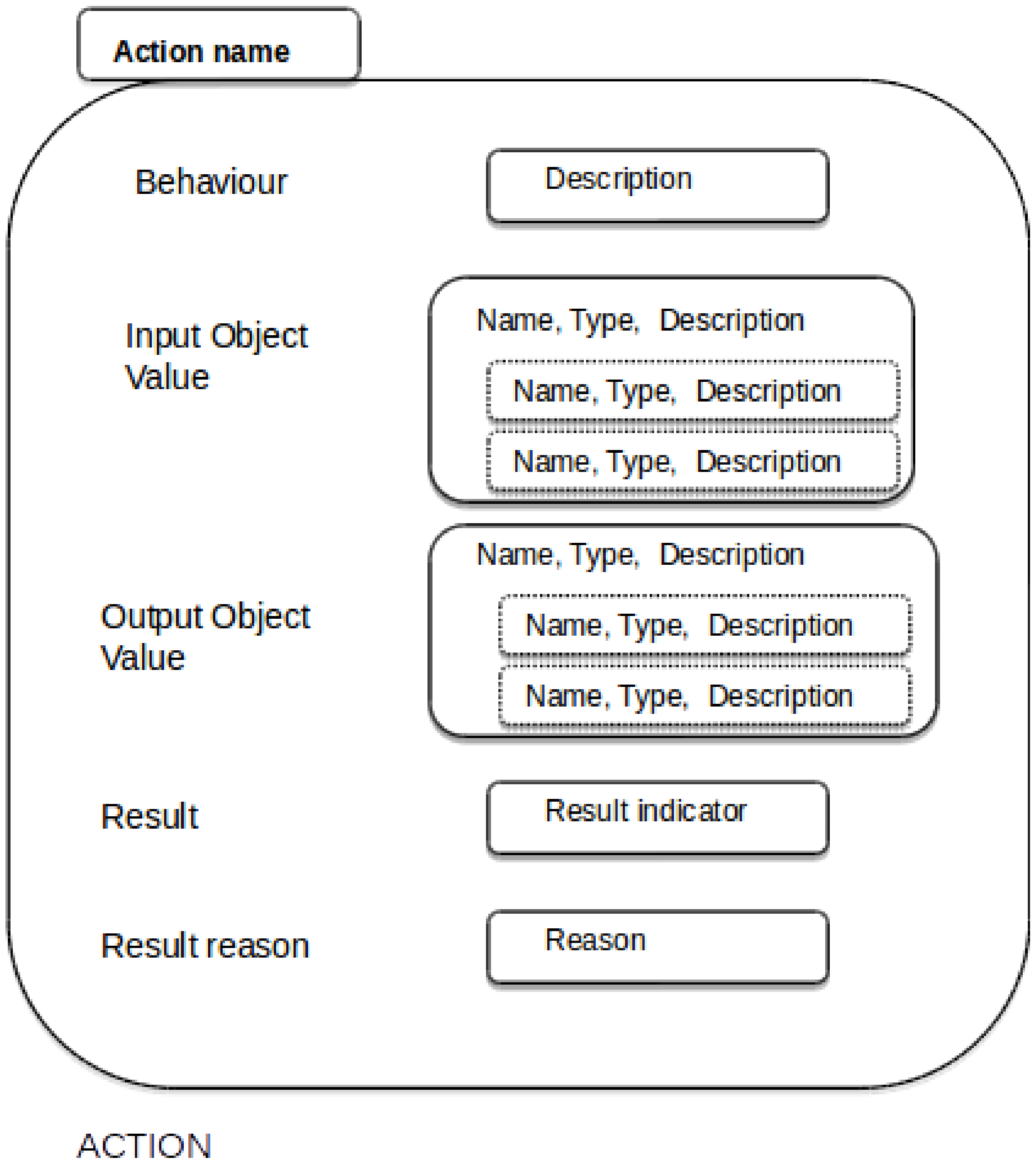


Figure 17. Action template

The action template needs to capture the information defined in the following bullet points.

- **ACTION** (mandatory). Can only be "CREATE, "DELETE", "WRITE", "READ", "CANCEL_READ", "START", "STOP".
- **BEHAVIOUR** (mandatory). Describes the preconditions under which the action can be executed, any side effects that the action execution may trigger and the effect upon the managed object.
- **INPUT OBJECT VALUE** (optional). Description of the input object value required by the action (name, type, textual description).

- **OUTPUT OBJECT VALUE** (optional). Description of the output object value given by the action (name, type, textual description).
- **RESULT** (mandatory). The result of an operation, indicating its success, partial success in the case of synchronized operations, or reason for failure.
- **RESULT REASON** (optional) Additional explanation of Result.

2.4.3. Notification template

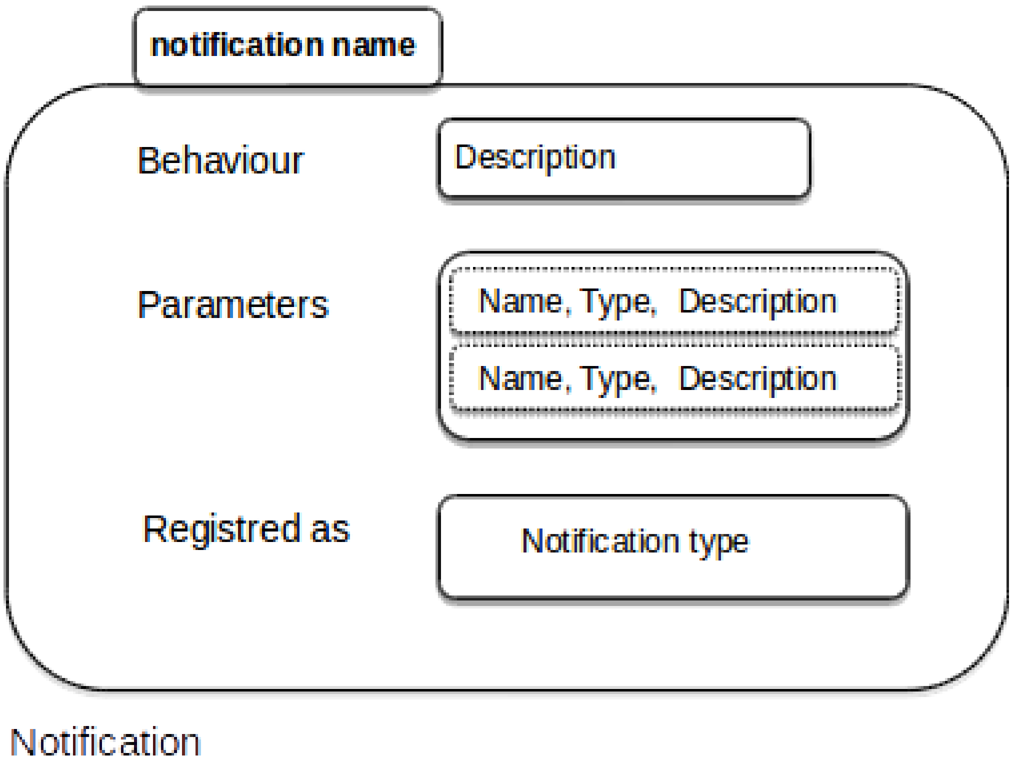


Figure 18. Notification template

The notification template captures the information defined in the following bullet points.

- **BEHAVIOUR** (optional). Describes the behaviour specific to the notification
- **PARAMETERS** (optional). Notification-specific parameters.
- **OBJECT VALUE** (optional). The information that is captured about the event.
- **REGISTERED AS** (mandatory). The type of this notification, could be one of the defined notification types.

2.4.4. Name binding template

The specification that a particular managed object is contained within another managed object and is identified by a particular attribute is called a name binding. Name bindings may also contain other information, such as the rules to be applied when creating and deleting managed objects which may differ depending on their location in the containment tree. Name bindings are defined by the contained managed object class, and always associated to it.

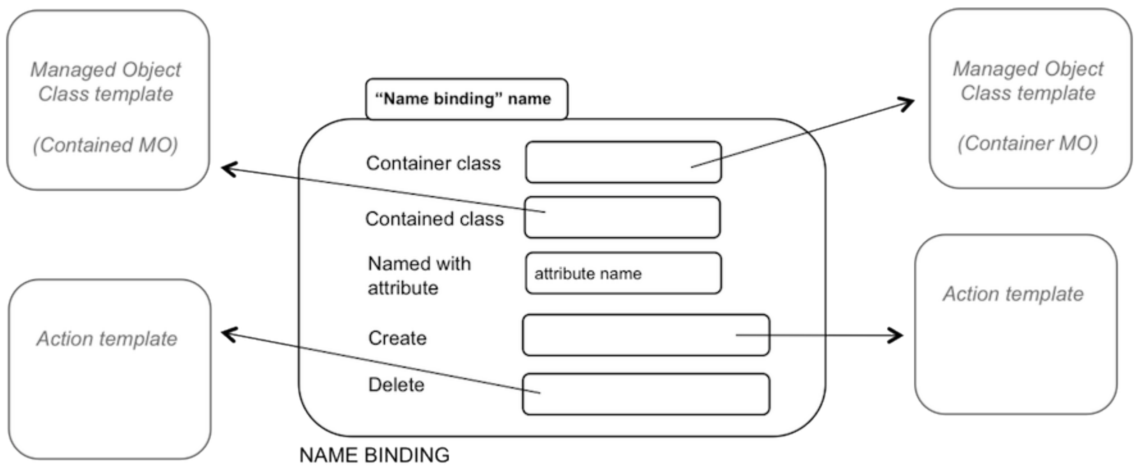


Figure 19. Name binding template

The name binding template needs to capture the information defined in the following bullet points.

- **Name binding name** (mandatory).
- **Container object class** (mandatory). The name of the MO class that will be the container of this managed object. Points to a managed object class template.
- **Contained object class** (mandatory). The name of the MO class that is the child of the "container object" in the containment tree.
- **Named with attribute** (mandatory). The attribute of the contained MO that is used to uniquely identify the contained MO within the container MO (using an Attribute Value Assertion or AVA).
- **Create** (optional). A pointer to an action template defining the specific behaviour for the "Create" operation, capturing any special conditions that apply to the containment relationship (for example, the container MO might define a maximum limit of MOs of class X that can be instantiated).

- **Delete** (optional). A pointer to an action template defining the specific behaviour for the "Delete" operation, capturing any special conditions that apply to the containment relationship (such as if the deletion of the container object should delete all contained objects of class X).

2.5. Example of use of the templates

Examples of the use of these templates can be found in Appendix B, *Managed Object Classes* which defines the RIB Managed Objects (MO).

3. RIB Validation

This section introduces some typical Manager-Agent interactions as a means to verify the validity of the Resource Information Base (RIB) objects specified. The section starts by outlining the validation methodology and then identifies some typical management activities initiated on the manager side and on the agent side under the tasks of configuration, performance and security.

3.1. Validation methodology

This section presents the methodology that is used to both further define the contents of the RIB, and to validate the use of the proposed Managed Object (MO) model. All objects in the RIB are MOs.

The methodology is quite simple:

a. Identify some typical *management activities*
b. Apply those *activities* to the currently defined MO model.

In some cases these activities will identify short-comings in the MO definitions, for example, missing attributes or notifications. In other cases these activities could identify problems in the structure of the RIB itself. For example, registering an application to a DIF that isn't known on that node. This implies the MO is contained in the wrong point of the containment tree, or dependent on the result of another management activity (to create the required containing MOs).

3.2. Identified management activities

This section outlines some typical foreseen management activities. These are broken down into two subsections, one for activities initiated from the Manager (Commands) and one for activities triggered by Agent notifications (Notification of events).

3.2.1. Manager → Management Agent (Commands)

- Configuration
  - Instantiation of a DIF in a computing system
  - Destruction of a DIF in a computing system

- Performance / Security
  - Monitoring of a DIF instance in a computing system

3.2.2. Agent → Manager (Notification of events)

- Configuration
  - Application registered to an IPC Process
  - Application unregistered from an IPC Process
- Performance / Security
  - Performance of an N-1 flow has degraded by a specified threshold.
  - Performance of an N-1 flow has returned to normal again.
  - One or more IPC Processes repeatedly joining and leaving a DIF.

3.3. Activity: Instantiation of a DIF

The DMS Manager process wants to instantiate a DIF in a given computing system. To do so it needs to instantiate an IPC Process and properly configure it (directly assign it to a DIF, tell it to enrol to one or more neighbours, etc). Depending on the decision of the DMS, the IPC Process creation can be divided into four different sub-cases:

1. Instantiation of an IPC Process and assignment to a DIF. The IPC Process is assigned to a DIF just after being instantiated, therefore all the configuration data that the new IPC Process needs to start operating as a member of the DIF needs to be provided as part of the request. Registration of the IPC Process to one or more N-1 DIFs must also happen (this case assumes that the IPC Process waits for enrolment attempts from other IPC Process that may or may not be members of the DIF yet).
2. As stated above in point 1, plus the new IPC Process is instructed to contact one or more neighbours and to initiate the enrolment procedure with them.
3. The IPC Process is instantiated but not assigned to any DIF. Instead it is instructed to initiate the enrolment procedure to join DIF X via one or more neighbours.
4. The IPC Process to be instantiated is a shim IPC Process, therefore information specific to the shim-DIF needs to be provided.

3.3.1. Use case: Instantiation and assignment to a DIF

This operation is a way of bootstrapping a DIF: creating the first IPC Process that is a member of the DIF, however there are also other scenarios where this operation can be of use. As depicted in the Figure below, the action of the Manager process has to cause the instantiation of a new IPC Process (IPCP), (orange IPC Process in the Figure), the registration of this new IPC Process to one or more N-1 DIFs and the assignment of the initialization of the new IPC Process as a member of the orange DIF. In order to perform the last step, the IPC Process needs to be provided all the data required to start operating as a member of the DIF, that is, all the DIF policies and required initialization data.

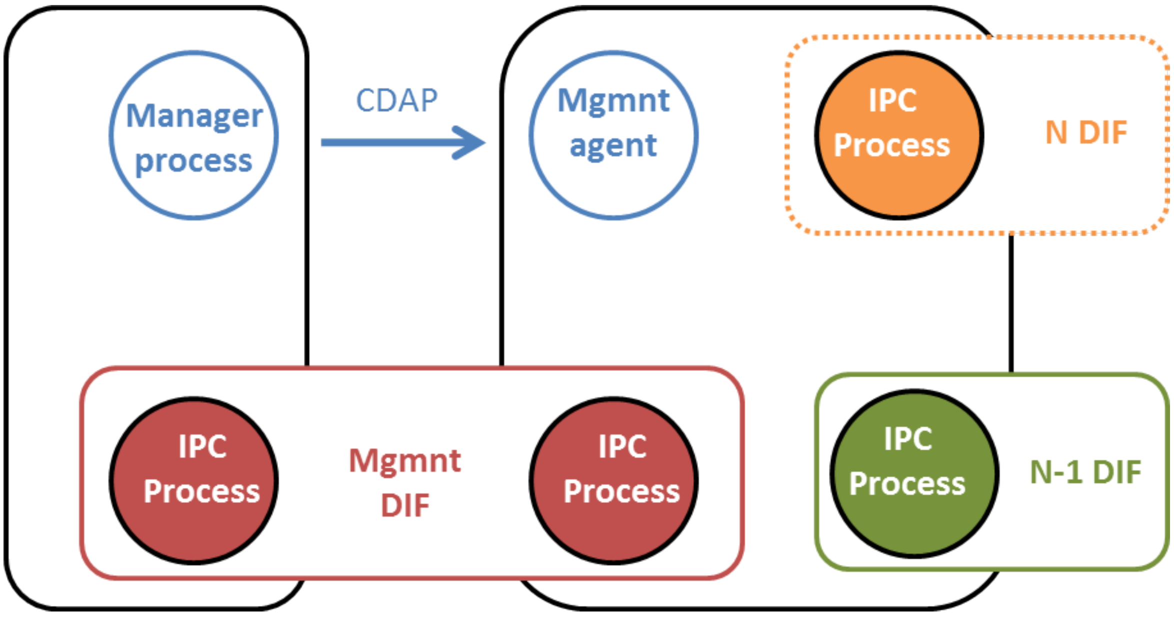


Figure 20. Instantiation of an IPC Process and assignment to a DIF

Information exchanged, behaviour and side effects

The DMS Manager process invokes a remote create operation on the system's RIB, exposed via the Management Agent of the system.

1. Source app: *DMS Manager process*, destination app: *Management Agent of the target system*.
2. Remote operation: *CREATE*.
3. Target object name: *{computingSystemID = 1}, {processingSystemID = 1}, {processID = 1}*
4. Target object class: *IPCProcess*

5. Scope: *0*
6. Object value: IPCP_CONFIG

Behaviour: When the Management Agent receives this message from the Manager process, the following actions will take place:

1. Instantiate an IPC Process, assigning it the name provided by the Manager. This action will cause a new IPC Process object (and all its contained objects) to appear in the RIB containment tree, as a child of the relevant ProcessingSystem object.
2. Register the IPC Process at the DIFs specified by the Manager. Every registration to an N-1 DIF will cause a new DirectoryForwardingTableEntry object to appear in the RIB containment tree, as a child of the DirectoryForwardingTable object of the IPC Process belonging to the N-1 DIF where the new IPC Process has been registered.
3. Assign the IPC Process to the DIF specified by the Manager, initializing it with all the policies particular of that DIF. This action will cause the modification of a number of RIB objects that are contained within the new IPC Process object (in order to reflect the applied configuration), as well as the creation of new objects in the containment tree (such as the ones representing the QoS cubes supported by the new DIF).

In case of errors in any of the steps, all the previous steps will be undone an an error will be returned.

3.3.2. Use case: Triggering of enrolment (IPC Process is a DIF member)

This case is an extension of case 1 and thus, the same operations and configurations are needed. After the instantiation of an IPCP, the registration to one or more N-1 DIFs and the assignment to a N DIF, the triggering of the enrolment is launched. This last step is explained in a more detail in this use case.

Assume an IPC Process has been instantiated in a computing system, registered to one or more N-1 DIFs and assigned to one DIF (Figure 20, "Instantiation of an IPC Process and assignment to a DIF"), the IPC Process is a member of the N DIF and hence, it can be enrolled to any other member of such DIF. For this enrolment to take place, there are three operations needed:

1. The flow allocation between both IPCPs over an N-1 flow.
2. The CACEP between both members.
3. The enrolment between both members (synchronization of shared states).

Once these three operations have been completed, the IPC Process is a member of the DIF and can send messages to its neighbour IPC Process. The following Figure 21, "CACEP and enrolment between two IPC Processes belonging to the same DIF." illustrates this use case.

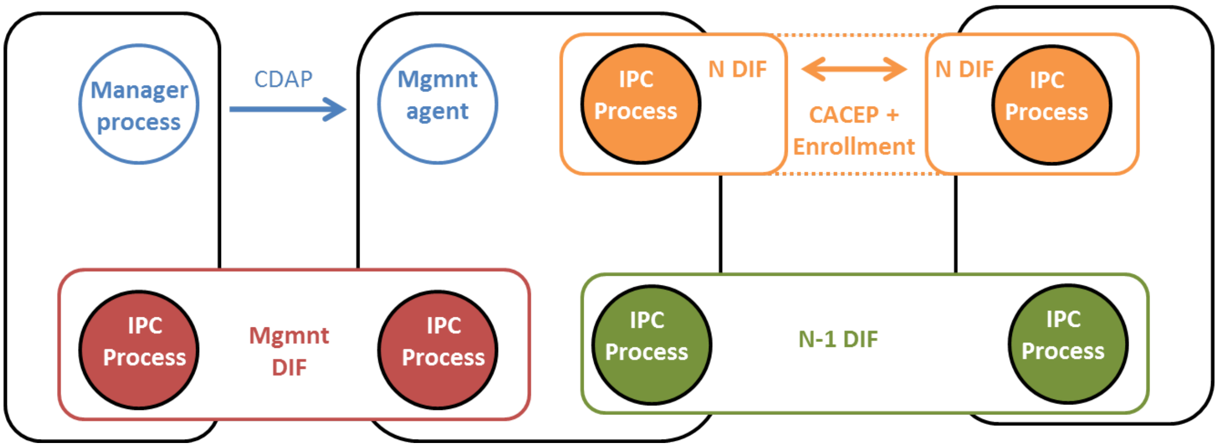


Figure 21. CACEP and enrolment between two IPC Processes belonging to the same DIF.

In order to complete these three operations, some information must be provided to the IPC Process.

Information exchanged, behaviour and side effects

The DMS Manager invokes a remote create on the system's RIB IPC Process, exposed via the Management Agent of the system.

1. Source app: *DMS Manager process*, destination app: *Management Agent of the target system*.
2. Remote operation: *CREATE*.
3. Target object name: *{computing_system_id = 1}, {processing_system_id = 1}, {ipc_process_id = 1}*
4. Target object class: *IPCProcess*
5. Scope: *0*
6. Object value: enrolment_CONFIG

Behaviour: When the Management Agent receives this message from the Manager process, the following actions will take place after the actions described in the previous use case:

1. Establishment of a flow between both IPC Processes using one of the given N-1 DIFs. To be able to complete this operation, both IPC Processes must be registered to at least one of the given N-1 DIFs. If an IPC Process is not registered to that DIF, it will try to register if it is possible, launching an error if not. A FlowAllocatorInstance and an ActiveFlow object are created in both IPC Processes to manage the flow.
2. Start the CACEP by sending the M_Connect Request.
3. Once the CACEP is completed, start the enrolment phase. The enrolment phase exchanges information between both IPC Processes with the objective to synchronize both shared state. Information exchanged includes neighbours and DirectoryForwardingTable.

3.3.3. Use case: Triggering of enrolment (IPC Processis not a DIF member)

This use case is a way of joining an existing DIF. As depicted in Figure 22, "Triggering of enrolment" below, the DMS Manager instantiates an IPC Process (IPCP) who want to join an existing DIF (orange). However, it does not have the configuration of the DIF, but shares at least one N-1 DIF (green) with a neighbour IPC Process. IPC Process (called IPCP$_A$) establishes a flow with IPC Process (called IPCP$_B$) using the N-1 DIF and asks IPCP$_B$ to join the N DIF and to enrol to it. Once these operations are successfully completed, IPCP$_A$ is a member of the N DIF and can send messages to IPCP$_B$.

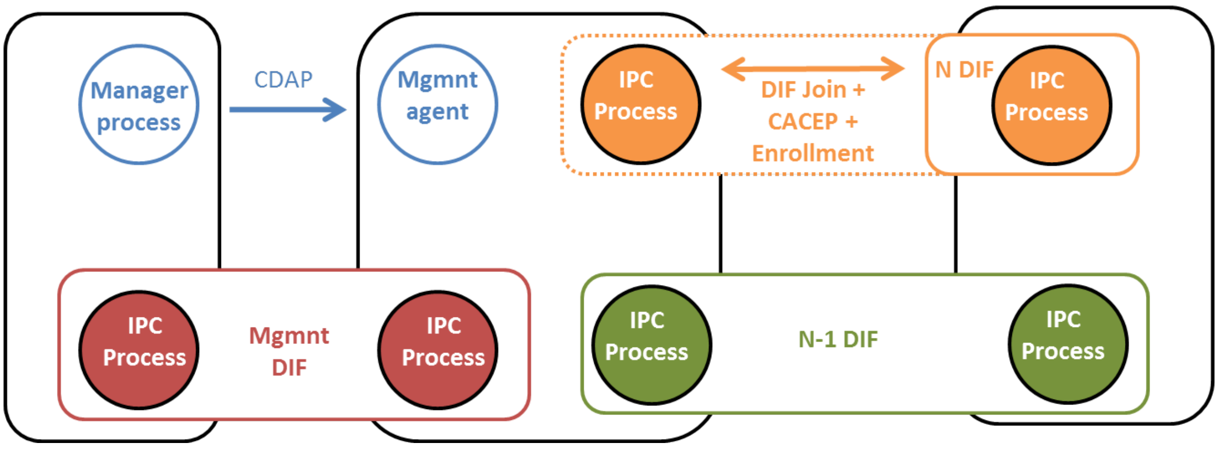


Figure 22. Triggering of enrolment

Information exchanged, behaviour and side effects

The DMS Manager invokes a remote create on the system's RIB IPC Process, exposed via the Management Agent of the system.

1. Source app: *DMS Manager process*, destination app: *Management Agent of the target system*.
2. Remote operation: *CREATE*.
3. Target object name: *{computing_system_id = 1}, {processing_system_id = 1}, {ipc_process_id = 1}*
4. Target object class: *IPCProcess*
5. Scope: *0*
6. Object Value:
    - SEQUENCE basicIPCPInformation
        - *STRING processName*: name of the IPC Process.
        - *STRING processInstance*: instance of the IPC Process.
        - *SEQUENCE_OF STRING synonymList*: list of synonyms of the IPC Process.
        - *STRING neighbor_name*: the name of the neighbour to enrol to.
        - *STRING n_minus_one_dif*: the name of the N-1 DIF to use for the communication between both IPCP.

Behaviour: When the Management Agent receives this message from the Manager process, the following actions will take place after the actions described in the previous use case:

1. Instantiate an IPC Process, assigning it the name provided by the Manager. This action will cause a new IPC Process object (and all its contained objects) to appear in the RIB containment tree, as a child of the relevant ProcessingSystem object.
2. Establishment of a flow between both IPC Processes using one of the given N-1 DIFs. To be able to complete this operation, both IPC Processes must be registered to at least one of the given N-1 DIFs. If an IPC Process is not registered to that DIF, it will try to register if it

is possible, launching an error if not. A FlowAllocatorInstance and an ActiveFlow objects are created in both IPC Process to manage the flow.

3. IPCP$_A$ asks for DIF assignment to IPCP$_B$. This last one checks its AccessControlPolicy to decide if it allows IPCP$_A$ to be a member of the N DIF.
4. If IPCP$_B$ is allowed to join the N DIF, start the CACEP by sending the M_Connect Request.
5. Once the CACEP is completed, start the enrolment phase. The enrolment phase exchanges information between both IPC Processes with the objective to synchronize both shared state.

3.3.4. Use case: Instantiation of a shim IPC Process

This use case correspond to the creation of a shim IPC Process over a selected communication technology (TCP, Ethernet, WIFI). These kinds of IPC Processes are different from the normal IPC Processes (created in RINA DIFs) because a *shim IPCPs* are an interface between a RINA DIF and the selected communication technology. A shim IPC Process is represented in the RIB as a ShimIPCProcess object instead of an IPCProcess object. Moreover, the objects contained in the RIB under a ShimIPCProcess are fundamentally different from the ones under a normal IPCProcess object, as they are mostly technology dependent.

In a shim-IPC Process, enrolment does not exist as it is defined for normal IPC Processes and registration to an N-1 DIF is not needed, since there is no N-1 DIF (a shim DIF is always the lowest level DIF). Hence, the only operation done in a shim IPC Process is the assignment to its corresponding shim DIF.

The figure (Figure 23, “Instantiation of a shim-IPC Process”) below illustrates this use case, where the Manager asks for a shim-IPC Process to be created in the Management agent, who instantiates the requested shim-IPC Process.

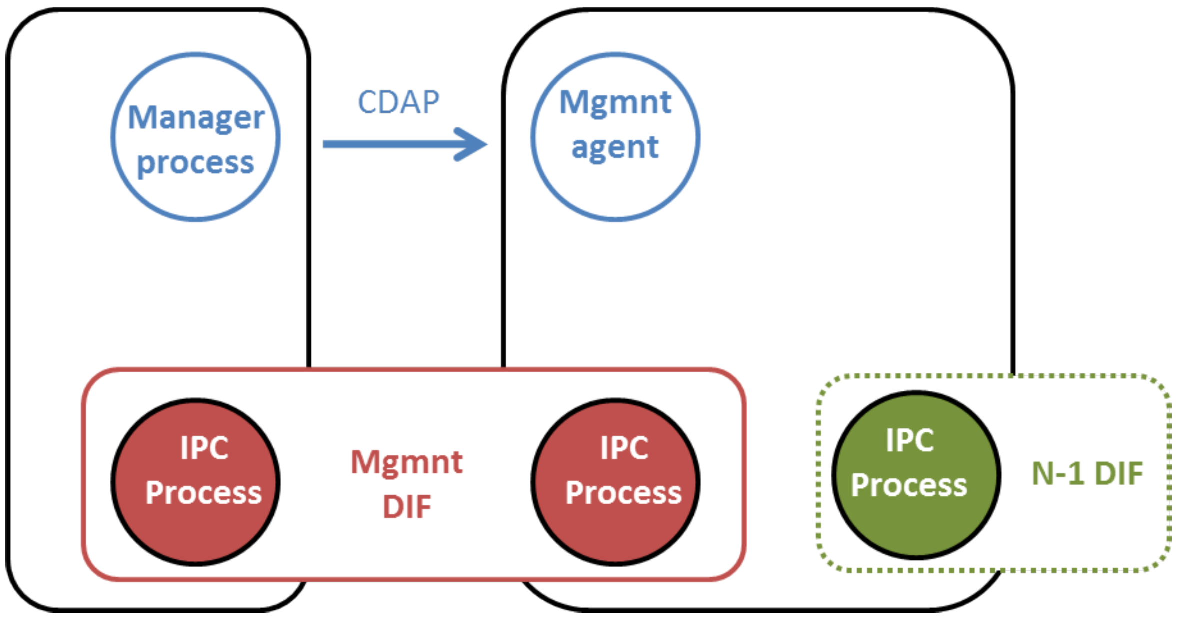


Figure 23. Instantiation of a shim-IPC Process

Information exchanged, behaviour and side effects

The DMS Manager invokes a remote create on the system's RIB IPC Process, exposed via the Management Agent of the system.

1. Source app: *DMS Manager process*, destination app: *Management Agent of the target system*.
2. Remote operation: *CREATE*.
3. Target object name: *{computing_system_id = 1}, {processing_system_id = 1}, {ipc_process_id = 1}*
4. Target object class: *ShimIPCProcess*
5. Scope: *0*
6. Input parameters:
   - SEQUENCE shimInformation
     - *STRING processName*: name of the IPC Process.
     - *STRING processInstance*: instance of the IPC Process.
     - *SEQUENCE_OF STRING synonymList*: list of synonyms of the IPC Process.
     - *STRING type*: type of the shim.
     - *SEQUENCE specificParams*: sequence of specific parameters of that shim.

Behaviour: When the Management Agent receives this message from the Manager process, the following actions will take place after the actions described in the previous use case:

1. Instantiate a Shim IPC Process, assigning it the name provided by the Manager process. This action will cause a new ShimIPCProcess object to appear in the RIB containment tree, as a child of the relevant ProcessingSystem object.
2. Assign the IPC Process to the DIF specified by the Manager, initializing it with all the policies particular of that DIF and creation of the child RIB objects. This action will cause the modification of a number of RIB objects that are contained within the new IPC Process object (in order to reflect the applied configuration), as well as the creation of new objects in the containment tree.

3.4. Activity: Destruction of a DIF

This activity has a number of usecases depending on the context.

3.4.1. Use Case: Soft destruction of a normal IPC

This is the case when an enrolled IPC Process is ordered to be destroyed by the DMS.

In such case the IPC has been successfully enrolled and is now part of a DIF. The IPC Process itself could be currently used by another DIF or Application Processes as a (N-1) DIF and could currently be using an (N-1) DIF or Shim-DIF.

First, the IPC Process is marked as "Deleting" and terminates the flows with the N+1 layers. This cause the upper layers (DIF or Application Processes) to become aware of the change and to react to the event by following their policies.

After such operation is completed, the IPC Process un-subscribes from the DIF, in order to warn its neighbours about its intentions to leave the DIF. The IPC Process then terminates its flow with the N-1 DIFs. At the end of the procedure, after becoming unable to serve (N+1) DIFs and to reach N-1 DIFs (the IPC Process is isolated), the IPC Process terminates itself.

Information exchanged, Behaviour and side effects

The DMS Manager invokes a remote delete on the system's RIB IPC Process, exposed via the Management Agent of the system.

- Source: DMS Manager Process; Destination: Management Agent of the target system.
- Remote operation: DELETE
- Target object name: {computing_system_id = 1}, {processing_system_id = 1}, {ipc_process_id = 1}
- Target object class: IPCProcess
- Scope: 0
- Object Value: BOOLEAN hardDelete = false.

Behaviour

When the MA receives this message from the DMS process, the following actions take place:

- The IPC Process informs its neighbour that it is dropping the subscription to the DIF.
- The MA interacts with the IPC Process and order it to begin the exiting procedures (cause the IPC Process to switch to a "Deleting" state).
- The IPC Process stops any active (N+1) connections (which is also the case of an N+1 IPC Process) by terminating the flows in such direction. This could start some sort of reaction on the upper layers (but this is a matter of policy).
- The IPC Process closes all its opened flows with the (N-1) layers.
- Finally the IPC Process destroy itself, releasing all its resources.

3.4.2. Use Case: Hard destruction of a normal IPC

This is the case where the DMS wants to bring down immediately an IPC Process. Such procedure does not give time to the IPC Process to directly perform any destroy-related operation (such as drop DIF subscription). Its behaviour is the same when an IPC Process crashes due to a critical hardware/software error failure. Thanks to the use of the timers the other members can sense that the IPC Process is no more available and can choose the right strategy to adapt to such a situation.

Information exchanged, Behaviour and side effects

The DMS Manager invokes a remote delete on the system's RIB IPC Process, exposed via the Management Agent of the system.

- Source: DMS Manager Process; Destination: Management Agent of the target system.
- Remote operation: DELETE
- Target object name: {computing_system_id = 1}, {processing_system_id = 1}, {ipc_process_id = 1}
- Target object class: IPCProcess
- Scope: 0
- Object Value: BOOLEAN hardDelete = true.

Behaviour

When the DMS begins with such an operation, the following actions take place:

- On the processing system where the IPC Process to be destroyed is located, the MA simply kills the process without giving it the opportunity to start the operations for a soft destruction.
- The RIB is destroyed.
- The DMS forwards the order to the DIF MAs, so that the information of the IPC Process destruction can be processed by the other processing systems. This is done to maintain a valid state of the information base. Such operations cause the IPC Process to be considered out-of-the-DIF from the other DIF members.
- Application Processes or IPC Processes which were communicating with the killed IPC Process will notice its unavailability when the communication timers expire.

3.4.3. Use Case: Destruction of the whole DIF

The destruction of the DIF occurs on all the members of the DIF itself. Once the DMS has selected the participants of a certain DIF, it just proceed sending IPC Process Destruction orders (as seen in the previous examples) one after another. The last step of this process is to purge the old information that the DMS has in it's own RIB, so that the DIF does not exist anymore and any old information of the IPC Processes, which were members of the DIF, are cleaned.

Information exchanged, Behaviour and side effects

The information exchanged will depend on what sort of method is chosen for the destruction of the IPC Processes. The exchanged information has been documented in the previous use cases (hard or soft) so not repeated here.

Behaviour

The DMS will perform the following actions in order to destroy the whole DIF:

- Query the RIB to extract information on the DIF to be deleted (i.e. the list of participating IPC Processes).
- Iteratively destroy all IPC Processes in the DIF. Once the DIF has no more IPC Processes, then the remaining information of the DIF are purged from the RIB cleaning up all the resources.

3.4.4. Use Case: ShimIPC soft destruction

In order to destroy this element the process must first close the active flows in order to become unreachable from other IPC Processes of the DIF. This avoids conflicts where messages arrive for this IPC but nobody is there to listen for them. After closing all its incoming/outgoing communication flows then the IPC can destroy itself and releasing all the resources to the processing system.

Information exchanged, Behaviour and side effects

The DMS Manager invokes a remote delete on the system's RIB ShimIPCProcess, exposed via the MA of the system.

- Source: DMS Manager Process; Destination: Management Agent of the target system.
- Remote operation: DELETE

- Target object name: {computing_system_id = 1}, {processing_system_id = 1}, {ipc_process_id = 1}
- Target object class: ShimIPCProcess
- Scope: 0
- Object Value: BOOLEAN hardDelete = false.

Behaviour

When the Management Agent receives this message from the Manager process, the following actions take place:

- The MA interacts with the shim IPC Process and orders it to begin the termination procedures.
- The shim IPC Process closes its currently active flows.
- The shim IPC Process performs the necessary technology dependent operations to terminate the communication (e.g in the case of a TCP/IP shim it will close the currently opened sockets).
- Finally the shim IPC Process auto-terminates itself.

3.5. Activity: Monitoring of a DIF

For the IPC Monitoring activity, different approaches can be followed. This section presents the monitoring strategies that can be followed by the Manager and Management Agent which can be configured by means of strategies. The centralized manager may be located either in one of the processing systems or in a standalone processing system.

Two main monitoring approaches can be followed:

- Centralized monitoring. The Manager is the entity in charge of the data collected from monitoring, and therefore, it's also the entity in charge of the management actions (i.e. the actions to be performed upon a certain event).
- Distributed monitoring. The Management Agent is the one in charge of storing the data collected from monitoring. In this case the management actions may be performed by the Manager or the Management Agent. In the former case, the Management Agent shall send event notifications to the Manager, in the latter the Manager shall configure the management actions to be carried out by the Management Agent. This possibilities

will be explored in the "threshold break" and "threshold restore" use cases.

As for the kind of monitoring, different patterns are possible:

- Reactive monitoring: The Management Agent reacts to polling messages from the Manager, i.e. the manager asks for a certain parameter and the Management Agent sends its value back.
- Proactive monitoring: The Management Agent pro-actively sends monitoring messages to the Manager. It can be of two kinds:
  - Periodic Monitoring: Proactive monitoring messages are sent periodically with a period T.
  - Event Monitoring: Proactive monitoring messages are sent when a certain event occurs (launched by the Management Agent).

These monitoring patterns are further explained below. However, it is also interesting to study different association approaches of the Manager and Management Agent. Although they are implicit in the monitoring approaches described above, these monitoring associations are to be further studied within the RINA context.

- Hierarchical monitoring: Management Agents collaborate in groups sending monitoring messages to a hierarchically superior entity. The case in which only one layer is present, with the Manager on top and the Management Agents below (all of them in the same level) can be understood as hierarchical monitoring. In case more layers are present, they will be composed of Management Agents aggregating a set of "child" Management Agents, with the Manager being the entity at the top of the hierarchy. This approach allows to optimize the monitoring process and reduce the overhead introduced in the network.
- Federated monitoring: This approach do not have a monitoring hierarchy. Instead, functionality is delegated from a central entity to the federated entities, or the central entity performs a certain function on behalf of the federated entities. In the hierarchical approach the upper layers aggregate lower layers, being an aggregation point for monitoring messages or being the commanders of the functionality of the layers below. In the federated approach, the central layers are not aggregation points. Rather, they perform some functionality for the lower layers or "delegate" some functionality to the lower layers.

In the following the above concepts are explained in more detail. Additionally, policies are defined to configure each kind of monitoring.

3.5.1. Reactive monitoring

The Manager polls a certain set of Management Agents querying specific measurement values and they respond with the values. The time granularity over which the parameter values are polled is up to the manager.

Policies

- Policy name: Reactive Monitoring policy
  - Component: Manager.
  - Description: This policy defines the reactive monitoring carried out by the Manager.
  - Parameters:
    - For each parameter p and for each node n, a reactive monitoring period T is associated.
    - Set of events that trigger a monitoring period change (e.g. when a node failure is detected) and the set of enquires that is initiated (the nodes that are connected through the same path links are polled to check that the links are not broken).
  - Default action: none.

3.5.2. Proactive monitoring

The Management Agents pro-actively send to the Manager monitoring information without any previous request from the Manager. Two different approaches are followed:

- Periodic: The MA sends monitoring messages to the CM periodically. The period (T_p) can be dynamically adjusted (by the Manager or locally by the Management Agent) for different parameter settings and different monitoring granularities.
- Event driven: The MA sends monitoring messages to the CM according to triggering events. A simple criterion for an event driven strategy is to consider the difference (Δ) of the current sample s_i in time i and

a sample of the last event s_E according to a certain threshold (α). The thresholds may be set by the Manager or locally by the Management Agents. Formally:
$\Delta = |s_i - s_E|$
$\Delta \geq \alpha \rightarrow$ event

Periodic Proactive Monitoring Policies

- Policy name: Proactive Monitoring policy
  - Component: Management Agent.
  - Description: This policy defines the proactive monitoring strategy followed by a Management Agent.
  - Parameters:
    - For each parameter p, a monitoring period T_p is associated.
  - Default action: none.

Periodic Proactive Monitoring Policies

- Policy name: Event Monitoring policy
  - Component: Management Agent.
  - Description: This policy defines the proactive monitoring strategy followed by a Management Agent.
  - Parameters:
    - For each event e detected in the Management Agent, a set of parameters to be monitored is associated.
    - Occurrence of a set of events to be monitored. E.g. for each parameter p, a threshold α and a sampling period T_{sp} can be associated. The events would be produced as described above.
  - Default action: none.

3.6. Activity: Performance threshold exceeded

Performance evaluation has to be carried out by means of quantitative parameters that specify the system status at a certain point. To determine the correct/acceptable system operation, these parameters must remain within a certain range delimited by thresholds. In this way, performance management actions can be triggered when the parameter values surpass the given thresholds.

We can divide the performance management operation into different phases:

1. Performance management strategy. The manager decides what performance policies apply, and communicates to the management agent the necessary information to set up its performance management configuration. This involves for example the definition of the specific policies to apply in the management agent, specification of the parameters involved and the setting of specific threshold values.
2. Performance management events. Once the performance management policies are configured, performance management events will be triggered upon operation time. These can be of two kinds:
    a. Events generated in the management agent: based on the parameters' measurements, the management agent detects the triggering condition.
    b. Events generated in the manager: based on the monitored data from the management agent, the manager detects the triggering condition and carries out the appropriate actions.
3. Performance management actions. Upon performance management events triggered in the previous phase, actions may be carried out to manage the situation. Three cases are possible:
    a. Management actions carried out by the manager. If the triggering condition was detected in the management agent, a notification is needed.
    b. Management actions carried out by the management agent. If the triggering condition was detected in the manager, a notification is needed.
    c. A combination of the two.

In the following we analyse the most likely possibilities for the use case. We consider the following possibilities for the points described above.

- Threshold break events triggering. We consider that this can be carried out in both the manager or the management agent. It depends on the monitoring strategy.

- Management actions decision. We consider that only the manager decides the actions to perform upon a certain threshold break event. The manager is the one that has an overall view of the system and can take decision based on optimization goals.
- Management actions execution. We consider that the management agent is the one that executes the management actions decided by the manager. The management agent is located at the network nodes, so it's the one that has easier access to network resources and carry out changes and updates.

3.6.1. Use case 1: Threshold break events triggered in the management agent

This is the most common use case. It corresponds to a proactive performance monitoring strategy where the notifications convey performance management events rather than plain parameter values. The use case is composed of the following parts:

1. Instantiation of a threshold break trigger
2. Performance management event delivery
3. Performance management actions

Part 1: Instantiation

The manager decides what performance management policies to apply, and then it commands the management agent to configure according to the policies.

Source app: *Manager process*, destination app: *Management Agent of the target system*

- *Operation*: CREATE
- *invokeID*: Automatic
- *scope*: 0
- *opCode*: Automatic
- *objClass*: ForwardingDiscriminator
- *objInst*: 1

- *objName*: {ProcessingSystem = 44, Discriminator}
- *objValue*: DISCRIMINATOR_POLICY_CONFIG
  - STRING destinationName: <manager name>
  - UNSIGNED INTEGER destinationInstance: <manager Instance>
  - SET_OF NOTIFICATION notifications:
    - STRING object: {Fully Qualified Name of the object}
    - STRING notification: thresholdBreak Notification.
  - POLICY_CONFIG filteringPolicy:
    - STRING name: 1-to-1
    - STRING version: 1
    - SET_OF POLICY_CONFIG_PARAMETER parameters:
      - STRING name: Threshold_up, maximum of the operational range of the parameter.
      - STRING value: <value>
      - STRING name: Threshold_down, minimum of the operational range of the parameter.
      - STRING value: <value>
      - STRING name: Period, Time lapse to check the parameter value periodically.
      - STRING value: <value>

Reply with RESULT = 0 expected.

Part 2: Delivery

When the management agent detects a triggering condition, the event is notified to the manager.

Source app: *Management Agent of the target system*, destination app: *Manager process*

- *Operation*: WRITE with reply requested
- *invokeID*: Automatic
- *objClass*: DiscriminatorReport

- *objInst*: 1
- *objName*: {<Manager RIB path>}
- *objValue*: SEQUENCE notificationParameters:
  - STRING fullyQualifiedName : fully qualified name of the report
  - STRING attributeExceeded: name of the attribute that exceed the threshold
  - STRING thresholdExceeded: name of the threshold exceeded
  - STRING attributeValue: value of the attribute
  - STRING thresholdValue: value of the threshold exceeded.
- *opCode*: Automatic

Reply WRITE_R with RESULT = 0 expected.

Part 3: Actions

To carry out the performance management actions, the manager may request the management agent to carry out some specific tasks.

Source app: DMS Manager process, destination app: Management Agent of the target system.

- *Operation*: CREATE/WRITE.
- *invokeID*: Automatic
- *objClass*: <targetted object class>
- *objInst*: <targetted object instance>
- *objName*: {<Fully Qualified Name targetted object>}
- *objValue*: SEQUENCE actionParameters
- *opCode*: Automatic

3.6.2. Use case 2: Threshold break events triggered in the manager

In this case the management agent forwards the information collected by monitoring to the manager. The manager checks whether the monitored data remains within the established thresholds, and upon a threshold break, the manager requests to the management agent to perform a certain performance management routine. (This is equivalent to the Part 3 of the previous use case).

3.7. Activity: Performance threshold restored

This use case is closely related to the threshold break use case. The threshold break use case applies when the performance, measured by means of parameter values, steps out of the operational/acceptable value ranges. On the contrary, this use case applies when the performance has come back within the operational/acceptable ranges again.

As in the threshold break use case, in order to carry out the performance evaluation and management to detect and handle the case in which the system has recovered the expected performance, the manager must have a management agent registered, so that they can communicate and notifications can be exchanged between them.

We can divide the performance management operation for this use case in different phases, equivalent to the ones in the threshold break use case:

1. Performance management strategy. The initial performance strategies from the manager must include the normal operational ranges.
2. Event trigger. Performance management events will be triggered at operation time indicating the performance recovery. These can be of two kinds:
   a. Events generated in the management agent: based on the parameter measurements, the management agent detects the triggering condition.
   b. Events generated in the manager: based on the monitoring data from the management agent, the manager detects the triggering condition.
3. Management actions may be carried out to manage the situation. Three cases are possible:
   a. Management actions carried out by the manager.
   b. Management actions carried out by the management agent.
   c. A combination of the two. Basically, the possible strategies for the threshold break use case and this use case are the same. However, the

manager may consider applying different strategies in both cases. For example, for the threshold break use case, the manager may detect the threshold step out by itself (based on monitoring values as described in the threshold break use case), but for detecting the performance recovery, the manager may want the management agent to detect the re-establishment of the parameter values within the operational ranges.

Summing up, the operations of both use cases are equivalent, but different configurations may apply for each one. In this sense, we can study the same possibilities for each of them, being the manager the entity that controls the operation of each of them.

In the following we describe the different possibilities. We refer to the re-establishment of the performance parameter values as “performance recovery”.

3.7.1. Use case 1: Triggered in the management agent

Performance recovery events are triggered in the management agent, management actions are decided in the manager and carried out in the management agent. This corresponds to a proactive performance monitoring strategy where the notifications convey performance management events rather than plain parameter values. The use case is composed of the following parts:

Part 1: Instantiation of a trigger

The manager decides what performance recovery policies to apply, and then it commands the management agent to configure according to the policies.

Warning: This part applies in the case that specific actions shall be instantiated upon a threshold break. Otherwise the trigger specifications for the performance recovery are the same as in the threshold break use case.

Source app: *Manager process*, destination app: *Management Agent of the target system*

- *Operation*: CREATE

- *invokeID*: Automatic
- *scope*: 0
- *opCode*: Automatic
- *objClass*: ForwardingDiscriminator
- *objInst*: 1
- *objName*: {ProcessingSystem = 44, Discriminator}
- *objValue*: DISCRIMINATOR_POLICY_CONFIG
  - STRING destinationName: <manager name>
  - UNSIGNED INTEGER destinationInstance: <manager Instance>
  - SET_OF NOTIFICATION notifications:
    - STRING object: {Fully Qualified Name of the object}
    - STRING notification: thresholdBreak Notification.
  - POLICY_CONFIG filteringPolicy:
    - STRING name: 1-to-1
    - STRING version: 1
    - SET_OF POLICY_CONFIG_PARAMETER parameters:
      - STRING name: Threshold_up, maximum of the operational range of the parameter.
      - STRING value: <value>
      - STRING name: Threshold_down, minimum of the operational range of the parameter.
      - STRING value: <value>
      - STRING name: Period, Time lapse to check the parameter value periodically.
      - STRING value: <value>

A create reply with RESULT = 0 expected.

Part 2: Event delivery

When the management agent detects a triggering condition, the event is notified to the manager.

Source app: *Management Agent of the target system*, destination app: *Manager process*

- *Operation*: WRITE with reply requested
- *invokeID*: Automatic
- *objClass*: DiscriminatorReport
- *objInst*: 1
- *objName*: {<Manager RIB path>}
- *objValue*: SEQUENCE newEnrolmentAttemptParameters :
    - STRING fullyQualifiedName : fully qualified name of the report
    - STRING attributeExceeded: name of the attribute that exceed the threshold
    - STRING thresholdExceeded: name of the threshold exceeded
    - STRING attributeValue: value of the attribute
    - STRING thresholdValue: value of the threshold exceeded.
- *opCode*: Automatic

Reply WRITE_R with RESULT = 0 expected.

Part 3: Performance management actions

To carry out the performance management actions, the manager may request the management agent to carry out some specific tasks.

Source app: DMS Manager process, destination app: Management Agent of the target system.

- *Operation*: CREATE/WRITE.
- *invokeID*: Automatic
- *objClass*: <targetted object class>
- *objInst*: <targetted object instance>
- *objName*: {<Fully Qualified Name targetted object>}
- *objValue*: SEQUENCE actionParameters
- *opCode*: Automatic

3.7.2. Use case 2: Triggered in the manager

The threshold restoration events are triggered in the manager based on monitoring data from the management agent, management actions decided in the manager and carried out in the management agent.

In this case the management agent monitors data to the manager following the process described in Section 3.5, “Activity: Monitoring of a DIF”. The manager checks whether the monitored data has come back within the established thresholds, and upon performance recovery, the manager requests to the management agent to perform certain performance recovery management routine. (This is equivalent to the Part 3 of the previous case).

3.8. Activity: Security monitoring

In D4.1 [D41] we identified the following seven threats that should be monitored in a DIF to detect runtime attacks:

T1	An IPC Process provides false information to other IPCPs
T2	An IPC Process is deliberately overwhelming other DIF members or the underlying DIF with messages
T3	An IPC Process is not forwarding messages
T4	One or more IPC Processes are repeatedly joining and leaving a DIF
T5	An IPC Process is repeatedly causing errors when attempting to join a DIF
T6	A compromise of the data stored in an IPC Process or DIF
T7	A compromise of an IPC Process or DIF function so that it malfunctions

We therefore need to ensure that the specifications for the MO model in D5.1 [D51] allow these threats to be monitored. Here we take one of the threats listed above as an example to validate the MOM definition and to further define the behaviour and structure of the RIB objects.

We consider T4 as an example. In this threat one or more IPC Processes repeatedly join and leave a DIF, consuming management resources to achieve a Denial of Service attack. To detect this attack, the MA could monitor the number of attempts made by IPCPs to join a DIF. Since T4 requires an IPC to successfully join the DIF, this could be detected by the MA, counting the number of IPCPs that fully enrol in the DIF (i.e. complete CACEP, authentication and enrolment successfully) via the IPC Process it is monitoring within the specified time period. Here, we consider the MA that monitors this parameter and the messages would be exchanged between the Manager and the MA. We have not yet considered how to get information to the MA from the IPC Process components about enrolment requests.

The first step is for the Manager to configure the MA to monitor the number of successful IPC Process attempts to join a DIF. It does this by requesting the MA to create a forwarding discriminator that will notify the manager when for example, 5 successful IPC Process attempts are made to enrol in the specified DIF. The discriminator then filters the notifications from the IPC Process. When it has received 5 notifications of successful IPC Process attempts to enrol in the DIF, the MA sends a report to the Manager. It is the Manager's role to decide how to deal with any excessive attempts to join the DIF.

Below are examples of the messages exchanged between the Manager and the MA.

3.8.1. Subscribing to enrolment notifications

The process can be summarised as, the manager creating a subscription, the agent confirming the subscription and the agent reporting errant behaviour.

Request: Manager creates the subscription

Source app: *Manager process*, destination app: *Management Agent of the target system*

- *Operation*: CREATE
- *invokeID*: Automatic
- *scope*: 0
- *opCode*: Automatic
- *objClass*: ForwardingDiscriminator

- *objInst*: 1
- *objName*: {ProcessingSystem = 44, Discriminator}
- *objValue*: DISCRIMINATOR_POLICY_CONFIG
    - STRING destinationName: <manager name>
    - UNSIGNED INTEGER destinationInstance: <manager Instance>
    - SET_OF NOTIFICATION notifications:
        - STRING object: {root, computingSystem =1, ProcessingSystem = 44, kernelApplicationProcess, OSAplicationProcess, processName = <name>, difManagement, enrolment, neigbors}
        - STRING notification: IncrementNumberOFEnrolmentAttempts.
    - POLICY_CONFIG filteringPolicy:
        - STRING name: 5-to-1 and successful enrolment attempt
        - STRING version: 1
        - SET_OF POLICY_CONFIG_PARAMETER parameters:
            - STRING name: numberOfNotifications
            - STRING value: 5
            - STRING name: isEnrolled
            - STRING value: true

Response: MA confirms subscription

Source app: *Management Agent of the target system*, destination app: *Manager process*

- *Operation*: CREATE_R
- *invokeID*: Automatic
- *scope*: 0
- *opCode*: Automatic
- *objClass*: ForwardingDiscriminator
- *objInst*: 1
- *objValue*: null
- *result*: 0

3.8.2. MA sends a report to the manager

Source app: *Management Agent of the target system*, destination app: *Manager process*

- *Operation*: WRITE with reply requested
- *invokeID*: Automatic
- *objClass*: DiscriminatorReport
- *objInst*: 1
- *objName*: {<Manager RIB path>}
- *objValue*: SEQUENCE newEnrolmentAttemptParameters :
    - STRING fullyQualifiedName : fully qualified name of the report.
    - STRING processName : <newEnrollingProcessName>
    - UNSIGNED INTEGER processInstance: <newEnrollingProcessInstance>
    - SET_OF UNSIGNED INTEGERS underlayingFlows : {<supportingFlow 1>, <supportingFlow 2>}.
    - DATE dateOfAttempt : <dateOfEnrolmentAttempt>
    - TIME timeOfAttempt : <timeOfEnrolmentAttempt>
    - BOOL isEnrolled : <enrolmentAttemptSuccess>
- *opCode*: Automatic

4. RIB library design

The Common Distributed Application Protocol (CDAP) is the protocol used by all IPCPs to communicate with each other. This means that any exchange of state between two IPCPs - exchanged over an N-1 flow - is sent via the CDAP protocol. CDAP is also the recommended protocol for the exchange of state within any DAF in general, and it is the protocol used by the management applications within the DMS Distributed Application Facility (DAF) [1].

The Resource Information Base (RIB) is the part of the internal state of an AP that is exposed to remote APs. This state is represented using an Object Oriented approach, where the state is a group of objects, from certain classes (classNames) with a certain inheritance relation, and with containment relations, in the form of a tree or a forest of trees. The interaction between APs takes place when they perform operations to the RIB of remote APs, via the RIB Daemon.

A RIB library, generally speaking, is a software library that aims at helping application developers build the different RIBs (and their RIB daemons, as defined in the reference model) and simplifying the interaction with the module in charge of processing the CDAP messages coming from the N-1 flow. The summary of these functions is:

a. Creation of RIB and RIB daemon instances;

b. Definition of the classes (type of objects) as well as the behaviour(s) associated to them;

c. Offer an API to allow manipulate the objects locally (initial population of the RIB, creation, destruction and modification at runtime);

d. Handle remote operations coming from the CDAP messages parsed and validated by the CDAP library.

The current IRATI stack, and specifically *librina*, contains an implementation of a RIB library which was developed mainly driven by the FP7-IRATI project's requirements - i.e. focusing only on the IPC Process RIB requirements. In order to maximize the reuse of code across

[1] A DAF that contains a Manager, and at least one management agent

applications that use RINA, as well as the IPC Process implementation, there is the need to define a common library that implements the common CDAP protocol handling and RIB management functionality. Therefore, the goal of this section is to discuss the implementation of the RIB library software component since its functionalities - within the PRISTINE project - have to be provided to any Application Process (AP) in general, not only the IPC Processes.

4.1. The LLCR library

The **L**ow **L**evel **C**DAP **R**IB (LLCR) library is an ongoing design effort for a generic RIB and CDAP library. The aim of this design exercise is to analyse the needs of both IPC Processes and Applicaiton Procseses in general, determining the common parts and mechanisms towards defining a clear set of APIs and finally providing the functionalities to Application Processes as well as IPC Processes.

The practical benefits of this approach will be the maximization of code reuse, an opportunity for improving the overall performances as well as ease the integration with existing applications. In addition, some design aspects of the LLCR library could improve compatibility among programming languages and hardware platforms of the final RIB library.

The outcome of these efforts is to converge the current implementation of the RIB classes from the IRATI project with the final *llcrlib* design, in order to have an improved RIB library.

4.1.1. Software architecture

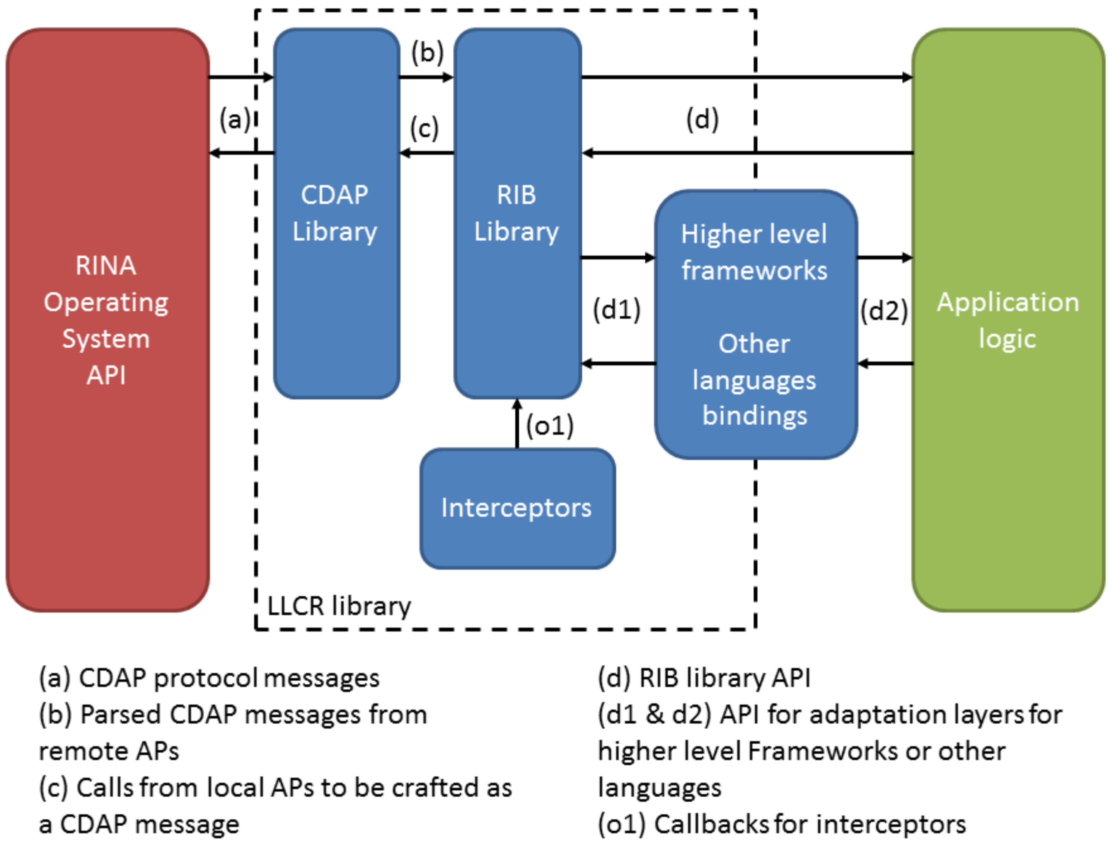

RINA Operating System API
CDAP Library
RIB Library
Higher level frameworks
Other languages bindings
Interceptors
Application logic
LLCR library
(a)
(b)
(c)
(d)
(d1)
(d2)
(o1)
(a) CDAP protocol messages
(b) Parsed CDAP messages from remote APs
(c) Calls from local APs to be crafted as a CDAP message
(d) RIB library API
(d1 & d2) API for adaptation layers for higher level Frameworks or other languages
(o1) Callbacks for interceptors

Figure 24. Components of the CDAP RIB library

CDAP library
: Provides basic CDAP encoding/decoding of messages. The library needs a RIB provider.

RIB library
: Provides an API to manage RIBs. The library needs a CDAP provider.

Interceptors (optional)
: These modules can intercept incoming remote operations before those are being passed to the application. Examples: ACLs, or debug loggers

Higher level frameworks & bindings to other languages
: Used to simplify the application logic

The two principal modules are the CDAP library and the RIB library. The benefit of having such a modular design is that both libraries can work on their own, meaning that they could be developed independently, adapting to the needs of the project in every stage.

4.1.2. Features of the CDAP llcrlib sub-library

The principal feature of the CDAP library is to provide crafting and parsing of CDAP messages and connection establishment (with CACEP). From the point of view of the RIB library, the use of the CDAP library hides all the complexity of the CDAP protocol so it can deal directly with the objects of the remote RIB it might have access to. A more detailed feature list is given below:

Schema definition
A schema is the set of rules of containment that can be applied to a RIB. Examples of this rules are a maximum number of instances for a specific type of object or forbid the containment of a specific type inside of another one.

Definition of classes
The RIB library should be able to define classes (types) of objects. The objects of a class share the same behaviour(s) with regards to the six CDAP operations, and hold the same value. Classes also define the contention relations, as well as the inheritance, including multiple inheritance. The library should provide built-in classes for basic types, those for simple values, with and without synchronization routines (locking), so that can be easily used by the application developers.

Support for multiple RIB instances
Supporting multiple instances of a RIB lets the application support multiple versions at the same time. It is important to note that all the RIB instances work with the same application's internal state. Each version offers a different view over the internal state of the application (the RIB is only a facade).

RIB validation
The RIB library should be able to validate the creation and destruction of objects in the RIB, including the position in the tree and the object operations, following the schema previously defined by the user.

Scoped Operations
The scope modifier of an operation allows operations to be applied over more than one object in the tree and should be supported by the library.

Interceptors

The RIB library should be able to define callbacks that let other entities - e.g. Managers or other APs - react on the reception of an operation just before calling the user's callback.

5. Management Agent design

This section describes the work-flows and the high level architecture of the Management Agent (MA). Specifically, the following section outlines the various work-flows in the life-cycle of the MA that include instantiation and bootstrapping of the MA, processing a request from the manager process and processing a change in the state of a managed IPC process. Then a high level overview of the management agent's software architecture based on the FP7 IRATI stack is provided. Finally, some of the main software components of the management agent that manage the communication between both the processing system's IPC Process (IPCP) and IPC Manager (IPCM) processes and the MA itself are described. This includes a monitoring task responsible for reacting to specific IPCM, IPCP and Operation System (OS) events, a notification component responsible for generating reports and core logic that maintains the business logic of the MA.

5.1. Workflows

The goal of this section is to describe the different workflows in the lifecycle of the MA in order to identify i) its major functionalities and group them into (software) components; ii) the interactions of the MA with the other actors in the software architecture - i.e. librina, IPCM and IPCP daemons at both kernel and user spaces.

The major workflows identified are described in the following sections.

5.1.1. Instantiation and bootstrapping of the MA

The MA will be instantiated at system startup along with the IPC Manager Daemon. Since the MA relies on the IPC Manager functionalities, its bootstrapping process can be described as follows:

1. Wait for IPC Manager to complete its bootstrapping phase.
2. Gather all the required information to do the initial population of the RIB(s).
3. Initialize other MA internal subsystems.
4. Setup the DIFs and minimal shim-DIFs required by the DMS.
5. Enrol to the DMS DAF.

Steps 4 and 5 are of particular interest due to their complexity and thus require a more detailed explanation. In order to establish the initial connection to the Manager, the MA needs to establish an N-1 Flow to the Manager over an existing DMS-DIF where the Manager is already registered. Therefore, the MA needs to get the necessary configuration a priori [2] , as well as the necessary Manager information (Application Name and - optionally - Application Instance Id, Application Entity and credentials). This configuration and information must be available for the MA locally during the bootstrapping of the MA (e.g. via a configuration file, local database or manually configured by the user via a Command Line Interface (CLI)).

Once the MA has been instantiated and the required DIFs configured, the MA is ready to enrol to the DMS-DAF. To do so, it will allocate a flow to the Manager process, establish an application connection, negotiating CDAP and RIB versions and optionally authenticating. Note that the DIF supporting the DMS-DAF might not be trusted and thus the MA could decide to protect the data it exchanges with the Manager by encrypting it.

Requirements:

- **JOIN-1**: The IPC Manager Daemon, at the behest of MA, needs to create all the IPC Processes required to allow the MA to join the DMS-DAF (therefore it needs to know a minimal configuration of these IPC Processes).
- **JOIN-2**: The MA needs to know the name of the Manager process.
- **JOIN-3**: The MA needs to be able to establish application connections with Manager AP(s), optionally authenticating (*CACEP component with authentication policy*). For a successful authentication the MA needs to have the proper credentials prior to contacting the Manager.
- **JOIN-4**: The MA needs to be able to exchange CDAP messages with the Manager (*CDAP library component*), who will operate remotely on the MA RIB (*RIB and RIB Daemon components*).
- **JOIN-5**: The MA needs to be able to successfully carry out the actions defined in the DMS-DAF Enrolment specification (*Enrolment Task component*).

[2] For example, the DMS DIF configuration and the minimal shim-DIF(s) the DMS-DIF requires.

- **JOIN-6**: The MA may need to be able to encrypt the SDUs it sends through an N-1 flow (*SDU Protection component*).

5.1.2. Processing a request from the Manager Process

The Manager will communicate its requests to the MA by operating on the MA RIB via the CDAP protocol. Once a CDAP Message arrives at the MA, it will be interpreted by its RIB Daemon, which will eventually invoke the operation over the object(s) targeted by the CDAP message.

The RIB of the MA is composed, among other information, of the RIBs of the IPC Processes (IPC Processes RIBs appear as sub-trees or branches in the MA RIB). Implementations of the MA can take advantage of this fact by delegating operations which target a RIB object in a sub-tree belonging to an IPCPs to the relevant IPC Process RIB itself, therefore the MA does not need to have repeated RIB objects to represent those in the IPC Processsub-tree. To do so, it could either pass the different parameters of the CDAP message (object name, object class, invoke-id, object-id, object-value, attribute-ids, scope, etc) to the IPC Process Daemon, or directly pass the encoded CDAP message which already contains all the information (the RIB Daemon of the IPC Process would decode it using its instance of the CDAP library). Upon receiving the request from the MA, the IPC Process Daemon would then invoke the operation on the RIB. Finally the IPC Process Daemon would reply to the MA, who in turn would create a CDAP response message and send it to the Manager. The full procedure is illustrated in Figure 25, “Workflow for a CDAP request from the Manager” below.

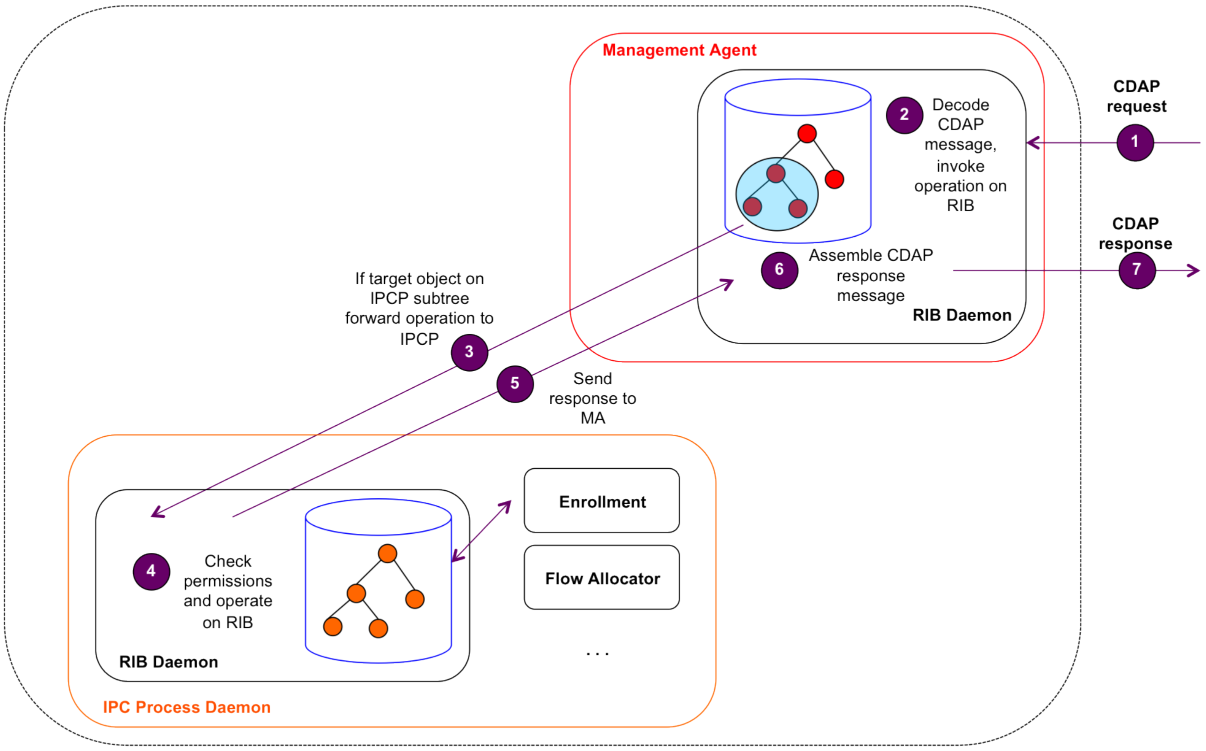

Management Agent
CDAP request
1
2
Decode CDAP message, invoke operation on RIB
CDAP response
7
6
Assemble CDAP response message
RIB Daemon
If target object on IPCP subtree forward operation to IPCP
3
5
Send response to MA
4
Check permissions and operate on RIB
RIB Daemon
Enrollment
Flow Allocator
. . .
IPC Process Daemon

Figure 25. Workflow for a CDAP request from the Manager

In the case of CDAP messages requesting the creation or destruction of IPC Processes, the workflow is slightly different. The IPC Manager Daemon is the entity that has the responsibility for managing the life-cycle of IPC Processes, therefore the MA has to delegate these requests to the IPC Manager Daemon, as depicted in Figure 26, "Workflow for a CDAP request targeting creation/destruction of IPCPs" below. Other operations would also follow this workflow, such as the Manager populating the Processing System with new policy sets.

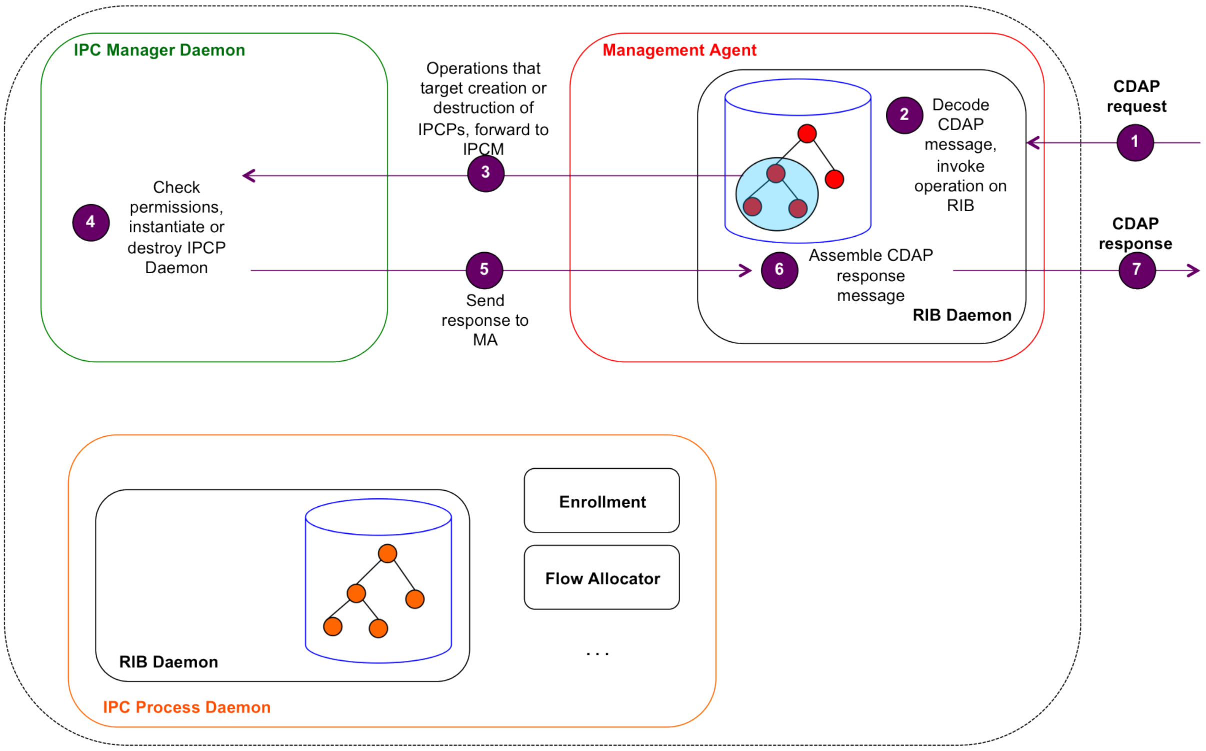

IPC Manager Daemon
Operations that target creation or destruction of IPCPs, forward to IPCM
Management Agent
CDAP request
1
2
Decode CDAP message, invoke operation on RIB
3
4
Check permissions, instantiate or destroy IPCP Daemon
5
Send response to MA
6
Assemble CDAP response message
RIB Daemon
CDAP response
7
Enrollment
Flow Allocator
...
RIB Daemon
IPC Process Daemon

Figure 26. Workflow for a CDAP request targeting creation/destruction of IPCPs

Requirements:

- **PROCREQ-1**: The MA needs to be able to request to the IPC Manager Daemon the creation and destruction of IPC Process Daemons, and to get the result of those operations.
- **PROCREQ-2**: The MA needs to be able to request operations on the RIB of the IPC Process Daemons in the processing system, and to get the result of those operations.

5.1.3. Processing a change in the state of a Managed IPC Process

One or more IPC Processes in the Processing System - where the MA resides - have an internal state change, produced by a change in one or more attributes of one or more objects in their RIBs. The MA has to learn about this state change events, log them, consult the Forwarding Discriminators and - in the case that a Manager was subscribed to that event - produce a report and send it to the Manager via the active CDAP connection. If the connectivity to the Manager was lost the MA can, at its discretion and based on the configuration, store the reports to be sent until the connectivity is restored and finally deliver them to the Manager.

The figure below (Figure 27, "Workflow of IPC Process triggering a Report being sent to the Manager") illustrates the aforementioned workflow. An internal task of the IPC Process Daemon triggers a change in an existing RIB object or an object is created/removed (e.g. the IPC Process has allocated a new flow). The RIB Daemon of the IPC Process checks if any of the MA(s) was subscribed to that event and, in case there were one or more, sends a notification to them capturing the change in the state - e.g. object(s) added, attribute(s) value changed, object(s) removed. The MA receives the notification, logs it and analyses the Forwarding Discriminators checking if a Report has to be produced and delivered to the Manager. If so, it creates a CDAP message with the report as the object value and sends it to the Manager process.

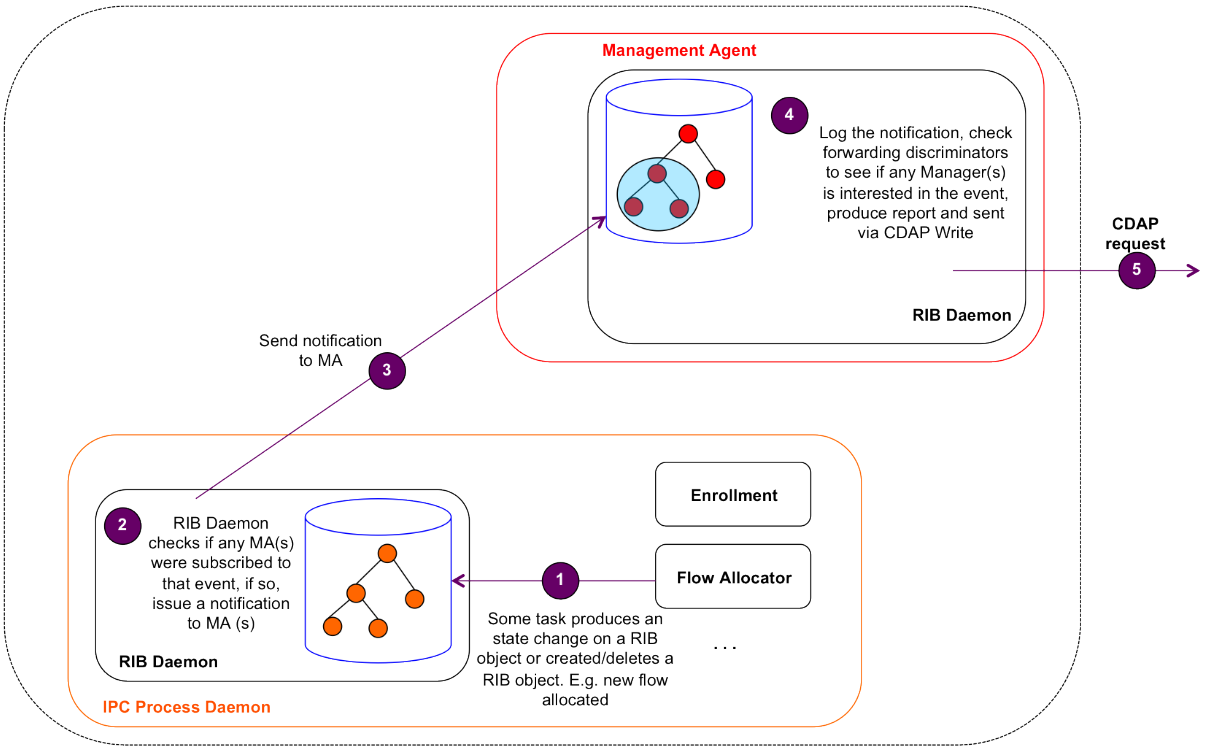

Management Agent
4
Log the notification, check forwarding discriminators to see if any Manager(s) is interested in the event, produce report and sent via CDAP Write
RIB Daemon
CDAP request
5
Send notification to MA
3
2
RIB Daemon checks if any MA(s) were subscribed to that event, if so, issue a notification to MA (s)
RIB Daemon
Enrollment
Flow Allocator
1
Some task produces an state change on a RIB object or created/deletes a RIB object. E.g. new flow allocated
. . .
IPC Process Daemon

Figure 27. Workflow of IPC Process triggering a Report being sent to the Manager

Requirements:

- **PROCCHG-1**: The MA needs to be notified when a state change in the IPC Process RIBs happens. This means, the MA subscribes to the changes of the IPC Process RIB.
- **PROCCHG-2**: The MA needs to allow the Manager to subscribe/ unsubscribe to different events representing state changes in the RIB exposed by the MA.

- **PROCCHG-3**: The MA needs to be able to produce reports from changes in the state of IPC Processes, and send a report informing about the event to the Manager if the Manager has subscribed to it.
- **PROCCHG-4**: The MA needs to be able to keep a log of the events. The logging strategy (e.g. type of events), verbosity and storage time, should be configurable.
- **PROCCHG-5**: The MA needs to be able to store reports in case the connectivity to the Manager has been interrupted.

5.2. Agent architecture

The MA functionalities will be implemented in a software component that will be interfaced to both the IPC Manager and IPC Process daemons. For the time being, refer to the documents ([irati-d21], [irati-d23], [irati-d31], [irati-d32] and [irati-d33]) for further details on the IRATI stack. IRATI's high level software architecture is depicted in Figure 28, "The IRATI software architecture", as it is not going to take major changes in order to fulfil the requirements envisioned for the first iteration of the PRISTINE project and thus they can be adopted as preliminary reference documentation.

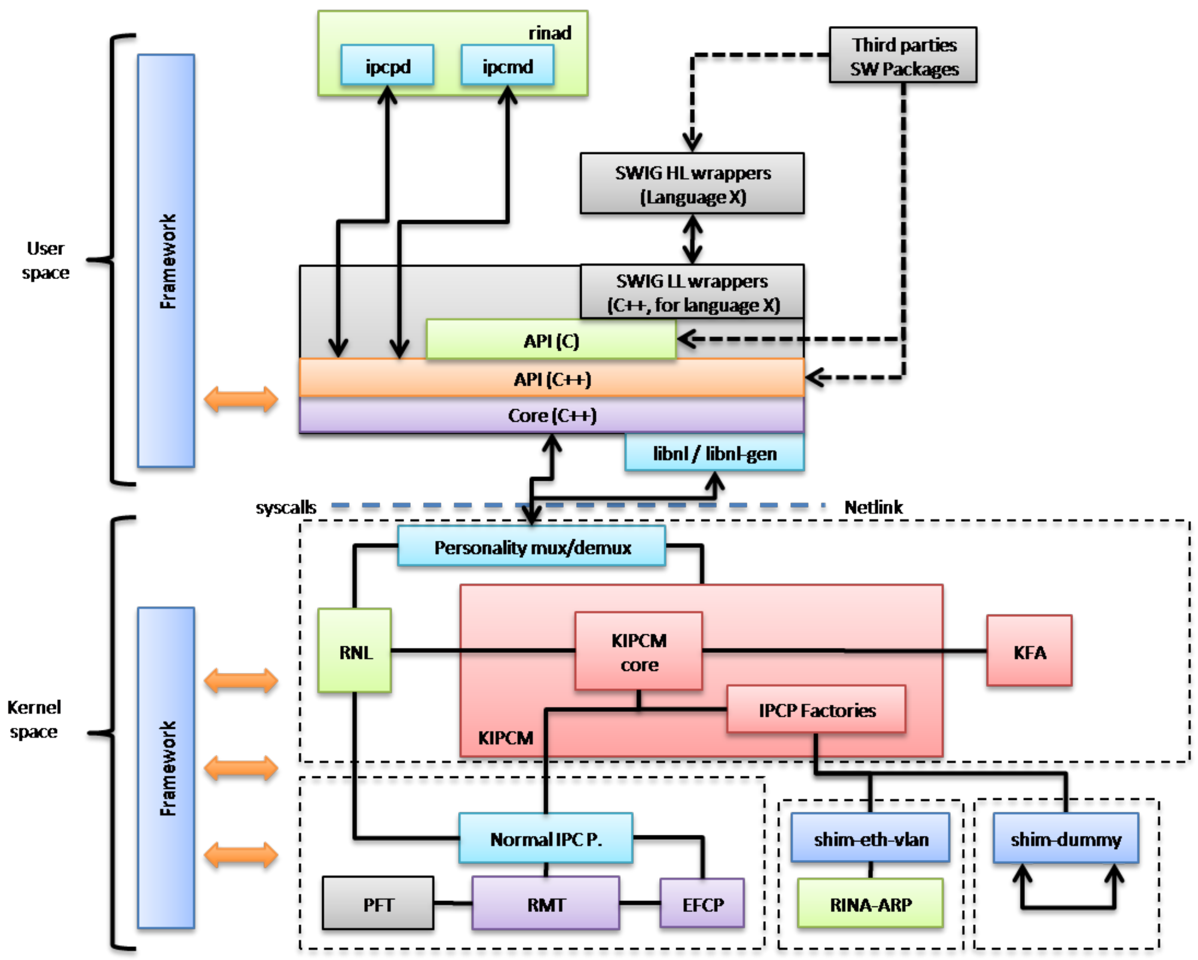


Figure 28. The IRATI software architecture

The main focus for the MA development within PRISTINE are the IPC Manager and IPC Process daemons (ipcmd and ipcpd in the above diagram) from the IRATI project [3].

The IPC Process daemon implements the layer management components of an IPC Process (RIB Daemon, RIB, CDAP parsers/generators, CACEP, Enrolment, Flow Allocation, Resource Allocation, Protocol Data Unit (PDU) Forwarding Table Generation, Security Management). The IPC Manager daemon primarily manages the IPC Process's life-cycle - e.g. the instantiation of an IPC Process daemon for each new IPC Process in the Processing System - and acts as a broker between applications and IPC Processes. Such daemons rely on a common framework - librina - which provides a common software development framework in order to maximise code reuse. The IPC Manager, IPC Process and librina high level software architecture is depicted in the following figure.

[3] IRATI ipcmd ⇒ PRISTINE IPCM and IRATI ipcmd ⇒ PRISTINE IPCP

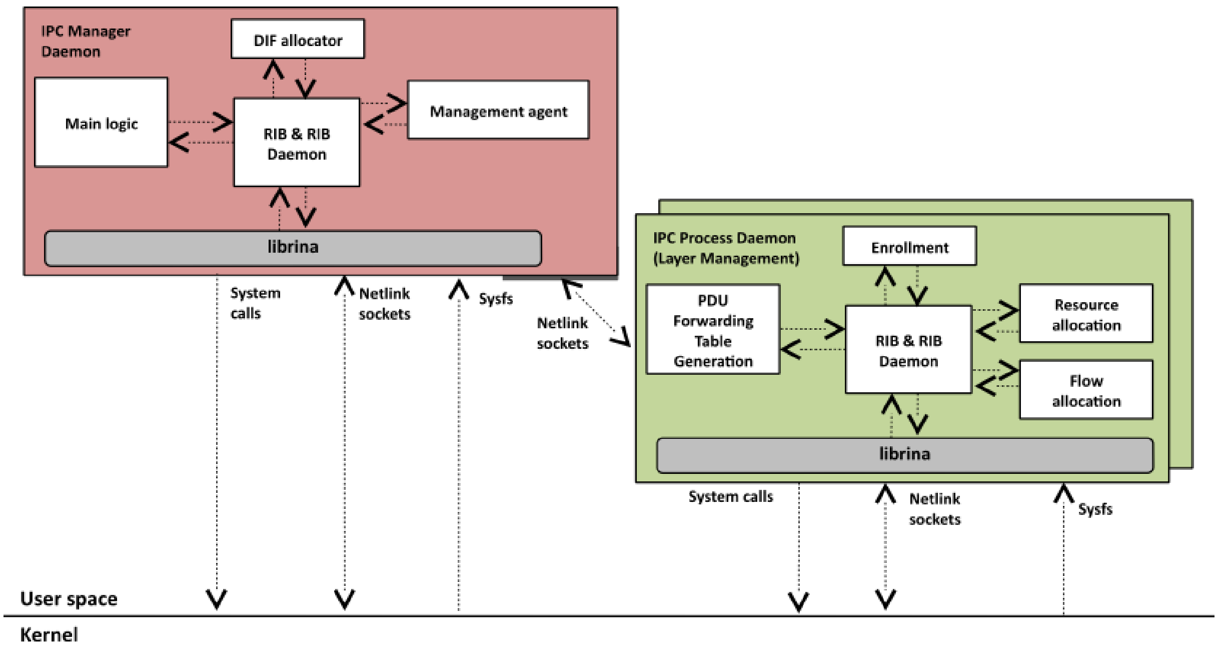


Figure 29. Details of the user-space architecture

The MA will have the structure of any RINA application process and its software architecture will leverage on librina features like IPC Process and IPC Manager daemons do, in order to implement its functionalities - i.e. RIB Daemon, CDAP parser/generator, SDU Protection, management of N-1 flows and enrolment.

In line with both the workflows, as described in the previous section, and the high-level architecture of the MA, as described in [D51], the software architecture of the MA can be decomposed into the components depicted in the following figure.

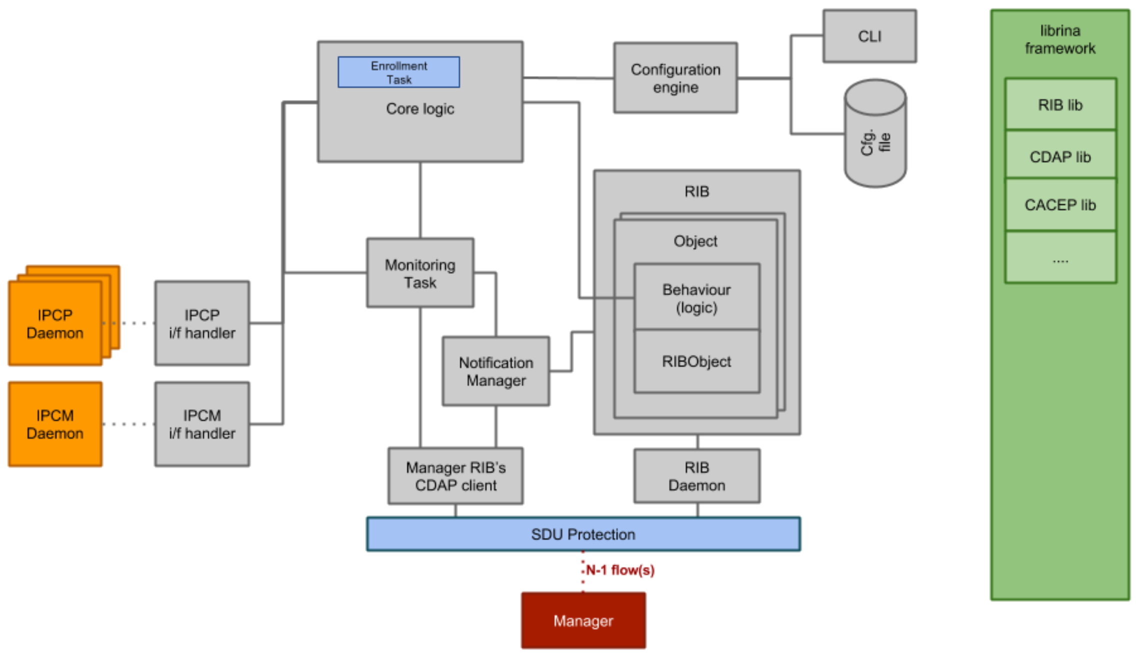


Figure 30. The Management Agent Daemon Software architecture

5.3. Components

The following sections provide details of the MA software components.

5.3.1. IPC Process and IPC Manager i/f handlers

These components manage the communications between both the Processing System's IPC Process and IPC Manager Daemons and the MA, eventually translating/mangling the received/to-be-sent events opportunely and finally dispatching them either internally or externally.

5.3.2. Monitoring Task

The Monitoring Task (MT) is the software component responsible for reacting to events coming from other entities, such as:

IPC Manager events
: Events of flow creation and destruction as well as changes in the state of the IPC Processes (i.e. deletion, creation)

IPC Process events
: Changes in their RIB Objects

OS events
: Critical events such as software failures, memory exhaustion, excessive CPU usage, hardware failures (e.g. I/O disk and Network Ineterface Card

(NIC) failures), changes in the hardware state (e.g. NIC link up/down events)

The aforementioned entities have different nature and thus the MT can be imagined as a place-holder containing different (background) activities that run in parallel. IPC Manager and IPC Process events will be handled by listeners that will be reacting to such events by propagating requests to the Core Logic. OS related events - such as NIC up/down events, CPU usage over/under-threshold - will be auto-generated by the MT, depending on both the (host) system configuration and the MA configuration file.

With reference to [D51], the MT module implements the IPC monitoring and OS monitoring functionalities.

5.3.3. Notification Manager

The Notification Manager (NM) is the software component responsible for generating reports upon change events of both the MA RIB and the Processing System's (delegated) IPC Process RIBs. The former events are collected by the NM directly from the MA's RIB while the latter are received through the $IPCP_x$ interface handlers. The NM packs such events into an internal queue which is enlarged until reaching a (configurable) threshold. Upon crossing that threshold, the NM generates a report which is forwarded to the Manager.

5.3.4. Core Logic

The Core Logic represents the container holding the "business logic" of the MA. It is primarily responsible of both performing the necessary bootstrapping actions as well as supporting the MA's runtime activities.

The bootstrapping phase will be started upon the reception of an event from the IPC Manager declaring it ready for processing commands. Upon receiving that event, the CORE will be populating the MA's RIB(s), initialise the internal subsystems, start the background jobs such as the Monitoring Task and will ask the IPC Manager to the setup the DMS DIFs - which will be required to operate correctly with the Manager. The bootstrapping will be considered complete upon successful enrolment to the DMS DAF.

5.3.5. RIB Daemon, RIB and RIB's CDAP client

The RIB Daemon manages the life-cycle of the RIB instance(s) pertaining to the MA. The logic behind the RIB Objects will be implementing the OS-mgmt and NM-mgmt functionalities described in [D51]. Refer to both the RIB Library and RIB sections for further details on the RIB Objects and the MA RIB.

5.3.6. Configuration engine

The Configuration Engine is the software component responsible of reacting to configuration commands that will be received either from a CLI or by reading a configuration file at bootstrap time. Its main purpose is to receive inputs to configure the rest of modules or components of the MA, e.g. CORE, NM, RIB daemon(s).

6. Future plans

This document describes the current progress achieved to develop a unified network management system for RINA. It focuses on the design decisions made for Management Agent (MA). This section will outline the roadmap for further implementation work for both the MA and the Manager.

6.1. Management Agent (MA)

Future work for the MA will include providing the ability to add / update RIB objects from the command line. In addition the MA will be integrated with the PRISTINE SDK.

Additional testing will be performed on the MA to evaluate the following functionality:

1. Join DMS DAF
2. Processing Manager requests
   a. Creation or destruction of IPC Process Daemons.
   b. Reading information about the state of one or more IPC Processes (by inspecting their RIBs).
   c. Modifying the state of one or more IPC Processes (by performing create, delete, write, start or stop operations on its RIBs).
3. Processing a change in the state of a Managed IPC Process

6.2. Manager

From the manager viewpoint, future work is needed to specify high-level strategies based on the Distributed cloud, Data-centre and Network service provider use cases (use-case specific strategies). These strategies will map low-level (RINA) strategies that will need to be deployed within the DIFs.

Some work is envisioned to analyse and define more generic strategies that are used by the manager (identify re-usable strategies). These strategies may optimise CDAP exchanges between the Manager and Management Agent.

From the testing point of view, some work is needed verify inter-operation between the Manager and Management agent (i.e CDAP commands) and the Management Agent and Manager (i.e event notifications). Strategically, this work in future may seed a RINA conformance and validation test suite.

List of definitions

Application Process (AP)
: The instantiation of a program executing in a processing system intended to accomplish some purpose. An Application Process contains one or more tasks or Application-Entities, as well as functions for managing the resources (processor, storage, and IPC) allocated to this AP.

Common Application Connection Establishment Phase (CACEP)
: CACEP allows Application Processes to establish an application connection. During the application connection establishment phase, the APs exchange naming information, optionally authenticate each other, and agree in the abstract and concrete syntaxes of CDAP to be used in the connection, as well as in the version of the RIB. It is also possible to use CACEP connection establishment with another protocol in the data transfer phase (for example, HTTP).

Common Distributed Application Protocol (CDAP)
: CDAP enables distributed applications to deal with communications at an object level, rather than forcing applications to explicitly deal with serialization and input/output operations. CDAP provides the application protocol component of a Distributed Application Facility (DAF) that can be used to construct arbitrary distributed applications, of which the DIF is an example. CDAP provides a straightforward and unifying approach to sharing data over a network without having to create specialized protocols.

Distributed Application Facility (DAF)
: A collection of two or more cooperating APs in one or more processing systems, which exchange information using IPC and maintain shared state.

Distributed-IPC-Facility (DIF)
: A collection of two or more Application Processes cooperating to provide Interprocess Communication (IPC). A DIF is a DAF that does IPC. The DIF provides IPC services to Applications via a set of API primitives that are used to exchange information with the Application's peer.

IPC-Process
An Application-Process, which is a member of a DIF and implement locally the functionality to support and manage IPC using multiple sub-tasks.

(N)-DIF
The DIF from whose point of view a description is written.

(N+1)-DIF
A DIF that uses a (N)-DIF. A (N)-DIF may only know the application process names of IPC-Processes in a (N+1)-DIF. Depending on the degree of trust between adjacent DIFs, (N)-DIF management may share other information with a (N-1)-DIF.

(N-1)-DIF
A DIF used by a (N)-DIF. The IPC Processes on the (N)-DIF appear as ordinary Application Processes to a (N-1)-DIF. Depending on the degree of trust between adjacent DIFs, (N)-DIF management may share other information with a (N-1)-DIF.

PDU
Protocol Data Unit, The string of octets exchanged among the Protocol Machines (PM). PDUs contain two parts: the header, which is understood and interpreted by the DIF, and User-Data, that is incomprehensible to this PM and is passed to its user

Resource Information Base (RIB)
The logical representation of information held by the IPC Process for the operation of the DIF.

shim DIF
The task of a shim DIF is to put as small as possible a veneer over a legacy protocol to allow a RINA DIF to use it unchanged. It is always the lowest level DIF and contains shim IPC processes.

shim IPC Process
An IPC Process that is an interface between a RINA DIF and the selected communication technology (Ethernet for example). It is distinguished as it may not provide full (or normal) IPC process functionality.

1. Acronym list

AE	Application Entity
AP	Application Process
CACEP	Common Application Connection Establishment Phase
CDAP	Common Distributed Application Protocol
CLI	Command Line Interface
DAF	Distributed Application Facility
DAP	Distributed Application Process
DIF	Distributed-IPC-Facility
DMS	Distributed Management System
IPC	Inter-Process Communication
IPCP	Inter-Process Communication Process
IPCM	Inter-Process Communication Manager
LLCR	Lower-Layer CDAP RIB library
MA	Management Agent
MO	Managed Object
MT	Monitoring Task (within Agent)
NIC	Network Interface Card
NM	Network Management
PDU	Protocol Data Unit
RIB	Resource Information Base
RINA	Recursive Inter-Network Architecture
SDU	Service Data Unit

References

- [irati-d21] FP7-IRATI Project - D2.1 First phase use cases updated RINA specifications and high-level software architecture. Available online at online[4].
- [irati-d23] FP7-IRATI Project - D2.3 Second phase use cases updated RINA specification and high-level software architecture (under EC evaluation). Available online at online[5].
- [irati-d31] FP7-IRATI Project - D3.1 First phase integrated RINA prototype over Ethernet for UNIX-like OS (under EC evaluation). Available online at online[6].
- [irati-d32] FP7-IRATI Project - D3.2 Second phase integrated RINA prototype over Ethernet for a UNIX-like OS (under EC evaluation). Available online at online[7].
- [irati-d33] FP7-IRATI Project - D3.3 Second phase integrated RINA prototype for Hypervisors for a UNIX-like OS (under EC evaluation). Available online at online[8].
- [D21] PRISTINE Consortium. Deliverable 2.1. Use cases description and requirements analysis report. May 2014. Available online[9].
- [D41] PRISTINE Consortium. Deliverable 4.1. Draft Conceptual and High-Level Engineering Design of Innovative Security and Reliability Enablers. September 2014. Available online at online[10].
- [D51] PRISTINE Consortium. Deliverable D5.1. Draft specification of common elements of the management framework. June 2014. Available online[11].

[4] http://irati.eu/wp-content/uploads/2012/07/IRATI-D2.1.pdf
[5] http://irati.eu/wp-content/uploads/2012/07/IRATI-D2.3.zip
[6] http://irati.eu/wp-content/uploads/2012/07/IRATI-D3.1-v1.0.pdf
[7] http://irati.eu/wp-content/uploads/2012/07/IRATI-D3.2-v1.0.pdf
[8] http://irati.eu/wp-content/uploads/2012/07/IRATI-D3.3-bundle.zip
[9] http://ict-pristine.eu/wp-content/uploads/2013/12/pristine_d21-usecases-and-requirements_draft.pdf
[10] http://ict-pristine.eu/wp-content/uploads/2013/12/pristine_d41-security-and-reliability-enablers_draft.pdf
[11] http://ict-pristine.eu/wp-content/uploads/2013/12/pristine_d51-common-management-elements_draft.pdf

- [x722] ITU-T. 'X.722 : Information technology - Open Systems Interconnection - Structure of management information: Guidelines for the definition of managed objects'. February 2000. Available online[12].

[12] https://www.itu.int/rec/T-REC-X.722/en

A. Appendix A: Types

A.1. Basic Types

According to [D51] and with the addition of the boolean type, basic types are:

- INTEGER
- UNSIGNED INTEGER
- REAL.
- UNSIGNED REAL
- STRING
- ENUMERATION
- BYTE STRING
- DATE
- TIME
- BOOL

Types are always represented by caps.

More complex types can be defined by the following constructs:

- SEQUENCE: ordered collection of variables of different types.
- SEQUENCE_OF: ordered collection of variables of the same type.
- SET: unordered collection of variables of different types. [1]
- SET_OF: unordered collection of variables of the same type.
- CHOICE : collection of distinct types from where to choose a type.

A.2. Complex types

Using the simple and complex types defined above, it is possible to define the RIB Managed Objects (MO).

[1]Due to the performance implications of Set when encoding and decoding, "SET_OF" and "SEQUENCE_OF" are preferred representations.

A.2.1. AE_REGISTRATION

Type used to represent one registration.

- *STRING entityName*: the name of the AE of the IPC Process registered to the N-1 DIF(s) (can be null).
- *STRING entityInstance*: the instance of the AE of the IPC Process registered to the N-1 DIF(s) (can be null).
- *SET_OF STRING difNames*: the names of the N-1 DIFs where the AE is registered.

A.2.2. APP_NAMING_INFO

Type used to store the Application Naming Information. It is represented as a type SEQUENCE and contains the following fields:

- *STRING processName*: Name of the process.
- *STRING processInstance*: instance of the process.

A.2.3. CREATE_IPCP_CONFIG

Type used to store the configurations needed for enrolment. It is represented as a type SEQUENCE and contains the following fields:

- *STRING neighbour_name*: the name of the neighbour to enrol to.
- *Section A.2.17, "IPCP_CONFIG"*: configurations of the IPC process.

A.2.4. CONNECTION_REQUEST

Type used to represent the information needed to request a connection.

- *UNSIGNED INTEGER srcCepId*: the source connection-endpoint id.
- *UNSIGNED INTEGER srcAddress*: the address of the source IPC Process.
- *UNSIGNED INTEGER destCepId*: the destination connection-endpoint id.
- *UNSIGNED INTEGER destAddress*: the address of the destination IPC Process.

- *UNSIGNED INTEGER qosId*: the id of the QoS cube where the flow supported by this connection belongs.
- *UNSIGNED INTEGER portId*: the id of the flow supported by this connection.
- *BOOL dtcpPresent*: indicates if DTCP is active for this connection.
- *UNSIGNED INTEGER initialATimer*: indicates the initial value of the A-timer.
- *UNSIGNED INTEGER seqNumRollOverThres*: the sequence number roll-over threshold.

A.2.5. CONNECTION_STATE

Type used to represent the state of a connection.

- *Section A.2.4, "CONNECTION_REQUEST" connectionRequest*: the basic information of the connection.
- *Section A.2.25, "POLICY_CONFIG" initSeqNumPolicy*: configuration of the initial sequence number policy.
- *Section A.2.25, "POLICY_CONFIG" sdrTimerInacPolicy*: configuration of the sender timer inactivity policy.
- *Section A.2.25, "POLICY_CONFIG" rcvrTimerInacPolicy*: configuration of the receiver timer inactivity policy.

A.2.6. DATA_TRANSFER_STATE

Type used to represent the state of the data transfer module.

- *UNSIGNED INTEGER addressLength*: length of the address field in the DTP header (in bytes).
- *UNSIGNED INTEGER qosIdLength*: length of the qos-id field in the DTP header (in bytes).
- *UNSIGNED INTEGER portIdLength*: length of the port-id field in the DTP header (in bytes).
- *UNSIGNED INTEGER cepIdLength*: length of the cep-id field in the DTP header (in bytes).

- *UNSIGNED INTEGER seqNumLength*:, length of the sequence number field in the DTP header (in bytes).
- *UNSIGNED INTEGER lengthLength*: length of the length field in the DTP header (in bytes).
- *UNSIGNED INTEGER maxPDUSize*: the maximum size of a DTP PDU (in bytes).
- *UNSIGNED INTEGER maxSDUSize*: the maximum size of an SDU written to the flow supported by this connection (in bytes).
- *Section A.2.25, "POLICY_CONFIG" unknownFlowPolicy*: configuration of the unknown flow policy.
- *Section A.2.25, "POLICY_CONFIG" pduForwardingTableGeneratorPolicy*: the configuration of the PDU Forwarding Table Generator, to generate the PDU Forwarding Table with input from routing protocols and other sources of information.

A.2.7. DFT_ENTRY

Type used to represent one directory forwarding table entry.

- *UNSIGNED INTEGER key*: unique key of this entry in the PDU Forwarding Table.
- *Section A.2.2, "APP_NAMING_INFO" name*: the destination application process naming information.
- *UNSIGNED INTEGER address*: the address of the next IPC Process where the Flow Request should be forwarded.

A.2.8. DIF_PROPERTIES

Type used to represent the properties of a DIF.

- *STRING difName*: name of the underlying (N-1) DIF.
- *UNSIGNED INTEGER maxSDUSize*: maximum SDU size allowed by the DIF (in bytes).
- *UNSIGNED INTEGER mpl*: maximum PDU lifetime in the DIF (in milliseconds).

- *SEQUENCE_OF Section A.2.28, "QOS_CUBE_DESCRIPTION" qosCubes*: description of the QoS cubes provided by the DIF.

A.2.9. DISCRIMINATOR_POLICY_CONFIG

Type used to store the configuration of a forwarding discriminator policy.

- *STRING destinationName*: name of the subscriber.
- *UNSIGNED INTEGER destinationInstance*: instance of the subscriber.
- *SET_OF Section A.2.21, "NOTIFICATION" notifications*: list of the notifications where the discriminator must be subscribed.
- *Section A.2.25, "POLICY_CONFIG" filteringPolicy*: configurations of the notification filtering policy.
- *UNSIGNED INTEGER lostNotifications*: count of the number of discarded notifications since the one previously sent.

A.2.10. DTCP_STATE

Type used to represent the state of the DTCP module.

- *UNSIGNED INTEGER flowControl*: 0 means flow control not in use, 1 means sliding window flow control, 2 means rate-based flow control.
- *BOOL rtxControl*: true if retransmission control is enabled, false otherwise.
- *POLICY_CONFIG lostControlPDUPolicy*: configuration of the Lost Control PDU Policy.
- *POLICY_CONFIG rttEstimatorPolicy*: configuration of the RTT estimator Policy.
- *UNSIGNED INTEGER maxTimeRetry*: Max time to attempt the retransmission of a packet, in ms (this is R).
- *UNSIGNED INTEGER dataRxmsnMax*: Max number of retransmission attempts.
- *UNSIGNED INTEGER portId*: the id of the flow supported by this connection.

- *UNSIGNED INTEGER intialRtxTime*:indicates the time to wait before transmitting a PDU.
- *POLICY_CONFIG rtxTimerExpiryPolicy*: configuration of the Retransmission Timer Expiry Policy.
- *POLICY_CONFIG sdrAckPolicy*: configuration of the Sender ACK Policy.
- *POLICY_CONFIG rcvingAckListPolicy*: configuration of the Receiving ACK List Policy.
- *POLICY_CONFIG rcvrAckPolicy*: configuration of the Receiver ACK Policy.
- *POLICY_CONFIG sendingAckListPolicy*: configuration of the Sending ACK List Policy.
- *POLICY_CONFIG rcvingAckListPolicy*: configuration of the Receiving ACK List Policy.
- *POLICY_CONFIG rcvrCtrlAckPolicy*: configuration of the Receiver Control Ack Policy.
- *POLICY_CONFIG clsdWindowPolicy*: configuration of the Closed Window Policy.
- *POLICY_CONFIG flowCtrlOverrunPolicy*: configuration of the Flow Control Overrun Policy.
- *POLICY_CONFIG rcvingAckListPolicy*: configuration of the Receiving ACK List Policy.

A.2.11. EFCP_CONNECTION_CONFIG

Type used to store the configurations of a EFCP connection. It is represented as a type SEQUENCE and contains the following fields:

- *BOOL dtcpPresent*: True if DTCP is active for the connection.
- *UNSIGNED INTEGER initialATimer*:
- *UNSIGNED INTEGER seqNumRolloverThreshold*: The sequence number roll-over threshold.
- *POLICY_CONFIG rcvrTimerInactivityPolicy*: Configuration of the receiver timer inactivity policy.

- *POLICY_CONFIG sdrTimerInactivityPolicy*: Configuration of the sender timer inactivity policy.
- *POLICY_CONFIG init_seqNumPolicy*: Configuration of the initial sequence number policy.
- *SEQUENCE dtcpConfig*: Configuration of the DTCP component.
  - *BOOL flowControlPresent*: Indicates whether flow control is in use.
  - *BOOL rtxControlPresent*: Indicates whether retransmission control is in use.
  - *POLICY_CONFIG lostControlPduPolicy*: Configuration of the lost control PDU policy.
  - *POLICY_CONFIG rttEstimatorPolicy*: Configuration of the Round-trip time estimator policy.
  - *SEQUENCE retxControlConfig*: Configuration of the retransmission control component.
    - *UNSIGNED INTEGER maxTimeToRetry*: Maximum time to attempt the retransmission of a packet.
    - *UNSIGNED INTEGER dataRtxMax*: The maximum amount of times the retransmission of a PDU will be attempted before some action occurs.
    - *UNSIGNED INTEGER initRtxTime*: Time to wait before retransmitting a PDU.
    - *POLICY_CONFIG rtx_timerExpiryPolicy*: Configuration of the retransmission timer expiry policy.
    - *POLICY_CONFIG senderAckPolicy*: Configuration of the sender ACK policy.
    - *POLICY_CONFIG rcvingAckListPolicy*: Configuration of the receiving ACK list policy.
    - *POLICY_CONFIG rcvrAckPolicy*: Receiver ACK policy.
    - *POLICY_CONFIG sendingAckPolicy*: Sending ACK policy.
    - *POLICY_CONFIG rcvrControlAckPolicy*: Configuration of the receiver control ACK policy.
  - *SEQUENCE flowControlConfig*: Configuration of the flow control component.

- *BOOL windowBased*: Is window-based flow control in use.
- *BOOL rateBased*: Is rate-based flow control in use. s*** *UNSIGNED INTEGER rcvBytesThres*: Receive bytes threshold.
- *UNSIGNED INTEGER rcvBytesPercent_thres*:
- *UNSIGNED INTEGER rcvBuffersThres*:
- *UNSIGNED INTEGER rcvBuffersPercent_thres*:
- *UNSIGNED INTEGER sendBytesThres*: Send bytes threshold.
- *UNSIGNED INTEGER sendBytesPercent_thres*:
- *UNSIGNED INTEGER sendBuffersThres*:
- *UNSIGNED INTEGER sendBuffersPercentThres*:
- *POLICY_CONFIG closedWindowPolicy*: Configuration of the closed window policy.
- *POLICY_CONFIG flowCtrlOverrunPolicy*: Configuration of the flow control overrun policy.
- *POLICY_CONFIG recFlowConflictPolicy*: Configuration of the reconcile flow conflict policy.
- *POLICY_CONFIG rcvingFlowCtrlPolicy*: Configuration of the receiving flow control policy.
- *SEQUENCE windowBasedConfig*: Configuration of window-based flow control.
  - *UNSIGNED INTEGER maxClosedWindow_queue_length*: Max length of the closed window queue.
  - *UNSIGNED INTEGER initialCredit*: Initial credit.
  - *POLICY_CONFIG txCtrlPolicy*: Configuration of the transmission control policy.
  - *POLICY_CONFIG rcvrFlowCtrlPolicy*: Configuration of the receiver flow control policy.
- *SEQUENCE rateBasedConfig*: Configuration of the rate-based flow control.
  - *UNSIGNED INTEGER sendingRate*:

  -

- *UNSIGNED INTEGER timePeriod*:
- *POLICY_CONFIG noRateSlowDownPolicy*: Configuration of the no rate slow down policy.
- *POLICY_CONFIG noOverrideDefPeakPolicy*: Configuration of the no override default peak policy.
- *POLICY_CONFIG rateReductionPolicy*: Configuration of the rate reduction policy.

A.2.12. ENROLLMENT_STATE

Type used to represent the state of enrolment module.

- *Section A.2.25, "POLICY_CONFIG" enrolmentPolicy*: the configuration of the enrolment policy.
- *Section A.2.25, "POLICY_CONFIG" newMemberAccessControlPolicy*: the configuration of the new member access control policy.

A.2.13. FLOW_ALLOCATOR_STATE

Type used to represent the state of the flow allocator.

- *Section A.2.25, "POLICY_CONFIG" newFlowRequestPolicy*: the configuration of the New Flow Request policy.
- *Section A.2.25, "POLICY_CONFIG" seqRollOverPolicy*: the configuration of the sequence number roll-over policy.
- *Section A.2.25, "POLICY_CONFIG" allocateNotifyPolicy*: the configuration of the allocate notify policy.
- *Section A.2.25, "POLICY_CONFIG" allocateRetryPolicy*: the configuration of the allocate retry policy.
- *Section A.2.25, "POLICY_CONFIG" newFlowAccessControlPolicy*: the configuration of the new flow access control policy (for incoming flows).
- *UNSIGNED INTEGER inFlowReq*: the total number of incoming flow requests.
- *UNSIGNED INTEGER inFlowReqRej*: the number of incoming flow requests that have been rejected.

- *UNSIGNED INTEGER outFlowRes*: the total number of outgoing flow requests.
- *UNSIGNED INTEGER outFlowReqRej*: the number of outgoing flow requests that have been rejected.

A.2.14. FLOW_PROPERTIES

Type used to represent the properties of a flow.

- *UNSIGNED INTEGER averageBandwidth*: in bits/second.
- *UNSIGNED INTEGER averageSduBandwidth*: in sdus/second.
- *UNSIGNED INTEGER peakBandwidthDuration*: in milliseconds.
- *UNSIGNED INTEGER burstPeriod*: in seconds.
- *UNSIGNED INTEGER burstDuration*: in fractions of Burst period.
- *REAL NUMBER undetectedBitErrorRate*: probability.
- *UNSIGNED INTEGER maxSduSize*: in bytes.
- *BOOL partialDelivery*: is partial delivery allowed.
- *BOOL incompleteDelivery*: is incomplete delivery in order.
- *BOOL in_orderDelivery*: SDUs have to be delivered in order.
- *UNSIGNED INTEGER maxAllowedSduGap*: Maximum allowable gap in SDUs.
- *UNSIGNED INTEGER maxDelay*: in milliseconds.
- *UNSIGNED INTEGER maxJitter*: in milliseconds.

A.2.15. FLOW_REQUEST

Type used to represent the information needed to request a Flow.

- *Section A.2.2, "APP_NAMING_INFO" remoteAppName*, the remote application entity that is using the N flow.
- *Section A.2.14, "FLOW_PROPERTIES" flowProperties*, the characteristics of the N flow (loss, delay, reliability, in order-delivery of SDUs, etc).

A.2.16. FLOW_STATE

Type used to represent the state of a flow.

- *UNSIGNED INTEGER localPortId*: the portId of the flow.
- *Section A.2.2, "APP_NAMING_INFO" localAppName*: the local application entity that is using the N flow.
- *UNSIGNED INTEGER remotePortId*: the portId of the flow at the remote IPC Process.
- *UNSIGNED INTEGER state*: 0 allocation in progress, 1 allocated, 2, deallocation in progress, 3 deallocated.
- *UNSIGNED INTEGER maxCreateFlowRetries*: the maximum number of attempts for allocating the flows.
- *UNSIGNED INTEGER createFlowRetries*: the current number of attempts for allocating this flow.
- *UNSIGNED INTEGER reservedCepIds*: the connection-endpoint ids reserved for the connections that will support this flow in this IPC Process.
- *UNSIGNED INTEGER currentCepId*: the connection-endpoint currently used by the connection supporting this flow.

A.2.17. IPCP_CONFIG

Type used to store the configurations of an IPC Process. It is represented as a type SEQUENCE and contains the following fields:

- *UNSIGNED INTEGER ipcProcessID*: the id of the IPC Process.
- *STRING processName*: the IPC Process name.
- *STRING processInstance*: the IPC Process instance.
- *SET_OF STRING synonymList*: list of synonyms of the IPC Process.
- *SET_OF STRING n_minus_one_difs*: the names of the N-1 DIFs where the new IPC Process will be registered.
- *STRING difName*: the name of the DIF where the IPC Process is assigned.
- *Section A.2.25, "POLICY_CONFIG" fragmentationPolicy*: Configuration of the fragmentation policy.
- *Section A.2.25, "POLICY_CONFIG" concatenationPolicy*: Configuration of the concatenation policy.
- *Section A.2.25, "POLICY_CONFIG" reassemblySeparationPolicy*: The reassembly and separation policy.

- *UNSIGNED INTEGER addressLength*: The length of an address in bits.
- *UNSIGNED INTEGER qosIDLength*: The length of a qos-id in bits.
- *UNSIGNED INTEGER portIDLength*: The length of a port_id in bits.
- *UNSIGNED INTEGER cepIDLength*: The length of the cep_id in bits.
- *UNSIGNED INTEGER seqNumLength*: The length of the sequence number in bits.
- *UNSIGNED INTEGER lengthLength*: The length of the length in bits.ameters.
- *SET_OF UNSIGNED INTEGER qosCubeIDs*: the ids of the QoS cube supported by this DIF.
- *UNSIGNED INTEGER maxPduSize*: The maximum size of a PDU in the DIF.
- *UNSIGNED INTEGER maxSduSize*: The maximum size of an SDU accepted by the IPC Process in this DIF.
- *Section A.2.25, "POLICY_CONFIG" unknownFlowPolicy*: Configuration of the unknown flow policy.
- *Section A.2.25, "POLICY_CONFIG" rmtQMonitorPolicy*: Configuration of the RMT queue monitor policy.
- *Section A.2.25, "POLICY_CONFIG" rmtSchedPolicy*: Configuration of the RMT scheduling policy.
- *Section A.2.25, "POLICY_CONFIG" maxQPolicy*: Configuration of the max queue policy.
- *Section A.2.25, "POLICY_CONFIG" pduForwardingPolicy*: Configuration of the PDU forwarding policy.
- *SET_OF Section A.2.35, "SDU_PROTECTION_POLICY_SET_CONFIG" sduProtPolicies*: A list of SDU Protection policy configurations.
- *Section A.2.25, "POLICY_CONFIG" authenticationPolicy*: Configuration of the authentication policy.
- *UNSIGNED INT concreteSyntaxID*: Id of the concrete syntax used by CDAP.
- *UNSIGNED INT ribVersion*: The RIB version to be used.
- *UNSIGNED INT objetConcreteSyntax*: The concrete syntax of the objects to be used.

- *Section A.2.25, "POLICY_CONFIG" updatePolicy*: Configuration of the update policy.
- *Section A.2.25, "POLICY_CONFIG" replicationPolicy*: Configuration of the replication policy.
- *Section A.2.25, "POLICY_CONFIG" subscriptionPolicy*: Configuration of the subscription policy.
- *Section A.2.25, "POLICY_CONFIG" loggingPolicy*: Configuration of the logging policy.
- *Section A.2.25, "POLICY_CONFIG" allocateNotifyPolicy*: Configuration of the Allocate Notify Policy.
- *Section A.2.25, "POLICY_CONFIG" allocateRetryPolicy*: Configuration of the Allocate Retry Policy.
- *Section A.2.25, "POLICY_CONFIG" newFlowReqPolicy*: Configuration of the New Flow Request Policy.
- *Section A.2.25, "POLICY_CONFIG" seqRollOverPolicy*: Configuration of the sequence number roll-over Policy.
- *Section A.2.25, "POLICY_CONFIG" addrValPolicy*: Configuration of the address validation policy.
- *Section A.2.25, "POLICY_CONFIG" addrAssPolicy*: Configuration of the address assignment policy.
- *Section A.2.25, "POLICY_CONFIG" dftReplPolicy*: Configuration of the Directory Forwarding Table replication policy.
- *Section A.2.25, "POLICY_CONFIG" dftGenPolicy*: Configuration of the Directory Forwarding Table generation policy.
- *Section A.2.27, "QOS_CUBE_CONFIG" qosCubes*: the QoS cubes supported by the DIF.
- *Section A.2.25, "POLICY_CONFIG" resourceAllocPolicy*: Configuration of the resource allocator policy.
- *Section A.2.25, "POLICY_CONFIG" pduFwdingTableGeneratorPolicy*: Configuration of the PDU Forwarding Table Generator.
- *Section A.2.25, "POLICY_CONFIG" routingPolicy*: configuration of the routing policy.
- *Section A.2.25, "POLICY_CONFIG" newMemAccCtrlPolicy*: Configuration of the new member access control policy.

- *Section A.2.25, "POLICY_CONFIG" newFlowAccCtrlPolicy*: Configuration of the new flow access control policy.
- *Section A.2.25, "POLICY_CONFIG" ribAccCtrlPolicy*: Configuration of the RIB access control policy.
- *Section A.2.25, "POLICY_CONFIG" auditPolicy*: Configuration of the auditing policy.
- *Section A.2.25, "POLICY_CONFIG" credentialMgmtPolicy*: Configuration of the credential management policy.

A.2.18. NEIGHBOR_REQUEST

Type used to represent a request to register one neighbour

- *STRING processName*: the application process name of the neighbour IPC Process.
- *UNSIGNED INTEGER processInstance*: the application process instance of the neighbour IPC Process.
- *SET_OF STRING underlyingDIFs*: the names of the N-1 DIFs in common with the neighbour IPC Process.

A.2.19. NEIGHBOR_STATE

Type used to represent the state of one neighbour.

- *BOOL isEnrolled*: true if the neighbour is currently enrolled. False otherwise.
- *UNSIGNED INTEGER address*: the current address of the neighbour IPC Process.
- *SET_OF UNSIGNED INTEGER underlyingFlows*: the port-id of the N-1 flow used to talk to the neighbour.
- *Section A.2.25, "POLICY_CONFIG" authenticationPolicy*: the configuration of the authentication policy used to authenticate the neighbour IPC Process.
- *UNSIGNED INTEGER numberOFEnrollmentAttempts*: the number of enrolment attempts tried with the neighbour (or that the neighbour has tried with this IP Process).

- *Section A.2.18, "NEIGHBOR_REQUEST"*: basic neighbour information.

A.2.20. NEXT_HOP_TABLE_ENTRY

Type used to represent one next hop table entry.

- *UNSIGNED INTEGER key*: unique key of this entry in the Next Hop Table.
- *UNSIGNED INTEGER address*: the destination address to be matched.
- *UNSIGNED INTEGER qosId*: the id of the QoS cube of the flow where the PDUs to be routed belong to.
- *UNSIGNED INTEGER nhopAddresses*: the addresses of the IPC Processes that are the next hops for the PDUs.

A.2.21. NOTIFICATION

Type used to represent one notification

- *STRING object*: Fully distinguished name of the object emitting the notification.
- *STRING notification*: Name of the notification where the discriminator must be subscribed to.

A.2.22. NSM_STATE

Type used to represent the state of the name space manager.

- *Section A.2.25, "POLICY_CONFIG" addressAssignmentPolicy*: the configuration of the address assignment policy.
- *Section A.2.25, "POLICY_CONFIG" addressValidationPolicy*: the configuration of the new address validation policy.
- *Section A.2.25, "POLICY_CONFIG" directoryForwardingTableGeneratorPolicy*: the configuration of the directory forwarding table

A.2.23. PAIR

- *STRING name*: name of the attribute.

- *STRING value*: value of the attribute.

A.2.24. PDU_FT_ENTRY

Type used to represent one PDU forwarding table entry.

- *UNSIGNED INTEGER key*: unique key of this entry in the PDU Forwarding Table.
- *UNSIGNED INTEGER address*: the destination address to be matched.
- *UNSIGNED INTEGER qosId*: the id of the QoS cube of the flow where the PDUs to be forwarded belong to.
- *UNSIGNED INTEGER portIds*: the N-1 flow(s) through which the PDU has to be forwarded.

A.2.25. POLICY_CONFIG

Type used to store the configurations of a policy. It is represented as a type SEQUENCE and contains the following fields:

- *STRING name*: Name of the policy.
- *STRING version*: Version of the policy.
- *SET_OF Section A.2.26, "POLICY_CONFIG_PARAMETER" parameters*: Parameters associated to the policy.

A.2.26. POLICY_CONFIG_PARAMETER

Type used to store the configurations of a policy parameter. It is represented as a type SEQUENCE and contains the following fields:

- *STRING name*: Name of the parameter.
- *STRING value*: Value of the parameter.

A.2.27. QOS_CUBE_CONFIG

Type used to store the configurations of a QoS Cube. It is represented as a type SEQUENCE and contains the following fields:

- *Section A.2.28, "QOS_CUBE_DESCRIPTION" qosCubeDescription*: parameters of the QoS cube.

- *Section A.2.11, "EFCP_CONNECTION_CONFIG" efcpConnectionsConfig*: Configuration of EFCP connections that support flows matched to this QoS cube.

A.2.28. QOS_CUBE_DESCRIPTION

Type used to store the description (external view) of a QoS Cube. It is represented as a type SEQUENCE and contains the following fields:

- *STRING name*: name of the QoS cube.
- *UNSIGNED INTEGER id*: id of the QoS cube.
- *Section A.2.29, "RANGE_OF" UNSIGNED INTEGER averageBandwidth*: in bits/second.
- *Section A.2.29, "RANGE_OF" UNSIGNED INTEGER averageSduBandwidth*: in sdus/second.
- *Section A.2.29, "RANGE_OF" UNSIGNED INTEGER peakBandwidthDuration*: in milliseconds.
- *Section A.2.29, "RANGE_OF" UNSIGNED INTEGER burstPeriod*: in seconds.
- *Section A.2.29, "RANGE_OF" UNSIGNED INTEGER burstDuration*: in fractions of Burst period.
- *Section A.2.29, "RANGE_OF" REAL NUMBER undetectedBitErrorRate*: probability.
- *UNSIGNED INTEGER maxSduSize*: in bytes.
- *BOOL partialDelivery*: is partial delivery allowed.
- *BOOL incompleteDelivery*: is incomplete delivery in order.
- *BOOL in_orderDelivery*: SDUs have to be delivered in order.
- *UNSIGNED INTEGER maxAllowedSduGap*: Maximum allowable gap in SDUs.
- *UNSIGNED INTEGER maxDelay*: in milliseconds.
- *UNSIGNED INTEGER maxJitter*: in milliseconds.

A.2.29. RANGE_OF

Type used to represent a range of integers. It is represented as a type SEQUENCE and contains the following fields:

- INTEGER min: Minimum value of the range.
- INTEGER max: Maximum value of the range.

A.2.30. RESOURCE_ALLOCATOR_STATE

Type used to represent the state of the resource allocator.

- *Section A.2.25, "POLICY_CONFIG" resourceAllocationPolicy*: the configuration of the resource allocation policy (covers dimensioning and allocation of RMT queues, actions to be taken when queues grow too much and distributed resource allocation collaborating with peer IPC Processes).
- *Section A.2.25, "POLICY_CONFIG" routingPolicy*: the configuration of routing strategy to generate the next hop table.
- *Section A.2.25, "POLICY_CONFIG" pduForwardingTableGeneratorPolicy*: the configuration of the PDU Forwarding Table Generator, to generate the PDU Forwarding Table with input from routing and other sources of information.

A.2.31. RIB_DAEMON_STATE

Type used to store the state of the RIB Daemon.

- *Section A.2.25, "POLICY_CONFIG" updatePolicy*: Configuration of the RIB update policy.
- *Section A.2.25, "POLICY_CONFIG" replicationPolicy*: Configuration of the replication policy.
- *Section A.2.25, "POLICY_CONFIG" logging_policy*: Configuration of the logging policy.
- *Section A.2.25, "POLICY_CONFIG" subscription_policy*: Configuration of the subscription policy.
- *Section A.2.25, "POLICY_CONFIG" access_control_policy*: Configuration of the RIB access control policy.

A.2.32. RMTN1Flow_STATE

Type used to represent the state of an RMT n-1 flow.

- *UNSIGNED INTEGER portId* : the portId of the N-1 flow used by the RMT.
- *BOOL stopped* :0 if stopped: 1 if started.

A.2.33. RMT_Q_PAIR_STATE

Type used to represent the state of an RMT queue pair.

- *UNSIGNED INTEGER qosId*: the qosId of the PDUs that will be processed in this queue (0 if traffic of the N-1 port is not distinguished by QoS-id).
- *UNSIGNED INTEGER connectionId*: the connectionId of the PDUs that will be processed in this queue (0 if queues are not allocated on a per-connection-id basis -equivalent of virtual circuits-).
- *UNSIGNED INTEGER inQCapBytes*: capacity of the incoming queue in bytes.
- *UNSIGNED INTEGER inQSizeBytes*: occupation of the incoming queue in bytes.
- *UNSIGNED INTEGER inQSizePDUs*: occupation of the incoming queue in PDUs.
- *UNSIGNED INTEGER inQDroppedPDUs*: number of incoming PDUs dropped.
- *UNSIGNED INTEGER inQProcessedPDUs*: number of incoming PDUs processed.
- *UNSIGNED INTEGER outQCapBytes*: capacity of the outgoing queue in bytes.
- *UNSIGNED INTEGER outQSizeBytes*: occupation of the outgoing queue in bytes.
- *UNSIGNED INTEGER outQSizePDUs*: occupation of the outgoing queue in PDUs.
- *UNSIGNED INTEGER outQDroppedPDUs*: number of outgoing PDUs dropped.
- *UNSIGNED INTEGER outQProcessedPDUs*: number of outgoing PDUs processed.

A.2.34. RMT_STATE

Type used to represent the state of the RMT.

- *Section A.2.25, "POLICY_CONFIG" rmtQMonitorPolicy*: the configuration of the RMT queue monitor policy.
- *Section A.2.25, "POLICY_CONFIG" rmtSchedulingPolicy*: the configuration of the RMT Scheduling Policy.
- *Section A.2.25, "POLICY_CONFIG" rmtMaxQPolicy*: the configuration of the RMT Max Queue Policy.

A.2.35. SDU_PROTECTION_POLICY_SET_CONFIG

Type used to store the configurations of a SDU protection policy. It is represented as a type SEQUENCE and contains the following fields:

- *STRING name*: Name of the SDU Protection policy.
- *Section A.2.25, "POLICY_CONFIG" encryptionPolicy*: Configuration of the encryption policy.
- *Section A.2.25, "POLICY_CONFIG" compressionPolicy*: Configuration of the compression policy.
- *Section A.2.25, "POLICY_CONFIG" error_checkPolicy*: Configuration of the error check policy.
- *Section A.2.25, "POLICY_CONFIG" ttlPolicy*: Configuration of the time to live policy.

A.2.36. SDU_DELIMITING_POLICY_SET_CONFIG

Type used to represent the policies used in the SDU Delimiting module.

- *Section A.2.25, "POLICY_CONFIG" fragmentationPolicy*: configuration of the fragmentation policy.
- *Section A.2.25, "POLICY_CONFIG" concatenationPolicy*: configuration of the concatenation policy.
- *Section A.2.25, "POLICY_CONFIG" reassemblyAndSeparationPolicy*: configuration of the reassembly and separation policy.

A.2.37. SDU_DELIMITING_POLICY_SET_STATE

Type used to represent the state of the SDU Delimiting policy.

- *STRING name*: uniquely identifies the SDU Protection policy set within the IPCP.
- *SET_OF UNSIGNED INTEGER flows*: the port-ids of the N Flows where these SDU delimiting policies are currently applied.
- *Section A.2.36, "SDU_DELIMITING_POLICY_SET_CONFIG" policiesConfig*: the configuration of the policies in the SDU Delimiting policy set

A.2.38. SDU_PROTECTION_POLICY_SET_STATE

Type used to store the configurations of a SDU protection policy. It is represented as a type SEQUENCE and contains the following fields:

- *SEQUENCE_OF UNSIGNED INTEGER underlyingFlows*: the port-ids of the N-1 Flows there these SDU protection policies are currently applied
- *Section A.2.35, "SDU_PROTECTION_POLICY_SET_CONFIG" policySetConfig*: Configuration of the SDU Protection policy set.

A.2.39. SEC_MAN_STATE

Type used to represent the state of the security manager.

- *Section A.2.25, "POLICY_CONFIG" auditingPolicy*: the configuration of the auditing policy.
- *Section A.2.25, "POLICY_CONFIG" credentialManagementPolicy*: the configuration of the credential management policy.generator.

A.2.40. UNDERLAYING_FLOW_DESCRIPTION

Type used to represent one flow.

- *UNSIGNED INTEGER portId*: the portId of the N-1 flow.
- *Section A.2.15, "FLOW_REQUEST"*: application information about the flow.

A.2.41. UNDERLAYING_FLOW_REQUEST

Type used to represent one flow request.

- *Section A.2.2, “APP_NAMING_INFO” localAppName*: the local application entity that is using the N-1 flow.
- *Section A.2.2, “APP_NAMING_INFO” remoteAppName*: the remote application entity that is using the N-1 flow.
- *Section A.2.14, “FLOW_PROPERTIES” flowProperties*: the characteristics of the N-1 flow (loss, delay, reliability, in order-delivery of SDUs, etc).

B. Managed Object Classes

B.1. ApplicationProcess MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
  - **processName**, STRING, Name of the process.
  - **processInstance**, UNSIGNED INTEGER, Instance of the process.
  - **synonymList**, SET_OF STRING, A set of synonyms for the process whose scope is restricted to the DAF it is a member of and may be structured to facilitate its use with in the DAF.
- **ACTIONS**
  - Section B.1.1, "Read ACTION", Section B.1.2, "Cancel_read ACTION", Section B.1.3, "Write ACTION".
- **NOTIFICATIONS** -
- **NAME BINDINGS** DAF, ProcessingSystem.

B.1.1. Read ACTION

- **BEHAVIOUR** Return the object value in a READ_R message.
- **INPUT PARAMETERS** -
- **OUTPUT PARAMETERS** -

B.1.2. Cancel_read ACTION

- **BEHAVIOUR** Cancel any pending read operations.
- **INPUT PARAMETERS** -
- **OUTPUT PARAMETERS** -

B.1.3. Write ACTION

- **BEHAVIOUR** If attributes are present set the attributes. If no attributes are present, replace the whole object.

- **INPUT PARAMETERS** -
- **OUTPUT PARAMETERS**
    - result, integer, 0 if the attributes are set or if the object is replaced, -1 if not.

B.1.4. DAF NAME BINDING

- **CONTAINER CLASS** DAF.
- **CONTAINED CLASS** ApplicationProcess.
- **NAMED WITH ATTRIBUTE** processName.
- **CREATE** Create.
- **DELETE** Delete.

B.1.5. ProcessingSystem NAME BINDING

- **CONTAINER CLASS** ProcessingSystem.
- **CONTAINED CLASS** ApplicationProcess.
- **NAMED WITH ATTRIBUTE** processName.
- **CREATE** Create.
- **DELETE** Delete.

Create ACTION

- **BEHAVIOUR** Create the Application Process.
- **INPUT PARAMETERS**
    - **processName**, STRING, Name of the process.
    - **synonymList**, SET_OF STRING, A set of synonyms for the process whose scope is restricted to the DAF it is a member of and may be structured to facilitate its use with in the DAF.
- **OUTPUT PARAMETERS**
    - **result**, INTEGER, 0 if enrollment is completed and the neighbor is created, -1 if not.
    - **resultReason**, STRING, in case there is an error instantiating the application process, a description of the error may be provided.

Delete ACTION

- **BEHAVIOUR** Kills the ApplicationProcess.
- **INPUT PARAMETERS** -
- **OUTPUT PARAMETERS**
  - **result**, INTEGER, 0 if the ApplicationProcess is destroyed, -1 if not.
  - **resultReason**, STRING, in case there is an error deleting the application process, a description of the error may be provided.

B.2. ComputingSystem MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class cannot be remotely created or deleted, since it represents the root of the containment subtree of a particular computing system. It is the root object of the Management Agent's RIB.
- **ATTRIBUTES**
  - **computingSystemId**, UNSIGNED INTEGER, uniquely identifies the computing system within the Management Domain.
- **ACTIONS** - .
- **NOTIFICATIONS** - .
- **NAME BINDINGS** Root*.

B.2.1. Root NAME BINDING

- **CONTAINER CLASS** Root.
- **CONTAINED CLASS** ComputingSystem.
- **NAMED WITH ATTRIBUTE computingSystemId**.
- **CREATE** This object is automatically created upon Management Agent startup.
- **DELETE** This object cannot be deleted.

B.3. Connection MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
  - **srcCepId**, UNSIGNED INTEGER, the source connection-endpoint id.
  - **srcAddress**, UNSIGNED INTEGER, the address of the source IPC Process.
  - **destCepId**, UNSIGNED INTEGER, the destination connection-endpoint id.
  - **destAddress**, UNSIGNED INTEGER, the address of the destination IPC Process.
  - **qosId**, UNSIGNED INTEGER, the id of the QoS cube where the flow supported by this connection belongs.
  - **portId**, UNSIGNED INTEGER, the id of the flow supported by this connection.
  - **dtcpPresent**, BOOL, indicates if DTCP is active for this connection.
  - **initialATimer**, UNSIGNED INTEGER, indicates the initial value of the A-timer.
  - **seqNumRollOverThres**, UNSIGNED INTEGER, the sequence number rollover threshold.
  - **initSeqNumPolicy**, POLICY_CONFIG, configuration of the initial sequence number policy.
  - **sdrTimerInacPolicy**, POLICY_CONFIG, configuration of the sender timer inactivity policy.
  - **rcvrTimerInacPolicy**, POLICY_CONFIG, configuration of the receiver timer inactivity policy.
  - **mbTransmited**, UNSIGNED REAL, number of megabytes transmitted.
  - **mbReceived**, , UNSIGNED REAL, number of megabytes received.
- **ACTIONS**
  - READ.
- **NOTIFICATIONS**

- CreateConnection.
- DeleteConnection.
- MBUsedThreshold.

- **NAME BINDINGS**
  - Connections.

B.3.1. READ ACTION

- **BEHAVIOUR** Return the Connection object in a READ_R message.
- **OUTPUT OBJECT VALUE**
  - connState, CONNECTION_STATE, encapsulates the data that describes the state of the EFCP Connection

B.3.2. CreateConnection NOTIFICATION

- **BEHAVIOUR** Emitted when a new connection is created.
- **OBJECT VALUE**
  - connState, CONNECTION_STATE, encapsulates the data that describes the state of the EFCP Connection
- **REGISTERED AS** CreateObjectNotification

B.3.3. DeleteConnection NOTIFICATION

- **BEHAVIOUR** Emitted at connection destruction.
- **OBJECT VALUE**
  - **srcCepId**, UNSIGNED INTEGER, the source connection-endpoint id.
- **REGISTERED AS** DeleteObjectNotification

B.3.4. MBUsedThreshold NOTIFICATION

- **BEHAVIOUR** Emitted when the addition of the mbTransmitted and mbReceived exceeds the given threshold. When exceeded, it will reset the counter and start counting again.
- **PARAMETERS**

  - threshold, UNSIGNED REAL, threshold to exceed before launching the notification.
- **OBJECT VALUE**
  - processName, STRING, Name of the process.
  - srcCepId, UNSIGNED INTEGER, the source connection-endpoint id.
  - mbTransmited, UNSIGNED REAL, number of megabytes transmitted.
  - mbReceived, UNSIGNED REAL, number of megabytes received.
- **REGISTERED AS** ChangeAttributeNotification

B.3.5. Connections NAME BINDING

- **CONTAINER CLASS** Connections.
- **CONTAINED CLASS** Connection.
- **NAMED WITH ATTRIBUTE** srcCepId.
- **CREATE** CREATE.
- **DELETE** DELETE.

CREATE ACTION

- **BEHAVIOUR**
  1. Create a Flow RIB object. Causes the Flow Allocator to trigger the flow allocation procedure. Normally created as a result of flows requested by applications or other IPCPs via the RINA IPC API, but also available as a management operation via the management agent.
- **INPUT OBJECT VALUE**
  - connRequest, CONNECTION_REQUEST, the data related to the connection request.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

DELETE ACTION

- **BEHAVIOUR**

1. Causes the targeted N connection to be deleted.

- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

B.4. Connections MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** - .
- **NOTIFICATIONS** - .
- **NAME BINDINGS**
  - DataTransfer.

B.4.1. DataTransfer NAME BINDING

- **CONTAINER CLASS** DataTransfer.
- **CONTAINED CLASS** Connections.
- **NAMED WITH ATTRIBUTE** Static, always "**connections**".

B.5. DAF MANAGED OBJECT CLASS

- **DERIVED FROM** Top .
- **BEHAVIOUR** This class represents a Distributed Application Facility.
- **ATTRIBUTES**
  - **dafID**, UNSIGNED INTEGER, uniquely identifies the DAF within the Management Domain.
- **ACTIONS**
  - READ, CANCEL_READ, WRITE.
- **NOTIFICATIONS** - .

- **NAME BINDINGS**
    - Root

B.5.1. Read ACTION

- **BEHAVIOUR** Return the object in a READ_R message.
- **OUTPUT OBJECT VALUE dafID**, UNSIGNED INTEGER, uniquely identifies the DAF within the Management Domain.

B.5.2. Root NAME BINDING

- **CONTAINER CLASS** Root.
- **CONTAINED CLASS** DAF.
- **NAMED WITH ATTRIBUTE** dafID.

B.6. DataTransfer MANAGED OBJECT CLASS

- **DERIVED FROM** ApplicationEntity.
- **ATTRIBUTES**
    - **addressLength**, UNSIGNED INTEGER, length of the address field in the DTP header (in bytes).
    - **qosIdLength**, UNSIGNED INTEGER, length of the qos-id field in the DTP header (in bytes).
    - **portIdLength**, UNSIGNED INTEGER, length of the port-id field in the DTP header (in bytes).
    - **cepIdLength**, UNSIGNED INTEGER, length of the cep-id field in the DTP header (in bytes).
    - **seqNumLength**, UNSIGNED INTEGER, length of the sequence number field in the DTP header (in bytes).
    - **lengthLength**, UNSIGNED INTEGER, length of the length field in the DTP header (in bytes).
    - **maxPDUSize**, UNSIGNED INTEGER, the maximum size of a DTP PDU (in bytes).

  - **maxSDUSize**, UNSIGNED INTEGER, the maximum size of an SDU written to the flow supported by this connection (in bytes).
  - **unknownFlowPolicy**, POLICY_CONFIG, configuration of the unknown flow policy.
  - **pduForwardingTableGeneratorPolicy**, POLICY_CONFIG, the configuration of the PDU Forwarding Table Generator, to generate the PDU Forwarding Table with input from routing protocol and other sources of information.
- **ACTIONS**
  - READ.
- **NOTIFICATIONS** - .
- **NAME BINDINGS**
  - IPCProcess.

B.6.1. READ ACTION

- **BEHAVIOUR** Return the DataTransfer attributes in a READ_R message.
- **OUTPUT OBJECT VALUE**
  - dtState, DATA_TRANSFER_STATE the state of the Data Transfer Task.
- **RESULT** 0 if successful, a negative integer indicating an error condition otherwise.

B.6.2. IPCProcess NAME BINDING

- **CONTAINER CLASS** IPCProcess.
- **CONTAINED CLASS** DataTransfer.
- **NAMED WITH ATTRIBUTE** Static, always "**dt**".

B.7. DestinationReports MANAGED OBJECT CLASS

- **DERIVED FROM** Top.

- **ATTRIBUTES**
  - destinationName, STRING, name of the subscriber.
  - destinationInstance, UNSIGNED INTEGER, instance of the subscriber.
- **ACTIONS**
  - READ, CANCEL_READ.
- **NOTIFICATIONS** - .
- **NAME BINDINGS**
  - LatestReports.

B.7.1. READ ACTION

- **BEHAVIOUR** Return the DestinationReport.
- **OUTPUT OBJECT VALUE**
  - destinationReportsConfig, SEQUENCE, configurations of the DestinationReports
    - destinationName, STRING, name of the subscriber.
    - destinationInstance, UNSIGNED INTEGER, instance of the subscriber.
- **RESULT** 0 if success, -1 if not.

B.7.2. CANCEL_READ ACTION

- **BEHAVIOUR** cancel any pending read operations.
- **RESULT** the number of pending read operations cancelled, -1 otherwise.

B.7.3. LatestReports NAME BINDING

- **CONTAINER CLASS** LatestReports.
- **CONTAINED CLASS** DestinationReports.
- **NAMED WITH ATTRIBUTE** destinationAddress.

B.8. DIF MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class represents a Distributed Interprocess Facility.
- **ATTRIBUTES**
    - **difID**, UNSIGNED INTEGER, uniquely identifies the DIF within the Management Domain.
- **ACTIONS**
    - READ, CANCEL_READ, WRITE.
- **NOTIFICATIONS** - .
- **NAME BINDINGS**
    - Root

B.8.1. Read ACTION

- **BEHAVIOUR** Return the object in a READ_R message.
- **OUTPUT OBJECT VALUE difID**, UNSIGNED INTEGER, uniquely identifies the DIF within the Management Domain.

B.8.2. Root NAME BINDING

- **CONTAINER CLASS** Root.
- **CONTAINED CLASS** DIF.
- **NAMED WITH ATTRIBUTE** difID.

B.9. DIFManagement MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR**. This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** -.
- **NOTIFICATIONS**

- **NAME BINDINGS**
    - IPCProcess.
    - ManagementAgent.

B.9.1. IPCProcess NAME BINDING

- **CONTAINER CLASS** IPCProcess.
- **CONTAINED CLASS** DIFManagement.
- **NAMED WITH ATTRIBUTE** Static name, always "**difmanagement**".

B.9.2. ManagementAgent NAME BINDING

- **CONTAINER CLASS** ManagementAgent.
- **CONTAINED CLASS** DIFManagement.
- **NAMED WITH ATTRIBUTE** Static name, always "**difmanagement**".

B.10. DIFProperties MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
    - difPropertiesID, UNSIGNED INTEGER, uniquely identifies the DIF properties within the Management Domain.
    - propertyName, STRING, name of the property.
    - values, SET_OF PAIR, stores the pair configuration, value.
- **ACTIONS** READ, CANCEL_READ.
- **NAME BINDINGS** UnderlayingDIF.

B.10.1. READ ACTION

- **BEHAVIOUR** Return the DIFProperties object in a READ_R message.
- **OUTPUT OBJECT VALUE**
    - difPropertiesConfig, SEQUENCE, attributes of the DIFProperties.

- difPropertiesID, UNSIGNED INTEGER, uniquely identifies the AvailableDIF within the Management Domain.
- propertyName, STRING, name of the property.
- values, SET_OF PAIR, stores the pair configuration, value

- **RESULT** 0 if success, -1 if not.

B.10.2. CANCEL_READ ACTION

- **BEHAVIOUR** Cancel any pending read operations.
- **RESULT** the number of pending read operations canceled, -1 if error.

B.10.3. UnderlayingDIF NAME BINDING

- **CONTAINER CLASS** UnderlayingDIF.
- **CONTAINED CLASS** DIFProperties.
- **NAMED WITH ATTRIBUTE** difPropertiesID.

B.11. DirectoryForwardingTable MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** - .
- **NOTIFICATIONS** -.
- **NAME BINDINGS**
  - NamespaceManager.

B.11.1. NamespaceManager NAME BINDING

- **CONTAINER CLASS** NamespaceManager.
- **CONTAINED CLASS** DirectoryForwardingTable.

- **NAMED WITH ATTRIBUTE** Static, always "**dft**".

B.12. DirectoryForwardingTableEntry MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
  - **key**, UNSIGNED INTEGER, unique key of this entry in the PDU Forwarding Table.
  - **name**, APP_NAMING_INFO, the destination application process naming information.
  - **address**, UNSIGNED INTEGER, the address of the next IPC Process where the Flow Request should be forwarded.
- **ACTIONS** READ.
- **NOTIFICATIONS**
- **NAME BINDINGS**
  - DirectoryForwardingTable.

B.12.1. READ ACTION

- **BEHAVIOUR** Return the attributes DirectoryForwardingTableEntry object in a READ_R message.
- **OUTPUT OBJECT VALUE**
  - dftEntry, DFT_ENTRY, encapsulates the data contained in the Directory Forwarding Table Entry.

B.12.2. CreateDirectoryForwardingTableEntry NOTIFICATION

- **BEHAVIOUR** Launches at new DirectoryForwardingTableEntry object creation.
- **OBJECT VALUE**
  - dftEntry, DFT_ENTRY, the data of the Directory Forwarding Table Entry.

- **REGISTERED AS** CreateObjectNotification

B.12.3. DeleteDirectoryForwardingTableEntry NOTIFICATION

- **BEHAVIOUR** Launches at DirectoryForwardingTableEntry object destruction.
- **OBJECT VALUE**
  - dftEntry, DFT_ENTRY, the data of the Directory Forwarding Table Entry.
- **REGISTERED AS** DeleteObjectNotification

B.12.4. DirectoryForwardingTable NAME BINDING

- **CONTAINER CLASS** DirectoryForwardingTable.
- **CONTAINED CLASS** DirectoryForwardingTableEntry.
- **NAMED WITH ATTRIBUTE** key.

CREATE ACTION

- **BEHAVIOUR**
  1. Add a static entry to the Directory Forwarding Table.
- **INPUT OBJECT VALUE**
  - dftEntry, DFT_ENTRY, the data of the Directory Forwarding Table Entry.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

DELETE ACTION

- **BEHAVIOUR**
  1. Remove an entry from the Directory Forwarding Table.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

B.13. Discriminators MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** - .
- **NOTIFICATIONS** - .
- **NAME BINDINGS**
  - ProcessingSystem.

B.13.1. Discriminators NAME BINDING

- **CONTAINER CLASS** ProcessingSystem.
- **CONTAINED CLASS** Flows.
- **NAMED WITH ATTRIBUTE** Static, always "**discriminators**".

B.14. DTCP MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
  - **flowControl**, UNSIGNED INTEGER, 0 means flow control not in use, 1 means sliding window flow control, 2 means rate-based flow control.
  - **rtxControl**, BOOL, true if retransmission control is enabled, false otherwise.
  - **lostControlPDUPolicy**, POLICY_CONFIG, configuration of the Lost Control PDU Policy.
  - **rttEstimatorPolicy**, POLICY_CONFIG, configuration of the RTT estimator Policy.
  - **maxTimeRetry**, UNSIGNED INTEGER, Max time to attempt the retransmission of a packet, in ms (this is R).
  - **dataRxmsnMax**, UNSIGNED INTEGER, Max number of retransmission attempts.

- **portId**, UNSIGNED INTEGER, the id of the flow supported by this connection.
- **intialRtxTime**, UNSIGNED INTEGER, indicates the time to wait before transmitting a PDU.
- **rtxTimerExpiryPolicy**, POLICY_CONFIG, configuration of the Retransmission Timer Expiry Policy.
- **sdrAckPolicy**, POLICY_CONFIG, configuration of the Sender ACK Policy.
- **rcvingAckListPolicy**, POLICY_CONFIG, configuration of the Receiving ACK List Policy.
- **rcvrAckPolicy**, POLICY_CONFIG, configuration of the Receiver ACK Policy.
- **sendingAckListPolicy**, POLICY_CONFIG, configuration of the Sending ACK List Policy.
- **rcvingAckListPolicy**, POLICY_CONFIG, configuration of the Receiving ACK List Policy.
- **rcvrCtrlAckPolicy**, POLICY_CONFIG, configuration of the Receiver Control Ack Policy.
- **clsdWindowPolicy**, POLICY_CONFIG, configuration of the Closed Window Policy.
- **flowCtrlOverrunPolicy**, POLICY_CONFIG, configuration of the Flow Control Overrun Policy.
- **rcvingAckListPolicy**, POLICY_CONFIG, configuration of the Receiving ACK List Policy.

- **ACTIONS**
  - READ.
- **NOTIFICATIONS** - .
- **NAME BINDINGS**
  - Connection.

B.14.1. READ ACTION

- **BEHAVIOUR** Return the DTCP object in a READ_R message.

- **OUTPUT OBJECT VALUE**
    - dtcpState, DTCP_STATE, encapsulates the data that describes the state of the DTCP part of the Connection

B.14.2. Connection NAME BINDING

- **CONTAINER CLASS** Connection.
- **CONTAINED CLASS** DTCP.
- **NAMED WITH ATTRIBUTE** Static, always "**dtcp**".

B.15. DTCPStateVector MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
    - TODO: Unspecified.
- **ACTIONS**
    - READ.
- **NOTIFICATIONS** -
- **NAME BINDINGS**
    - DTCP.

B.15.1. READ ACTION

- **BEHAVIOUR** Return the DTCPStateVector object in a READ_R message.
- **OUTPUT OBJECT VALUE**
    - dtcpState, DTCP_SV_STATE, encapsulates the data that describes the DTCP State Vector

B.15.2. DTCP NAME BINDING

- **CONTAINER CLASS** DTCP.
- **CONTAINED CLASS** DTCPStateVector.

- **NAMED WITH ATTRIBUTE** Static, always "**dtcpsv**".

B.16. DTPStateVector MANAGED OBJECT CLASS

- DERIVED FROM Top.
- ATTRIBUTES
    - **droppedPDUs**, UNSIGNED INT, the number of PDUs that have been dropped
    - **inPDUs**, UNSIGNED INT, the number of PDUs received
    - **inBytes**, UNSIGNED INT, the number of bytes received
    - **inBps**, UNSIGNED INT, incoming Bytes per second (average over a certain time period)
    - **outPDUs**, UNSIGNED INT, the number of PDUs sent
    - **outBps**, UNSIGNED INT, outgoing Bytes per second (average over a certain time period)
    - **outBytes**, UNSIGNED INT, the number of bytes sent
    - **maxSeqNumRcvd**, UNSIGNED INT, highest sequence number received
    - **lastSeqNumSent**, UNSIGNED INT, the sequence number of the latest PDU that has been sent
- ACTIONS
    - READ.
- NOTIFICATIONS
- NAME BINDINGS
    - ConnectionNameBinding.

B.16.1. READ ACTION

- BEHAVIOUR Return the DTPStateVector object in a READ_R message.
- OUTPUT OBJECT VALUE
    - dtcpState, DTP_SV_STATE, encapsulates the data that describes the in the DTP State Vector

B.16.2. ConnectionNameBinding NAME BINDING

- CONTAINER CLASS Connection.
- CONTAINED CLASS DTPStateVector.
- NAMED WITH ATTRIBUTE Static, always "**dtpsv**".

B.17. Enrollment MANAGED OBJECT CLASS

- **DERIVED FROM** ApplicationEntity.
- **ATTRIBUTES**
  - **enrollmentPolicy**, POLICY_CONFIG, the configuration of the enrollment policy.
  - **newMemberAccessControlPolicy**, POLICY_CONFIG, the configuration of the new member access control policy.
- **ACTIONS**
  - READ.
- **NOTIFICATIONS**
- **NAME BINDINGS**
  - DIFManagement.

B.17.1. READ ACTION

- **BEHAVIOUR** Return the Enrollment attributes in a READ_R message.
- **OUTPUT OBJECT VALUE**
  - enrollmentState, ENROLLMENT_STATE the state of the Enrollment Task.
- **RESULT** 0 if successful, a negative integer indicating an error condition otherwise.

B.17.2. DIFManagement NAME BINDING

- **CONTAINER CLASS** DIFManagement.

- **CONTAINED CLASS** Enrollment.
- **NAMED WITH ATTRIBUTE** Static, always "**enrollment**".

B.18. Flow MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
  - **localPortId**, UNSIGNED INTEGER, the portId of the flow.
  - **localAppName**, APP_NAMING_INFO, the local application entity that is using the N flow.
  - **remotePortId**, UNSIGNED INTEGER, the portId of the flow at the remote IPC Process.
  - **remoteAppName**, APP_NAMING_INFO, the remote application entity that is using the N flow.
  - **flowProperties**, FLOW_PROPERTIES, the characteristics of the N flow (loss, delay, reliability, in order-delivery of SDUs, etc).
  - **state**, UNSIGNED INTEGER, 0 allocation in progress, 1 allocated, 2, deallocation in progress, 3 deallocated.
  - **maxCreateFlowRetries**, UNSIGNED INTEGER, the maximum number of attempts for allocating the flows.
  - **createFlowRetries**, UNSIGNED INTEGER, the current number of attempts for allocating this flow.
  - **reservedCepIds**, UNSIGNED INTEGER, the connection-endpoint ids reserved for the connections that will support this flow in this IPC Process.
  - **currentCepId**, UNSIGNED INTEGER, the connection-endpoint currently used by the connection supporting this flow.
- **ACTIONS**
  - READ.
- **NOTIFICATIONS**
- **NAME BINDINGS**
  - FlowsNameBinding.

B.18.1. READ ACTION

- **BEHAVIOUR** Return the Flow object in a READ_R message.
- **OUTPUT OBJECT VALUE**
    - flowState, FLOW_STATE, encapsulates the data that describes the state of the N-flow

B.18.2. CreateFlow NOTIFICATION

- **BEHAVIOUR** Launches at new Flow object creation.
- **OBJECT VALUE**
    - flowState, FLOW_STATE, encapsulates the data that describes the state of the N-flow
- **REGISTERED AS** CreateObjectNotification

B.18.3. DeleteFlow NOTIFICATION

- **BEHAVIOUR** Launches at Flow object destruction.
- **OBJECT VALUE**
    - localPortId, UNSIGNED INTEGER, the portId of the flow.
- **REGISTERED AS** DeleteObjectNotification

B.18.4. FlowsNameBinding NAME BINDING

- **CONTAINER CLASS** Flows.
- **CONTAINED CLASS** Flow.
- **NAMED WITH ATTRIBUTE** localPortId.
- **CREATE** CREATE.
- **DELETE** DELETE.

CREATE ACTION

- **BEHAVIOUR**
    1. Create a Flow RIB object. Causes the Flow Allocator to trigger the flow allocation procedure. Normally requested by applications via

the RINA IPC API, but also available as a management operation via the management agent.

- **INPUT OBJECT VALUE**
  - flowRequest, FLOW_REQUEST, the data related to the flow request (destination application, flow characteristics).
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

DELETE ACTION

- **BEHAVIOUR**
  1. Causes the targeted N flow to be deallocated.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

B.19. FlowAllocator MANAGED OBJECT CLASS

- **DERIVED FROM** ApplicationEntity.
- **ATTRIBUTES**
  - **newFlowRequestPolicy**, POLICY_CONFIG, the configuration of the New Flow Request policy.
  - **seqRollOverPolicy**, POLICY_CONFIG, the configuration of the sequence number rollover policy.
  - **allocateNotifyPolicy**, POLICY_CONFIG, the configuration of the allocate notify policy.
  - **allocateRetryPolicy**, POLICY_CONFIG, the configuration of the allocate retry policy.
  - **newFlowAccessControlPolicy**, POLICY_CONFIG, the configuration of the new flow access control policy (for incoming flows).
  - **inFlowReq**, UNSIGNED INTEGER, the total number of incoming flow requests.
  - **inFlowReqRej**, UNSIGNED INTEGER, the number of incoming flow requests that have been rejected.

  - **outFlowRes**, UNSIGNED INTEGER, the total number of outgoing flow requests.
  - **outFlowReqRej**, UNSIGNED INTEGER, the number of outgoing flow requests that have been rejected.
- ACTIONS
  - READ.
- **NOTIFICATIONS** - .
- **NAME BINDINGS**
  - IPCProcess.

B.19.1. READ ACTION

- **BEHAVIOUR** Return the Flow Allocator attributes in a READ_R message.
- **OUTPUT OBJECT VALUE**
  - flowAllocatorState, FLOW_ALLOCATOR_STATE the state of the Flow Allocator.
- **RESULT** 0 if successful, a negative integer indicating an error condition otherwise.

B.19.2. IPCProcess NAME BINDING

- **CONTAINER CLASS** IPCProcess.
- **CONTAINED CLASS** FlowAllocator.
- **NAMED WITH ATTRIBUTE** Static, always "**fa**".

B.20. Flows MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** - .

- **NOTIFICATIONS** - .
- **NAME BINDINGS**
  - FlowAllocator.

B.20.1. FlowAllocator NAME BINDING

- **CONTAINER CLASS** FlowAllocator.
- **CONTAINED CLASS** Flows.
- **NAMED WITH ATTRIBUTE** Static, always "**flows**".

B.21. ForwardingDiscriminator MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
  - forwardingDiscriminatorID, UNSIGNED INTEGER, uniquely identifies the ForwardingDiscriminator within the Management Domain.
  - destinationName, STRING, name of the subscriber.
  - destinationInstance, UNSIGNED INTEGER, instance of the subscriber.
  - reportArchivePolicy, POLICY_CONFIG, represents a policy that decide how to save the reports.
  - filteringPolicy, POLICY_CONFIG, represents a policy that decide how to save the reports.
  - lostNotifications, UNSIGNED INTEGER, count of the number of discarded notifications since the one previously sent.
- **ACTIONS**
  - READ, CANCEL_READ, WRITE. START. STOP.
- **NOTIFICATIONS**
  - CreateForwardingDiscriminator.
  - DeleteForwardingDiscriminator.
- **NAME BINDINGS**

- Discriminator.

B.21.1. READ ACTION

- **BEHAVIOUR** If the message contain specific attributes, return only their value. Otherwise, return the ForwardingDiscriminator attributes in a READ_R message.
- **OUTPUT OBJECT VALUE**
  - forwardingDiscriminatorConfig, SEQUENCE
    - forwardingDiscriminatorID, UNSIGNED INTEGER, uniquely identifies the ForwardingDiscriminator within the Management Domain.
    - reportArchivePolicy, POLICY_CONFIG, represents a policy that decide how to save the reports.
    - filteringPolicy, POLICY_CONFIG, represents a policy that decide how to save the reports.
    - lostNotifications, UNSIGNED INTEGER, count of the number of discarded notifications since the one previously sent.
- **RESULT** 0 if success, -1 if not.

B.21.2. CANCEL_READ ACTION

- **BEHAVIOUR** cancel any pending read operations.
- **RESULT** the number of pending read operations canceled, -1 otherwise.

B.21.3. WRITE ACTION

- **BEHAVIOUR**
  1. Overwrite the attributes of the ForwardingDiscriminator object.
- **INPUT OBJECT VALUE**
  - discriminatorPolicy, DISCRIMINATOR_POLICY_CONFIG, represents a policy that configures the catching, filtering and reporting of the notifications.
- **RESULT** 0 if the policy has been overwrited, -1 otherwise.

B.21.4. START ACTION

- **BEHAVIOUR** Start the discriminator if it is stoped. Otherwise do nothing.
- **RESULT** 0 if the discriminator was stoped and it has been started, -1 otherwise.

B.21.5. STOP ACTION

- **BEHAVIOUR** Stop the discriminator if it is started. Otherwise do nothing.
- **RESULT** 0 if the discriminator was started and it has been stoped, -1 otherwise.

B.21.6. CreateForwardingDiscriminator NOTIFICATION

- **BEHAVIOUR** Launches at new ForwardingDiscriminator object creation.
- **OBJECT VALUE**
  - forwardingDiscriminatorConfig, sequence
    - forwardingDiscriminatorID, unsigned integer, uniquely identifies the ForwardingDiscriminator within the Management Domain.
    - discriminatorPolicy, sequence, represents a policy that configures the catching, filtering and reporting of the notifications.
- **REGISTERED AS** CreateObjectNotification

B.21.7. DeleteForwardingDiscriminator NOTIFICATION

- **BEHAVIOUR** Launches at ForwardingDiscriminator object destruction.
- **OBJECT VALUE**
  - forwardingDiscriminatorID, unsigned integer, uniquely identifies the ForwardingDiscriminator within the Management Domain.
- **REGISTERED AS** DeleteObjectNotification

B.21.8. Discriminators NAME BINDING

- **CONTAINER** CLASS Discriminators.

- **CONTAINED CLASS** ForwardingDiscriminator.
- **NAMED WITH ATTRIBUTE** forwardingDiscriminatorID.
- **CREATE** CREATE.
- **DELETE** DELETE.

CREATE ACTION

- **BEHAVIOUR**
  1. Create the ForwardingDiscriminator in the RIB.
  2. Subscribe the ForwardingDiscriminator in the given notifications and add the forwardingDiscriminatorID to every list of discriminators of the object where it is subscribed.
- **INPUT OBJECT VALUE**
  - discriminatorPolicy, DISCRIMINATOR_POLICY_CONFIG, represents a policy that configures the catching, filtering and reporting of the notifications.
- **RESULT** 0 if ForwardingDiscriminator object has been created, -1 otherwise.

DELETE ACTION

- **BEHAVIOUR**
  1. Unsubscribe to any notification it was subscribed to
  2. Remove any DiscriminatorReport under it.
  3. Remove the RIB object.
- **RESULT** 0 if the ForwardingDiscriminator object is erased, -1 otherwise.

B.22. Hardware MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** - .

- **NOTIFICATIONS**
- **NAME BINDINGS**
    - ProcessingSystem.

B.22.1. ProcessingSystem NAME BINDING

- **CONTAINER CLASS** ProcessingSystem.
- **CONTAINED CLASS** Hardware.
- **NAMED WITH ATTRIBUTE** Static name, always "**hardware**"

B.23. IPCManagement_ma MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** - .
- **NOTIFICATIONS** -
- **NAME BINDINGS**
    - ManagementAgent.

B.23.1. ManagementAgent NAME BINDING

- **CONTAINER CLASS** ManagementAgent.
- **CONTAINED CLASS** IPCManagement.
- **NAMED WITH ATTRIBUTE** Static name, always "**ipcmanagement_ma**".

B.24. IPCManagement MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class is just a container with no attributes.

- **ATTRIBUTES** - .
- **ACTIONS** - .
- **NOTIFICATIONS** - .
- **NAME BINDINGS**
    - IPCProcess.
    - ManagementAgent.

B.24.1. IPCProcess NAME BINDING

- **CONTAINER CLASS** IPCProcess.
- **CONTAINED CLASS** IPCManagement.
- **NAMED WITH ATTRIBUTE** Static name, always "**ipcmanagement**".

B.24.2. ManagementAgent NAME BINDING

- **CONTAINER CLASS** ManagementAgent.
- **CONTAINED CLASS** IPCManagement.
- **NAMED WITH ATTRIBUTE** Static name, always "**ipcmanagement**".

B.25. IPCProcess MANAGED OBJECT CLASS

- **DERIVED FROM** ApplicationProcess.
- **ATTRIBUTES**
    - **processId**, UNSIGNED INTEGER, unique identifier of an IPC process within a system.
- **ACTIONS**
    - READ.
    - CANCEL_READ.
- **NOTIFICATIONS**
    - CreateIPCProcess.
    - DeleteIPCProcess.
    - ErrorIPCProcess.

- **NAME BINDINGS**
  - DIF.
  - OSApplicationProcess.

B.25.1. READ ACTION

- **BEHAVIOUR** Return the IPC process object in a READ_R message.
- **OUTPUT OBJECT VALUE**
  - Sequence ipcProcessConfig
    - processName, STRING, Name of the process.
    - processInstance, UNSIGNED INTEGER, Instance of the process.
    - synonymList, SET OF STRING, A set of synonyms for the process whose scope is restricted to the DIF it is a member of and may be structured to facilitate its use with in the DIF.
- **RESULT** 0 if success, -1 if not.

B.25.2. CANCEL_READ ACTION

- **BEHAVIOUR** Cancel any pending read operations.
- **RESULT** the number of pending read operations canceled, -1 if error.

B.25.3. CreateIPCProcess NOTIFICATION

- **BEHAVIOUR** Emitted at IPCProcess creation.
- **OBJECT VALUE**
  - createNotification, SEQUENCE, attributes for create IPC process notification
    - processName, STRING, name of the process.
    - processId, UNSIGNED INTEGER, unique identifier of an IPC process within a system.
- **REGISTERED AS** CreateObjectNotification

B.25.4. DeleteIPCProcess NOTIFICATION

- **BEHAVIOUR** Emitted at IPCProcess destruction.

- **OBJECT VALUE**
  - deleteNotification, SEQUENCE, attributes for delete IPC process notification
    - processName, STRING, name of the process.
    - processId, UNSIGNED INTEGER, unique identifier of an IPC process within a system.
- **REGISTERED AS** DeleteObjectNotification

B.25.5. ErrorIPCProcess NOTIFICATION

- **BEHAVIOUR** Emitted when an error occurs in the IPC Process operation.
- **PARAMETERS**
  - error, unsigned integer, code of the error.
  - reason, string, reason of the error.
- **OBJECT VALUE**
  - processName, STRING, name of the process.
  - processId, UNSIGNED INTEGER, unique identifier of an IPC process within a system.
- **REGISTERED AS** ErrorNotification

B.25.6. DIF NAME BINDING

- **CONTAINER CLASS** DIF.
- **CONTAINED CLASS** IPCProcess.
- **NAMED WITH ATTRIBUTE** processName.
- **CREATE** Create.
- **DELETE** Delete.

B.25.7. OSApplicationProcess NAME BINDING

- **CONTAINER CLASS** OSApplicationProcess.
- **CONTAINED CLASS** IPCProcess.

- **NAMED WITH ATTRIBUTE** processId.
- **CREATE** Create.
- **DELETE** Delete.

Create ACTION

- **BEHAVIOUR**
  1. Instantiate the IPC process. After the instantiation of the IPCP the following objects will be created in the containment tree.
     - IPCProcess, IPCManagement, RIBDaemon, ResourceAllocator, DIFManagement, DataTransfer, SDUDelimiting, RelayingAndMultiplexing, FlowAllocator, SDUProtection, IPCResourceManager, UnderlayingDIFs, UnderlayingFlows, UnderlayingRegistrations, NamespaceManagement, SecurityManagement, DirectoryForwardingTable.
  2. Register the IPCP to one or more N-1 DIFs. Optional, only if N-1 DIF Information is present. For each successful registration, the following object will be created or updated in the containment tree.
     - UnderlayingRegistration.
  3. Assign to the provided DIF. Optional, only if DIF assignment and configuration information is present. The following objects will be created in the containment tree.
     - QoSCube
  4. Enroll to the given neighbor. Optional, only if neighbor information is present. The following objects will be created in the containment tree (for each new neighbor)
     - Neighbor.
- **INPUT OBJECT VALUE**
  - **instantiate_dif_data**, CREATE_IPCP_CONFIG, The configuration data to instantiate an IPC Process in a system.
- **RESULT** if successful a positive integer that is the process id of the created IPC process; if not successful a negative integer indicating an error condition.
- **RESULT REASON** Further description of the error (optional).

Delete ACTION

- **BEHAVIOUR** If the hardDelete boolean is false (default)
    1. Unenroll the IPC process from any neighbor it is enrolled to.
    2. Unassign the IPC process from the DIF where it has been assigned (if any).
    3. Unregister the IPC process from the N-1 DIFs where it has been registered (if any).
    4. Kill the IPC process.
    5. After these four steps the IPC Process and all its contained objects will be deleted from the containment tree. If the hardDelete boolean is true
    6. The IPC Process and all its contained objects will be deleted from the containment tree.
    7. N-1, N+1 related IPC processes and neighbors must be able to detect the failure after some time.
- **INPUT OBJECT VALUE**
    - BOOLEAN hardDelete: if true IPCProcess must be automatically killed.
- **RESULT** 0 if successful, a negative integer indicating an error condition if not successful.
- **RESULT REASON** Further description of the error (optional).

B.26. IPCResourceManager MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** - .
- **NOTIFICATIONS** - .
- **NAME BINDINGS**

- IPCManagement.

B.26.1. IPCManagement NAME BINDING

- **CONTAINER CLASS** IPCManagement.
- **CONTAINED CLASS** IPCResourceManager.
- **NAMED WITH ATTRIBUTE** Static, always "**irm**".

B.27. KernelApplicationProcess MANAGED OBJECT CLASS

- **DERIVED FROM** ApplicationProcess.
- **BEHAVIOUR** This class is the Kernel of the ProcessingSystem.
- **ACTIONS** - .
- **NOTIFICATIONS** - .
- **NAME BINDINGS** **Root**.

B.27.1. Root NAME BINDING

- **CONTAINER CLASS** root.
- **CONTAINED CLASS** KernelApplicationProcess.
- **NAMED WITH ATTRIBUTE** Static name, always "**kernelApplicationProcess**".

B.28. LatestReports MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ACTIONS**
  - READ, CANCEL_READ.
- **NOTIFICATIONS** - .
- **NAME BINDINGS**

  - RIBDaemon.

B.28.1. RIBDaemon NAME BINDING

- **CONTAINER CLASS** RIBDaemon.
- **CONTAINED CLASS** LatestReports.
- **NAMED WITH ATTRIBUTE** Static, always "**latestReports**".

B.29. ManagementAgent MANAGED OBJECT CLASS

- **DERIVED FROM** ApplicationProcess.
- **ATTRIBUTES**
  - **managementAgentId**, UNSIGNED INTEGER, unique identifier of a management agent within a system.
- **ACTIONS**
  - READ.
- **NAME BINDINGS**
  - DAF.
  - OSApplicationProcess.

B.29.1. READ ACTION

- **BEHAVIOUR** Return the management agent object in a READ_R message.
- **OUTPUT OBJECT VALUE**
  - **managementAgentId**, UNSIGNED INTEGER, unique identifier of a management agent within a system.
- **RESULT** 0 if success, -1 if not.

B.29.2. CANCEL_READ ACTION

- **BEHAVIOUR** Cancel any pending read operations.
- **RESULT** the number of pending read operations canceled, -1 if error.

B.29.3. DAF NAME BINDING

- **CONTAINER CLASS** DAF.
- **CONTAINED CLASS** ManagementAgent.
- **NAMED WITH ATTRIBUTE** managementAgentId.

B.29.4. OSApplicationProcess NAME BINDING

- **CONTAINER CLASS** OSApplicationProcess.
- **CONTAINED CLASS** ManagementAgent.
- **NAMED WITH ATTRIBUTE** managementAgentId.

B.30. NamespaceManager MANAGED OBJECT CLASS

- **DERIVED FROM** ApplicationEntity.
- **ATTRIBUTES**
  - **addressAssignmentPolicy**, POLICY_CONFIG, the configuration of the address assignment policy.
  - **addressValidationPolicy**, POLICY_CONFIG, the configuration of the new address validation policy.
  - **directoryForwardingTableGeneratorPolicy**, POLICY_CONFIG, the configuration of the directory forwarding table generator.
- **ACTIONS**
  - READ.
- **NOTIFICATIONS** -.
- **NAME BINDINGS**
  - DIFManagement.

B.30.1. READ ACTION

- **BEHAVIOUR** Return the Namespace Manager attributes in a READ_R message.

- **OUTPUT OBJECT VALUE**
    - nsmState, NSM_STATE the state of the Enrollment Task.
- **RESULT** 0 if successful, a negative integer indicating an error condition otherwise.

B.30.2. DIFManagement NAME BINDING

- **CONTAINER CLASS** DIFManagement.
- **CONTAINED CLASS** NamespaceManager.
- **NAMED WITH ATTRIBUTE** Static, always "**nsm**".

B.31. Neighbor MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
    - **processName**, STRING, the application process name of the neighbor IPC Process.
    - **processInstance**, UNSIGNED INTEGER, the application process instance of the neighbor IPC Process.
    - **address**, UNSIGNED INTEGER, the current address of the neighbor IPC Process.
    - **underlyingDIFs**, SET_OF STRING, the names of the N-1 DIFs in common with the neighbor IPC Process.
    - **underlyingFlows**, SET_OF UNSIGNED INTEGER, the port-id of the N-1 flow used to talk to the neighbor.
    - **authenticationPolicy**, POLICY_CONFIG, the configuration of the authentication policy used to authenticate the neighbor IPC Process.
    - **numberOFEnrollmentAttempts**, UNSIGNED INTEGER, the number of enrollment attempts tried with the neighbor (or that the neighbor has tried with this IP Process).
    - **isEnrolled**, BOOL, true if the neighbor is currently enrolled. False otherwise.
- **ACTIONS**

- READ.
- **NOTIFICATIONS**
  - CreateNeighbor.
  - DeleteNeighbor.
  - IncrementNumberOFEnrollmentAttempts.
- **NAME BINDINGS**
  - Neighbors.

B.31.1. READ ACTION

- **BEHAVIOUR** Return the Neighbor object in a READ_R message.
- **OUTPUT OBJECT VALUE**
  - neighborState, NEIGHBOR_STATE, encapsulates the data that describes the neighbor.

B.31.2. CreateNeighbor NOTIFICATION

- **BEHAVIOUR** Emitted when a new Neighbor object is created.
- **OBJECT VALUE**
  - neighborState, NEIGHBOR_STATE, encapsulates the data that describes the neighbor.
- **REGISTERED AS** CreateObjectNotification

B.31.3. DeleteNeighbor NOTIFICATION

- **BEHAVIOUR** Launches at Neighbor object destruction.
- **OBJECT VALUE**
  - deleteNotificationInfo, SEQUENCE, information provided by the delate Neighbor notification.
    - processName, STRING, the application process name of the neighbor IPC Process.
    - processInstance, UNSIGNED INTEGER, the application process instance of the neighbor IPC Process.
- **REGISTERED AS** DeleteObjectNotification

B.31.4. IncrementNumberOFEnrollmentAttempts NOTIFICATION

- **BEHAVIOUR** Emitted when numberOfEnrollmentAttempts is incremented.
- **OBJECT VALUE**
  - IncrementNumberOFEnrollmentAttempts, SEQUENCE, information provided by the IncrementNumberOFEnrollmentAttempts notification.
    - processName, STRING, the application process name of the neighbor IPC Process.
    - processInstance, UNSIGNED INTEGER, the application process instance of the neighbor IPC Process.
    - underlyingFlows, SET_OF UNSIGNED INTEGER, the port-id of the N-1 flow used to talk to the neighbor.
    - isEnrolled, BOOL, true if the neighbor is currently enrolled. False otherwise.
- **REGISTERED AS** ChangeAttributeNotification

B.31.5. Neighbors NAME BINDING

- **CONTAINER CLASS** Neighbors.
- **CONTAINED CLASS** Neighbor.
- **NAMED WITH ATTRIBUTE** processName.

CREATE ACTION

- **BEHAVIOUR**

  a. Create the Neighbor RIB object. Causes the target IPC Process to allocate an N-1 flow to the neighbor.
- **INPUT OBJECT VALUE**
  - neighbor, NEIGHBOR_REQUEST, the data related to the "discover neighbor" request (naming information, N-1 DIFs to use, etc)
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

DELETE ACTION

- **BEHAVIOUR**
  1. Causes the IPC Process to release the application connection with the neighbor IPC Process and to deallocate any N-1 flows in common with him.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

B.32. Neighbors MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** - .
- **NOTIFICATIONS** - .
- **NAME BINDINGS**
  - Enrollment.

B.32.1. Enrollment NAME BINDING

- **CONTAINER CLASS** Enrollment.
- **CONTAINED CLASS** Neighbors.
- **NAMED WITH ATTRIBUTE** Static, always "**neighbors**".

B.33. NextHopTable MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** - .
- **NOTIFICATIONS** -.

- **NAME BINDINGS**
    - ResourceAllocator.

B.33.1. ResourceAllocator NAME BINDING

- **CONTAINER CLASS** ResourceAllocator.
- **CONTAINED CLASS** NextHopTable.
- **NAMED WITH ATTRIBUTE** Static, always "**nhopt**".

B.34. NextHopTableEntry MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
    - **key**, UNSIGNED INTEGER, unique key of this entry in the Next Hop Table.
    - **address**, UNSIGNED INTEGER, the destination address to be matched.
    - **qosId**, UNSIGNED INTEGER, the id of the QoS cube of the flow where the PDUs to be routed belong to.
    - **nhopAddresses**, SET_OF UNSIGNED INTEGER, the addresses of the IPC Processes that are the next hops for the PDUs.
- **ACTIONS**
    - READ.
- **NOTIFICATIONS**
    - CreateNextHopTableEntry.
    - DeleteNextHopTableEntry.
- **NAME BINDINGS**
    - NextHopTable.

B.34.1. READ ACTION

- **BEHAVIOUR** Return the NextHopTableEntry object in a READ_R message.

- **OUTPUT OBJECT VALUE**
    - nhoptEntry, NEXT_HOP_TABLE_ENTRY, encapsulates the data contained in the PDU Forwarding Table Entry.

B.34.2. CreateNextHopTableEntry NOTIFICATION

- **BEHAVIOUR** Launches at new NextHopTableEntry object creation.
- **OBJECT VALUE**
    - nhoptEntry, NEXT_HOP_TABLE_ENTRY, encapsulates the data contained in the PDU Forwarding Table Entry.
- **REGISTERED AS** CreateObjectNotification

B.34.3. DeleteNextHopTableEntry NOTIFICATION

- **BEHAVIOUR** Launches at NextHopTableEntry object destruction.
- **OBJECT VALUE**
    - key, UNSIGNED INTEGER, unique key of this entry in the Next Hop Table.
- **REGISTERED AS** DeleteObjectNotification

B.34.4. NextHopTable NAME BINDING

- **CONTAINER CLASS** NextHopTable.
- **CONTAINED CLASS** NextHopTableEntry.
- **NAMED WITH ATTRIBUTE** key.

CREATE ACTION

- **BEHAVIOUR**
    1. Add a static entry to the Next Hop Table.
- **INPUT OBJECT VALUE**
    - nhoptEntry, NEXT_HOP_TABLE_ENTRY, the data of the Next Hop Table Entry.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

DELETE ACTION

- **BEHAVIOUR**
  1. Remove an entry from the Next Hop Table.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

B.35. OSApplicationProcess MANAGED OBJECT CLASS

- **DERIVED FROM** ApplicationProcess.
- **BEHAVIOUR** This class is the Operating System of the ProcessingSystem.
- **ACTIONS** - .
- **NOTIFICATIONS** - .
- **NAME BINDINGS** **KernelApplicationProcess**.

B.35.1. KernelApplicationProcess NAME BINDING

- **CONTAINER CLASS** KernelApplicationProcess.
- **CONTAINED CLASS** OSApplicationProcess.
- **NAMED WITH ATTRIBUTE** Static name, always "**osApplicationProcess**".

B.36. PDUForwardingTable MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** - .
- **NOTIFICATIONS** -.
- **NAME BINDINGS**
  - ResourceAllocator.

B.36.1. ResourceAllocator NAME BINDING

- **CONTAINER CLASS** ResourceAllocator.
- **CONTAINED CLASS** PDUForwardingTable.
- **NAMED WITH ATTRIBUTE** Static, always "**pduft**".

B.37. PDUForwardingTableEntry MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
    - **key**, UNSIGNED INTEGER, unique key of this entry in the PDU Forwarding Table.
    - **address**, UNSIGNED INTEGER, the destination address to be matched.
    - **qosId**, UNSIGNED INTEGER, the id of the QoS cube of the flow where the PDUs to be forwarded belong to.
    - **portIds**, SEQUENCE_OF UNSIGNED INTEGER, the N-1 flow(s) through which the PDU has to be forwarded.
- **ACTIONS**
    - READ.
- **NOTIFICATIONS**
    - CreatePDUForwardingTableEntry.
    - DeletePDUForwardingTableEntry.
- **NAME BINDINGS**
    - PDUForwardingTable.

B.37.1. READ ACTION

- **BEHAVIOUR** Return the PDUForwardingTableEntry object in a READ_R message.
- **OUTPUT OBJECT VALUE**

  - pduftEntry, PDU_FT_ENTRY, encapsulates the data contained in the PDU Forwarding Table Entry.

B.37.2. CreatePDUForwardingTableEntry NOTIFICATION

- **BEHAVIOUR** Launches at new PDUForwardingTableEntry object creation.
- **OBJECT VALUE**
  - pduftEntry, PDU_FT_ENTRY, the data of the PDU Forwarding Table Entry.
- **REGISTERED AS** CreateObjectNotification

B.37.3. DeletePDUForwardingTableEntry NOTIFICATION

- **BEHAVIOUR** Launches at PDUForwardingTableEntry object destruction.
- **OBJECT VALUE**
  - **key**, UNSIGNED INTEGER, unique key of this entry in the PDU Forwarding Table.
- **REGISTERED AS** DeleteObjectNotification

B.37.4. PDUForwardingTable NAME BINDING

- **CONTAINER CLASS** PDUForwardingTable.
- **CONTAINED CLASS** PDUForwardingTableEntry.
- **NAMED WITH ATTRIBUTE** key.
- **CREATE** CREATE.
- **DELETE** DELETE.

CREATE ACTION

- **BEHAVIOUR**
  1. Add a static entry to the PDU Forwarding Table.
- **INPUT OBJECT VALUE**

    - pduftEntry, PDU_FT_ENTRY, the data of the PDU Forwarding Table Entry.

- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

DELETE ACTION

- **BEHAVIOUR**
    1. Remove an entry from the PDU Forwarding Table.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

.xml

B.38. ProcessingSystem MANAGED OBJECT CLASS

- **CLASS NAME** ProcessingSystem.
- **DERIVED FROM** Top .
- **BEHAVIOUR** This class cannot be remotely created or deleted, since it represents the root of the containment subtree of a particular processing system. It is the root object of the Management Agent's RIB.
- **ATTRIBUTES**
    - **processingSystemId**, UNSIGNED INTEGER, uniquely identifies the processing system within the Management Domain.
- **ACTIONS** - .
- **NOTIFICATIONS** - .
- **NAME BINDINGS** ComputingSystem.

B.38.1. ComputingSystem NAME BINDING

- **CONTAINER CLASS** ComputingSystem.
- **CONTAINED CLASS** ProcessingSystem.
- **NAMED WITH ATTRIBUTE** **processingSystemId**.

- **CREATE** This object is automatically created upon Management Agent startup.
- **DELETE** This object cannot be deleted.

B.39. QoSCube MANAGED OBJECT CLASS

- **DERIVED** FROM Top.
- **ATTRIBUTES**
  - **name**, STRING, name of the QoS cube.
  - **id**, UNSIGNED INTEGER, id of the QoS cube.
  - **averageBandwidth**, RANGE_OF UNSIGNED INTEGER, in bits/second.
  - **averageSduBandwidth**, RANGE_OF UNSIGNED INTEGER, in sdus/second.
  - **peakBandwidthDuration**, RANGE_OF UNSIGNED INTEGER, in milliseconds.
  - **burstPeriod**, RANGE_OF UNSIGNED INTEGER, in seconds.
  - **burstDuration**, RANGE_OF UNSIGNED INTEGER, in fractions of Burst period.
  - **undetectedBitErrorRate**, RANGE_OF REAL NUMBER, probability.
  - **maxSduSize**, UNSIGNED INTEGER, in bytes.
  - **partialDelivery**, BOOL, is partial delivery allowed.
  - **incompleteDelivery**, BOOL, is incomplete delivery in order.
  - **orderDelivery**, BOOL, SDUs have to be delivered in order.
  - **maxAllowedSduGap**, UNSIGNED INTEGER, Maximum allowable gap in SDUs.
  - **maxDelay**, UNSIGNED INTEGER, in milliseconds.
  - **maxJitter**, UNSIGNED INTEGER, in milliseconds.
  - **efcpPoliciesConfig**, EFCP_CONNECTION_CONFIG, Configuration of EFCP connections that support flows matched to this QoS cube.

- **ACTIONS**
  - READ, WRITE.
- **NOTIFICATIONS**
  - CreateQoSCube.
  - DeleteQoSCube.
  - ChangeEFCPPoliciesConfig.
- **NAME BINDINGS**
  - QoSCubesNameBinding.

B.39.1. READ ACTION

- BEHAVIOUR Return the configuration of the QoS cube.
- OUTPUT OBJECT VALUE
  - qosCubeConfig, QOS_CUBE_CONFIG, the configuration of the QoS cube.
- RESULT 0 indicates success, a negative number indicates an error otherwise.

B.39.2. WRITE ACTION

- BEHAVIOUR Modify the configuration of the policies that support this QoS cube.
- INPUT OBJECT VALUE
  - qosCubeConfig, QOS_CUBE_CONFIG, the configuration of the QoS cube.
- RESULT 0 indicates success, a negative number indicates an error otherwise.

B.39.3. CreateQoSCube NOTIFICATION

- **BEHAVIOUR** Launches at QoSCube creation.
- **OBJECT VALUE**
  - createNotification, SEQUENCE, attributes for create QoSCube notification

    - **name**, STRING, name of the QoS cube.
    - **id**, UNSIGNED INTEGER, id of the QoS cube.
- **REGISTERED AS** CreateObjectNotification

B.39.4. DeleteQoSCube NOTIFICATION

- **BEHAVIOUR** Launches at QoSCube destroy.
- **OBJECT VALUE**
  - deleteNotification, SEQUENCE, attributes for delete IPC process notification
    - **name**, STRING, name of the QoS cube.
    - **id**, UNSIGNED INTEGER, id of the QoS cube..
- **REGISTERED AS** DeleteObjectNotification

B.39.5. ChangeEFCPPoliciesConfig NOTIFICATION

- **BEHAVIOUR** Launches at a change of the efcpPoliciesConfig attribute.
- **PARAMETERS**
  - oldValue, SET_OF UNSIGNED INTEGER, old value of the attribute
  - newValue, SET_OF UNSIGNED INTEGER, new value of the attribute
- **REGISTERED AS** ChangeAttributeNotification

B.39.6. QoSCubesNameBinding NAME BINDING

- **CONTAINER CLASS** QoSCubes.
- **CONTAINED CLASS** QoSCube.
- **NAMED WITH ATTRIBUTE** qosCubeId.
- **CREATE** CREATE.
- **DELETE** DELETE.

CREATE ACTION

- **BEHAVIOUR**

1. Add a QoS cube to the IPCP. The EFCP policies associated to the QoS cube have to be present in the Processing System.

- **INPUT OBJECT VALUE**
  - qosCubeConfig, QOS_CUBE_CONFIG, the configuration of the QoS cube.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise (unsupported EFCP policies, QoS cube already present).

DELETE ACTION

- **BEHAVIOUR**
  1. Causes the targeted QoS cube to be removed from the RIB. If there are N-flows that belong to this QoS cube they will be terminated.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

B.40. QoSCubes MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** - .
- **NOTIFICATIONS** -.
- **NAME BINDINGS**
  - ResourceAllocator.

B.40.1. ResourceAllocator NAME BINDING

- **CONTAINER CLASS** ResourceAllocator.
- **CONTAINED CLASS** QoSCubes.
- **NAMED WITH ATTRIBUTE** Static, always "**qoscubes**".

B.41. QueryDIFAllocator MANAGED OBJECT CLASS

- **DERIVED FROM** TOP.
- **ATTRIBUTES**
  - **queryDIFAllocatorId**, UNSIGNED INTEGER, unique identifier of an QueryDIFAllocator within a system.
- **ACTIONS**
  - READ.
  - CANCEL_READ.
- **NOTIFICATIONS**
  - ErrorQueryDIFAllocator.
- **NAME BINDINGS**
  - IPCResourceManager.

B.41.1. READ ACTION

- **BEHAVIOUR** Return the QueryDIFAllocator object in a READ_R message.
- **OUTPUT OBJECT VALUE**
  - **queryDIFAllocatorId**, UNSIGNED INTEGER, unique identifier of an QueryDIFAllocator within a system.
- **RESULT** 0 if success, -1 if not.

B.41.2. CANCEL_READ ACTION

- **BEHAVIOUR** Cancel any pending read operations.
- **RESULT** the number of pending read operations canceled, -1 if error.

B.41.3. ErrorQueryDIFAllocator NOTIFICATION

- **BEHAVIOUR** Launches at error in the QueryDIFAllocator operation.
- **PARAMETERS**
  - error, UNSIGNED INTEGER, code of the error.
  - reason, STRING, reason of the error.

- **OBJECT VALUE**
  - queryDIFAllocatorId, UNSIGNED INTEGER, unique identifier of an QueryDIFAllocator within a system.
- **REGISTERED AS** ErrorNotification

B.41.4. IPCResourceManager NAME BINDING

- **CONTAINER CLASS** IPCResourceManager.
- **CONTAINED CLASS** IPCProcess.
- **NAMED WITH ATTRIBUTE** queryDIFAllocatorId.
- **CREATE** This object is created at the IPC Process instantiation.
- **DELETE** This object is deleted at the IPC Process destroy.

B.42. RelayingAndMultiplexing MANAGED OBJECT CLASS

- **DERIVED FROM** ApplicationEntity.
- **ATTRIBUTES**
  - **rmtQMonitorPolicy**, POLICY_CONFIG, the configuration of the RMT queue monitor policy.
  - **rmtSchedulingPolicy**, POLICY_CONFIG, the configuration of the RMT Scheduling Policy.
  - **rmtMaxQPolicy**, POLICY_CONFIG, the configuration of the RMT Max Queue Policy.
- **ACTIONS**
  - READ.
- **NOTIFICATIONS** - .
- **NAME BINDINGS**
  - IPCProcess.

B.42.1. READ ACTION

- **BEHAVIOUR** Return the RMT attributes in a READ_R message.

- **OUTPUT OBJECT VALUE**
    - rmtState, RMT_STATE the state of the RMT.
- **RESULT** 0 if successful, a negative integer indicating an error condition otherwise.

B.42.2. IPCProcess NAME BINDING

- **CONTAINER CLASS** IPCProcess.
- **CONTAINED CLASS** RelayingAndMultiplexing.
- **NAMED WITH ATTRIBUTE** Static, always "**rmt**".

B.43. Report MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
    - **reportID**, UNSIGNED INTEGER, uniquely identifies the DiscriminatorReport within the Management Domain.
    - **timeOfReport**, TIME, time of the report creation.
- **ACTIONS**
    - READ.
    - CANCEL_READ.
- **NOTIFICATIONS** - .
- **NAME BINDINGS**
    - DestinationReports.

B.43.1. READ ACTION

- **BEHAVIOUR** Return the report
- **OUTPUT OBJECT VALUE**
    - report, SEQUENCE, report information.
    - reportID, UNSIGNED INTEGER, uniquely identifies the DiscriminatorReport within the Management Domain.

- timeOfReport, TIME, time of the report creation.

- **RESULT** 0 if success, -1 if not.

B.43.2. CANCEL_READ ACTION

- **BEHAVIOUR** cancel any pending read operations.
- **RESULT** the number of pending read operations canceled, -1 otherwise.

B.43.3. DestinationReports NAME BINDING

- **CONTAINER CLASS** DestinationReports.
- **CONTAINED CLASS** Report.
- **NAMED WITH ATTRIBUTE** reportID.
- **DELETE** DELETE.

DELETE ACTION

- **BEHAVIOUR**
  1. Delete the DiscriminatorReport
- **RESULT** 0 if the DiscriminatorReport object is erased, -1 otherwise.

B.44. ResourceAllocator MANAGED OBJECT CLASS

- **DERIVED FROM** ApplicationEntity.
- **ATTRIBUTES**
  - **resourceAllocationPolicy**, POLICY_CONFIG, the configuration of the resource allocation policy (covers dimensioning and allocation of RMT queues, actions to be taken when queues grow too much, distributed resource allocation collaborating with peer IPCPs).
  - **routingPolicy**, POLICY_CONFIG, the configuration of routing, to generate the next hop table.
  - **pduForwardingTableGeneratorPolicy**, POLICY_CONFIG, the configuration of the PDU Forwarding Table Generator, to generate

the PDU Forwarding Table with input from routing and other sources of information.

- **ACTIONS**
  - READ.
- **NOTIFICATIONS** -.
- **NAME BINDINGS**
  - IPCProcess.

B.44.1. READ ACTION

- **BEHAVIOUR** Return the ResourceAllocator attributes in a READ_R message.
- **OUTPUT OBJECT VALUE**
  - resourceAllocatorState, RESOURCE_ALLOCATOR_STATE the state of the Resource Allocator.
- **RESULT** 0 if successful, a negative integer indicating an error condition otherwise.

B.44.2. IPCProcess NAME BINDING

- **CONTAINER CLASS** IPCProcess.
- **CONTAINED CLASS** ResourceAllocator.
- **NAMED WITH ATTRIBUTE** Static, always "**resalloc**".

B.45. RIBDaemon MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
  - **updatePolicy**, POLICY_CONFIG, configuration of the RIB update policy.
  - **replicationPolicy**, POLICY_CONFIG, configuration of the RIB replication policy.

- **loggingPolicy**, POLICY_CONFIG, configuration of the RIB logging policy.
- **subscriptionPolicy**, POLICY_CONFIG, configuration of the RIB subscription policy.
- **ribAccessControlPolicy**, POLICY_CONFIG, configuration of the access control policy enforced by the RIB Daemon.

- **ACTIONS**
  - READ.
  - CANCEL_READ.
- **NOTIFICATIONS** -.
- **NAME BINDINGS**
  - ApplicationProcess.
  - ManagementAgent.

B.45.1. READ ACTION

- **BEHAVIOUR** Return the RIBDaemon attributes in a READ_R message.
- **OUTPUT OBJECT VALUE**
  - ribDaemonState, RIB_DAEMON_STATE, state of the RIB Daemon.

B.45.2. CANCEL_READ ACTION

- **BEHAVIOUR** Cancel any pending read operations.
- **RESULT** the number of pending read operations canceled, -1 if error.

B.45.3. ApplicationProcess NAME BINDING

- **CONTAINER CLASS** ApplicationProcess.
- **CONTAINED CLASS** RIBDaemon.
- **NAMED WITH ATTRIBUTE** Static name, always "**ribdaemon**".

B.45.4. ManagementAgent NAME BINDING

- **CONTAINER CLASS** ManagementAgent.

- **CONTAINED CLASS** RIBDaemon.
- **NAMED WITH ATTRIBUTE** Static name, always "**ipcmanagement**".

B.46. RMTN1Flow MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
  - **portId**, UNSIGNED INTEGER, the portId of the N-1 flow used by the RMT.
  - **stopped**, BOOL, 0 if stopped, 1 if started.
- **ACTIONS**
  - READ, START, STOP.
- **NOTIFICATIONS**
  - CreateRMTN1Flow.
  - DeleteRMTN1Flow.
- **NAME BINDINGS**
  - RMTN1Flows.

B.46.1. READ ACTION

- **BEHAVIOUR** Return the RMTN1Flow object in a READ_R message.
- **OUTPUT OBJECT VALUE**
  - rmtn1flow_state, RMTN1Flow_STATE, the state of the N-1 flow.

B.46.2. START ACTION

- **BEHAVIOUR**
  1. The RMT can use the N-1 flow again, it can add outgoing SDUs to the outgoing queues associated to the N-1 port-id, and take incoming SDUs from the incoming queues associated to this port-id.
- **RETURN** 0 indicates success, a negative integer indicates failure (N-1 flow was not stopped, could not start N-1 port).

B.46.3. STOP ACTION

- **BEHAVIOUR**
  1. The RMT can no longer use the N-1 flow, it has to stop processing SDUs from input queues or adding SDUs to output queues associated to this N-1 flow.
- **RETURN** 0 indicates success, a negative integer indicates failure (N-1 flow was already stopped, could not stop N-1 port).

B.46.4. CreateRMTN1Flow NOTIFICATION

- **BEHAVIOUR** Launches at new RMTN1Flow object creation.
- **OBJECT VALUE**
  - rmtn1flow_state, RMTN1Flow_STATE, the state of the N-1 flow.
- **REGISTERED AS** CreateObjectNotification

B.46.5. DeleteRMTN1Flow NOTIFICATION

- **BEHAVIOUR** Launches at RMTN1Flow object destruction.
- **OBJECT VALUE**
  - portId, UNSIGNED INTEGER, the portId of the N-1 flow used by the RMT.
- **REGISTERED AS** DeleteObjectNotification

B.46.6. RMTN1Flows NAME BINDING

- **CONTAINER CLAS** RMTN1Flows.
- **CONTAINED CLASS** RMTN1Flow.
- **NAMED WITH ATTRIBUTE** portId.
- **CREATE** CREATE.
- **DELETE** DELETE.

CREATE ACTION

- **BEHAVIOUR**

1. Create the RMTN1Flow RIB object. This action is normally performed by the resource allocator after N-1 flow creation. As part of this operation one or more input/output queue pairs will be instantiated and configured.

- **INPUT OBJECT VALUE**
  - portId, UNSIGNED INTEGER, the port-id of the N-1 flow that will be used by the RMT.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

DELETE ACTION

- **BEHAVIOUR**
  1. Causes the targeted N-1 flow to stop being used by the RMT. All the queues associated to that port will be deleted.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

B.47. RMTN1Flows MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** - .
- **NOTIFICATIONS** - .
- **NAME BINDINGS**
  - RelayingAndMultiplexing.

B.47.1. RelayingAndMultiplexing NAME BINDING

- **CONTAINER CLASS** RelayingAndMultiplexing.
- **CONTAINED CLASS** RMTN1Flows.
- **NAMED WITH ATTRIBUTE** Static, always "**n1flows**".

B.48. RMTQueuePair MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
  - **queueID**, UNSIGNED INTEGER, ID of the Queue, it can be one of the following ones:
    - qosId, the qosId of the PDUs that will be processed in this queue (0 if traffic of the N-1 port is not distinguished by QoS-id).
    - connectionId, the connectionId of the PDUs that will be processed in this queue (0 if queues are not allocated on a per-connection-id basis -equivalent of virtual circuits-).
  - **inQCapBytes**, UNSIGNED INTEGER, capacity of the incoming queue in bytes.
  - **inQSizeBytes**, UNSIGNED INTEGER, occupation of the incoming queue in bytes.
  - **inQSizePDUs**, UNSIGNED INTEGER, occupation of the incoming queue in PDUs.
  - **inQDroppedPDUs**, UNSIGNED INTEGER, number of incoming PDUs dropped.
  - **inQProcessedPDUs**, UNSIGNED INTEGER, number of incoming PDUs processed.
  - **outQCapBytes**, UNSIGNED INTEGER, capacity of the outgoing queue in bytes.
  - **outQSizeBytes**, UNSIGNED INTEGER, occupation of the outgoing queue in bytes.
  - **outQSizePDUs**, UNSIGNED INTEGER, occupation of the outgoing queue in PDUs.
  - **outQDroppedPDUs**, UNSIGNED INTEGER, number of outgoing PDUs dropped.
  - **outQProcessedPDUs**, UNSIGNED INTEGER, number of outgoing PDUs processed.
- **ACTIONS**

  - READ.
- **NOTIFICATIONS**
  - CreateRMTQueuePair.
  - DeleteRMTQueuePair.
- **NAME BINDINGS**
  - RMTN1FlowQoSNameBinding.
  - RMTN1FlowConnNameBinding.

B.48.1. READ ACTION

- **BEHAVIOUR** Return the RMT queue pair object attributes in a READ_R message.
- **OUTPUT OBJECT VALUE**
  - rmtQPairState, RMT_Q_PAIR_STATE, the state of the input/output queue pair.

B.48.2. CreateRMTQueuePair NOTIFICATION

- **BEHAVIOUR** Launches at new RMTQueuePair object creation.
- **OBJECT VALUE**
  - queuePairConfig, SEQUENCE, object containing the configurations needed to create a QueuePair
    - portId, UNSIGNED INTEGER, the port-id of the N-1 flow that will be used by the RMT.
    - queueID, UNSIGNED INTEGER, ID of the Queue.
- **REGISTERED AS** CreateObjectNotification

B.48.3. DeleteRMTQueuePair NOTIFICATION

- **BEHAVIOUR** Launches at RMTQueuePair object destruction.
- **OBJECT VALUE**
  - queuePairConfig, SEQUENCE, object containing the configurations of a QueuePair

    - portId, UNSIGNED INTEGER, the port-id of the N-1 flow that will be used by the RMT.
    - queueID, UNSIGNED INTEGER, ID of the Queue.

- **REGISTERED AS** DeleteObjectNotification

B.48.4. RMTN1Flow NAME BINDING

- **CONTAINER CLASS** RMTN1Flow.
- **CONTAINED CLASS** RMTQueuePair.
- **NAMED WITH ATTRIBUTE** queueID.
- **CREATE** CREATE.
- **DELETE** DELETE.

CREATE ACTION

- **BEHAVIOUR**
  1. Create the QueuePair RIB object. This action is normally performed by the resource allocator after N-1 flow creation. As part of this operation one input/output queue pair will be created.
- **INPUT OBJECT VALUE**
  - queuePairConfig, SEQUENCE, object containing the configurations needed to create a QueuePair
    - portId, UNSIGNED INTEGER, the port-id of the N-1 flow that will be used by the RMT.
    - qosId, UNSIGNED INTEGER, the qos-id of the PDUs that will be processed by this queue.
    - connId, UNSIGNED INTEGER, the connection-id of the PDUs that will be processed by this queue.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

DELETE ACTION

- **BEHAVIOUR**

1. The input/output queue pair will be destroyed.

- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

B.48.5. RMTN1FlowConn NAME BINDING

- **CONTAINER CLASS** MISSING??.
- **CONTAINED CLASS** RMTQueuePair.
- **NAMED WITH ATTRIBUTE** ??.

B.49. Root MANAGED OBJECT CLASS

- **CLASS NAME** Root.
- **DERIVED FROM** **Top** .
- **BEHAVIOUR** This class is just the root of the containment tree. It can only be instantiated once and contains all the objects in the RIB.
- **ATTRIBUTES** -.
- **ACTIONS** - .
- **NOTIFICATIONS** - .
- **NAME BINDINGS** - .

B.50. SDUDelimiting MANAGED OBJECT CLASS

- DERIVED FROM Top.
- BEHAVIOUR This class is just a container with no attributes.
- ATTRIBUTES - .
- ACTIONS - .
- NOTIFICATIONS
- NAME BINDINGS

- IPCProcess.

B.50.1. IPCProcess NAME BINDING

- CONTAINER CLASS IPCProcess.
- CONTAINED CLASS SDUDelimiting.
- NAMED WITH ATTRIBUTE Static, always "**sdudel**".

B.51. SDUDelimitingPolicySet MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
  - **name**, STRING, uniquely identifies the SDU Protection policy set within the IPCP.
  - **flows**, SET_OF UNSIGNED INTEGER, the port-ids of the N Flows where these SDU delimiting policies are currently applied.
  - **fragmentationPolicy**, POLICY_CONFIG, configuration of the fragmentation policy.
  - **concatenationPolicy**, POLICY_CONFIG, configuration of the concatenation policy.
  - **reassemblyAndSeparationPolicy**, POLICY_CONFIG, configuration of the reassembly and separation policy.
- **ACTIONS**
  - READ, WRITE.
- **NOTIFICATIONS -**
- **NAME BINDINGS**
  - SDUDelimiting.

B.51.1. READ ACTION

- **BEHAVIOUR** return an SDU Delimiting policy set.
- **OUTPUT OBJECT VALUE**

- policySetConfiguration, DU_DELIMITING_POLICY_SET_STATE, the configuration of the policies in the SDU Delimiting policy set

- **RESULT** 0 if success, a negative number indicating an error otherwise.

B.51.2. WRITE ACTION

- **BEHAVIOUR**
  1. Modify one or more policies in an existing SDU Delimiting Policy Set. New flows assigned to this policy set will use the new policies.
- **INPUT OBJECT VALUE**
  - policySetConfiguration, SDU_DELIMITING_POLICY_SET_CONFIG, the configuration of the policies in the SDU Delimiting policy set
- **RESULT** 0 if success, a negative number indicating an error otherwise.

B.51.3. SDUDelimiting NAME BINDING

- **CONTAINER CLASS** SDUDelimiting.
- **CONTAINED CLASS** SDUDelimitingPolicySet.
- **NAMED WITH ATTRIBUTE** policySetId.
- **CREATE** CREATE.
- **DELETE** DELETE.

CREATE ACTION

- **BEHAVIOUR**
  1. Check if all the software packages needed to use the delimiting policies are present. If not launch an error, otherwise create the object in the RIB.
  2. The IPCP can use this SDU Delimiting set of policies for the new N-flows.
- **INPUT OBJECT VALUE**
  - policySetConfiguration, SDU_DELIMITING_POLICY_SET_CONFIG, the configuration of the policies in the SDU Delimiting policy set.

- **RESULT** 0 if the policy set validation is successful and the policies are available in the targeted processing system, a negative integer indicating an error otherwise.

DELETE ACTION

- **BEHAVIOUR**
  1. Check if this SDU Delimiting policy set is being used to delimit the IPCP traffic on an existing N-flow
  2. If not, delete the SDUDelimitingPolicySet RIB object and all its attributes, otherwise return an error.
- **RESULT** 0 if the operation is successful, a negative integer indicating an error otherwise.

B.52. SDUProtection MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** - .
- **NOTIFICATIONS** - .
- **NAME BINDINGS**
  - IPCManagement.

B.52.1. IPCManagement NAME BINDING

- **CONTAINER CLASS** IPCManagement.
- **CONTAINED CLASS** SDU Protection.
- **NAMED WITH ATTRIBUTE** Static, always "**sdup**".

B.53. SDUProtectionPolicySet MANAGED OBJECT CLASS

- **DERIVED FROM** Top.

- ATTRIBUTES
  - **name**, STRING, uniquely identifies the SDU Protection policy set within the IPCP.
  - **underlyingFlows**, SET_OF UNSIGNED INTEGER, the port-ids of the N-1 Flows there these SDU protection policies are currently applied.
  - **encryptionPolicy**, POLICY_CONFIG, the configuration of the encryption policy.
  - **compressionPolicy**, POLICY_CONFIG, the configuration of the compression policy.
  - **ttlPolicy**, POLICY_CONFIG, the configuration of the lifetime enforcement policy.
  - **errorCheckPolicy**, POLICY_CONFIG , the configuration of the error check policy.
- ACTIONS
  - READ, WRITE.
- NOTIFICATIONS
  - CreateSDUProtectionPolicySet.
  - DeleteSDUProtectionPolicySet.
  - ChangeEncryptionPolicy.
  - ChangeCompressionPolicy.
  - ChangeTTLPolicy.
  - ChangeErrorCheckPolicy.
- NAME BINDINGS
  - SDUProtection.

B.53.1. READ ACTION

- **BEHAVIOUR** return an SDU Protection policy set.
- OUTPUT OBJECT VALUE
  - policySetConfiguration, SDU_PROTECTION_POLICY_SET_STATE, the configuration of the policies in the SDU Protection policy set

- **RESULT** 0 if success, a negative number indicating an error otherwise.

B.53.2. WRITE ACTION

- **BEHAVIOUR**
  1. Modify one or more policies in an existing SDU Protection Policy Set. New N-1 flows assigned to this policy set will use the new policies, the IPCP will have to renegotiate with their peers the SDU Protection policies for active N-1 flows that were using this policy set.
- **INPUT OBJECT VALUE**
  - policySetConfiguration, SDU_PROTECTION_POLICY_SET_CONFIG, the configuration of the policies in the SDU Protection policy set
- **RESULT** 0 if success, a negative number indicating an error otherwise.

B.53.3. CreateSDUProtectionPolicySet NOTIFICATION

- **BEHAVIOUR** Emitted when a new SDUProtectionPolicySet object is created.
- **OBJECT VALUE**
  - policySetConfiguration, SDU_PROTECTION_POLICY_SET_CONFIG, the configuration of the policies in the SDU Protection policy set.
- **REGISTERED AS** CreateObjectNotification

B.53.4. DeleteSDUProtectionPolicySet NOTIFICATION

- **BEHAVIOUR** Emitted at SDUProtectionPolicySet object destruction.
- **OBJECT VALUE**
  - name, STRING, uniquely identifies the SDU Protection policy set within the IPCP.
- **REGISTERED AS** DeleteObjectNotification

B.53.5. ChangeEncryptionPolicy NOTIFICATION

- **BEHAVIOUR** Emitted when the encryptionPolicy attribute changes.
- **PARAMETERS**

- oldValue, SET_OF UNSIGNED INTEGER, old value of the attribute
- newValue, SET_OF UNSIGNED INTEGER, new value of the attribute

- **REGISTERED AS** ChangeAttributeNotification

B.53.6. ChangeCompressionPolicy NOTIFICATION

- **BEHAVIOUR** Emitted when the compressionPolicy attribute changes.
- **PARAMETERS**
  - oldValue, SET_OF UNSIGNED INTEGER, old value of the attribute
  - newValue, SET_OF UNSIGNED INTEGER, new value of the attribute
- **REGISTERED AS** ChangeAttributeNotification

B.53.7. ChangeTTLPolicy NOTIFICATION

- **BEHAVIOUR** Emitted when the ttlPolicy attribute changes.
- **PARAMETERS**
  - oldValue, SET_OF UNSIGNED INTEGER, old value of the attribute
  - newValue, SET_OF UNSIGNED INTEGER, new value of the attribute
- **REGISTERED AS** ChangeAttributeNotification

B.53.8. ChangeErrorCheckPolicy NOTIFICATION

- **BEHAVIOUR** Emitted when the errorCheckPolicy attribute changes.
- **PARAMETERS**
  - oldValue, SET_OF UNSIGNED INTEGER, old value of the attribute
  - newValue, SET_OF UNSIGNED INTEGER, new value of the attribute
- **REGISTERED AS** ChangeAttributeNotification

B.53.9. SDUProtection NAME BINDING

- **CONTAINER CLASS** SDUProtection.

- **CONTAINED CLASS** SDUProtectionPolicySet.
- **NAMED WITH ATTRIBUTE** name.
- **CREATE** CREATE.
- **DELETE** DELETE.

CREATE ACTION

- **BEHAVIOUR**
  1. Check if all the software packages needed to use the encryption policy, the compression policy, the TTL policy and the error check policy are present. If not launch an error, otherwise create the object in the RIB.
  2. The IPCP can use this SDU Protection set of policies over new N-1 flows.
- **INPUT OBJECT VALUE**
  - policySetConfiguration, SDU_PROTECTION_POLICY_SET_CONFIG, the configuration of the policies in the SDU Protection policy set.
- **RESULT** 0 if the policy set validation is successful and the policies are available in the targeted processing system, a negative integer indicating an error otherwise.

DELETE ACTION

- **BEHAVIOUR**
  1. Check if this SDU Protection policy set is being used to protect the IPCP traffic over an existing N-1 flow.
  2. If not, delete the SDUProtection RIB object and all its attributes, otherwise return an error.
- **RESULT** 0 if the operation is successful, a negative integer indicating an error otherwise.

B.54. SecurityManagement MANAGED OBJECT CLASS

- **DERIVED FROM** ApplicationEntity.

- **ATTRIBUTES**
    - **auditingPolicy**, POLICY_CONFIG, the configuration of the auditing policy.
    - **credentialManagementPolicy**, POLICY_CONFIG, the configuration of the credential management policy.generator.
- **ACTIONS**
    - READ.
- **NOTIFICATIONS** -.
- **NAME BINDINGS**
    - DIFManagement.

B.54.1. READ ACTION

- **BEHAVIOUR** Return the Namespace Manager attributes in a READ_R message.
- **OUTPUT OBJECT VALUE**
    - secManState, SEC_MAN_STATE the state of the Security Management Task.
- **RESULT** 0 if successful, a negative integer indicating an error condition otherwise.

B.54.2. DIFManagement NAME BINDING

- **CONTAINER CLASS** DIFManagement.
- **CONTAINED CLASS** SecurityManagement.
- **NAMED WITH ATTRIBUTE** Static, always "**secman**".

B.55. Software MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** - .

- **NOTIFICATIONS** - .
- **NAME BINDINGS**
    - ProcessingSystem.

B.55.1. ProcessingSystem NAME BINDING

- **CONTAINER CLASS** ProcessingSystem.
- **CONTAINED CLASS** Software.
- **NAMED WITH ATTRIBUTE** Static name, always "**software**"

B.56. Subscription MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
    - subscriptionID, UNSIGNED INTEGER, uniquely identifies the Subscription within the Management Domain.
    - subscriber, STRING, name of the subscriber.
    - objName, STRING, name of the object where the subscriber is subscribed to.
    - operations, SEQUENCE_OF STRING, operations over the objName where the subscriber is subscribed to.
- **ACTIONS**
    - READ, CANCEL_READ.
- **NOTIFICATIONS**
    - CreateSubscription.
    - DeleteSubscription.
- **NAME BINDINGS**
    - RIBDaemon.

B.56.1. READ ACTION

- **BEHAVIOUR** return the Subscription attributes in a READ_R message.

- **OUTPUT OBJECT VALUE**
  - subscriptionConfig, SEQUENCE
    - subscriptionID, UNSIGNED INTEGER, uniquely identifies the Subscription within the Management Domain.
    - subscriber, STRING, name of the subscriber.
    - objName, STRING, name of the object where the subscriber is subscribed to.
    - operations, SEQUENCE_OF STRING, operations over the objName where the subscriber is subscribed to.
- **RESULT** 0 if success, -1 if not.

B.56.2. CANCEL_READ ACTION

- **BEHAVIOUR** cancel any pending read operations.
- **RESULT** the number of pending read operations canceled, -1 if error.

B.56.3. WRITE ACTION

- **BEHAVIOUR** Change the operations where the subscriber is subscribed to.
- **INPUT OBJECT VALUE**
  - remove, BOOL, true indicates that the included operations are going to be deleted from the subscribed operations. false indicates that the included operations are to be added to the subscribed operations.
- **RESULT** 0 if success, -1 if not.

B.56.4. CreateSubscription NOTIFICATION

- **BEHAVIOUR** Launches at Subscription creation.
- **OBJECT VALUE**
  - Sequence subscriptionConfig
    - subscriptionID, unsigned integer, uniquely identifies the Subscription within the Management Domain.
    - subscriber, string, name of the subscriber.

- objName, string, name of the object where the subscriber is subscribed to.
- operations, sequence of string, operations over the objName where the subscriber is subscribed to.

- **REGISTERED AS** CreateObjectNotification

B.56.5. DeleteSubscription NOTIFICATION

- **BEHAVIOUR** Launches at Subscription deletion.
- **OBJECT VALUE**
  - subscriptionID, unsigned integer, uniquely identifies the Subscription within the Management Domain.
- **REGISTERED AS** DeleteObjectNotification

B.56.6. RIBDaemon NAME BINDING

- **CONTAINER CLASS** RIBDaemon.
- **CONTAINED CLASS** Subscription.
- **NAMED WITH ATTRIBUTE** subscriptionID.
- **CREATE** CREATE.
- **DELETE** DELETE.

CREATE ACTION

- **BEHAVIOUR** Subscribe the subscriber to the operations over the object given by the objectName. Create the object in the RIB
- **INPUT OBJECT VALUE**
  - Sequence subscriptionConfig
    - subscriptionID, unsigned integer, uniquely identifies the Subscription within the Management Domain.
    - subscriber, string, name of the subscriber.
    - objName, string, name of the object where the subscriber is subscribed to.
    - operations, sequence of string, operations over the objName where the subscriber is subscribed to.

- **RESULT** 0 if Subscription object has been created, -1 if not.

DELETE ACTION

- **BEHAVIOUR** Delete the subscription and the RIB object.
- **RESULT** 0 if the Subscription object is erased, -1 if not.

B.57. Top MANAGED OBJECT CLASS

- **CLASS NAME** Top.
- **DERIVED FROM** - .
- **BEHAVIOUR** This is an abstract class, the common top-level class from which all other MO classes inherit. It just provides a set of attributes that are common to all managed objects.
- **ATTRIBUTES**
  - **objectClass**, STRING, name of the managed object class.
  - **objectName**, STRING, name of the managed object instance (uniquely the MO instance within the containment tree).
  - **objectInstance**, UNSIGNED INTEGER, (uniquely identifies the object instance within the containment tree)
- **ACTIONS** - .
- **NOTIFICATIONS** - .
- **NAME BINDINGS** - .

B.58. UnderlayingDIF MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
  - **difName**, STRING, name of the underlaying (N-1) DIF.
  - **maxSDUSize**, UNSIGNED INTEGER, maximum SDU size allowed by the DIF (in bytes).

  - **mpl**, UNSIGNED INTEGER, maximum PDU lifetime in the DIF (in milliseconds).
  - **qosCubes**, SEQUENCE OF QOS_CUBE_DESCRIPTION, description of the QoS cubes provided by the DIF.
- **ACTIONS**
  - READ.
- **NOTIFICATIONS**
  - CreateUnderlayingDIF.
  - DeleteUnderlayingDIF.
- NAME BINDINGS
  - UnderlayingDIFsNameBinding.

B.58.1. READ ACTION

- **BEHAVIOUR** Return the AvailableDIF object in a READ_R message.
- **OUTPUT OBJECT VALUE**
  - difProperties, DIF_PROPERTIES, the characteristics of the underlaying DIF.
- **RESULT** 0 if success, -1 if not.

B.58.2. CreateUnderlayingDIF NOTIFICATION

- **BEHAVIOUR** Emitted when an UnderlayingDIF is created.
- **OBJECT VALUE**
  - difProperties, DIF_PROPERTIES, the characteristics of the underlaying DIF.
- **REGISTERED AS** CreateObjectNotification

B.58.3. DeleteUnderlayingDIF NOTIFICATION

- **BEHAVIOUR** Emitted at the destruction of the UnderlayingDIF.
- **OBJECT VALUE**
  - difName, STRING, name of the underlaying (N-1) DIF.
- **REGISTERED AS** DeleteObjectNotification

B.58.4. UnderlayingDIFsNameBinding NAME BINDING

- **CONTAINER CLASS** UnderlayingDIFs.
- **CONTAINED CLASS** UnderlayingDIF.
- **NAMED WITH ATTRIBUTE** difName.
- **CREATE** CREATE.
- **DELETE** DELETE.

CREATE ACTION

- **BEHAVIOUR**
  1. Create the UnderlayingDIF RIB object. Allows the IPC Process to use the N-1 DIF (for registering or for allocating N-1 flows).
- **INPUT OBJECT VALUE**
  - difProperties, DIF_PROPERTIES, the characteristics of the underlaying DIF.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

DELETE ACTION

- **BEHAVIOUR**
  1. This DIF is no longer a valid N-1 DIF for this IPC Process. All active N-1 flows will be deallocated, and all the active registrations to the N-1 DIF will be cancelled.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

B.59. UnderlayingDIFs MANAGED OBJECT CLASS

- **DERIVED** FROM Top.
- **BEHAVIOUR** This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** - .

- **NOTIFICATIONS** - .
- **NAME BINDINGS**
  - IPCResourceManager.

B.59.1. IPCResourceManager NAME BINDING

- **CONTAINER CLASS** IPCResourceManager.
- **CONTAINED CLASS** UnderlayingDIFs.
- **NAMED WITH ATTRIBUTE** Static, always "**underdifs**".

B.60. UnderlayingFlow MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
  - **portId**, UNSIGNED INTEGER, the portId of the N-1 flow.
  - **localAppName**, APP_NAMING_INFO, the local application entity that is using the N-1 flow.
  - **remoteAppName**, APP_NAMING_INFO, the remote application entity that is using the N-1 flow.
  - **flowProperties**, FLOW_PROPERTIES, the characteristics of the N-1 flow (loss, delay, reliability, in order-delivery of SDUs, etc).
- **ACTIONS**
  - READ.
- **NOTIFICATIONS**
  - CreateUnderlayingFlow.
  - DeleteUnderlayingFlow.
- **NAME BINDINGS**
  - UnderlayingFlows.

B.60.1. READ ACTION

- **BEHAVIOUR** Return the UnderlayingFlow object in a READ_R message.

- **OUTPUT OBJECT VALUE**
  - UnderlayingFlowDescriptor, UNDERLAYING_FLOW_DESCRIPTION, encapsulates the data that describes the N-1 flow.

B.60.2. CreateUnderlayingFlow NOTIFICATION

- **BEHAVIOUR** Emitted when an UnderyingFlow is created.
  - UnderlayingFlowDescriptor, UNDERLAYING_FLOW_DESCRIPTION
- **REGISTERED AS** CreateObjectNotification

B.60.3. DeleteUnderlayingFlow NOTIFICATION

- **BEHAVIOUR** Emitted at the destruction of the UnderyingFlow.
- **OBJECT VALUE**
  - deleteUnderlayingFlow, SEQUENCE, attributes for create IPC process notification
    - portId, UNSIGNED INTEGER, the portId of the N-1 flow.
    - localAppName, APP_NAMING_INFO, the local application entity that is using the N-1 flow.
    - remoteAppName, APP_NAMING_INFO, the remote application entity that is using the N-1 flow.
- **REGISTERED AS** DeleteObjectNotification

B.60.4. UnderlayingFlows NAME BINDING

- **CONTAINER CLASS** UnderlayingFlows.
- **CONTAINED CLASS** UnderlayingFlow.
- **NAMED WITH ATTRIBUTE** portId.
- **CREATE** CREATE.
- **DELETE** DELETE.

CREATE ACTION

- **BEHAVIOUR**

1. Create the UnderlayingFlow RIB object. Causes the target IPC Process to request an N-1 flow to a destination IPCP with certain properties.

- **INPUT OBJECT VALUE**
  - flowRequest, UNDERLAYING_FLOW_REQUEST, the data related to the flow request (destination application, flow characteristics).
- RESULT 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

DELETE ACTION

- **BEHAVIOUR**
  1. Causes the targeted N-1 flow to be deallocated.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

B.61. UnderlayingFlows MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** - .
- **NOTIFICATIONS** - .
- **NAME BINDINGS**
  - IPCResourceManager.

B.61.1. IPCResourceManager NAME BINDING

- **CONTAINER CLASS** IPCResourceManager.
- **CONTAINED CLASS** UnderlayingFlows.
- **NAMED WITH ATTRIBUTE** Static, always "**underflows**".

B.62. UnderlayingRegistration MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **ATTRIBUTES**
  - **entityName**, STRING, the name of the AE of the IPC Process registered to the N-1 DIF(s) (can be null).
  - **entityInstance**, STRING, the instance of the AE of the IPC Process registered to the N-1 DIF(s) (can be null).
  - **difNames**, SET_OF STRING, the names of the N-1 DIFs where the AE is registered.
- **ACTIONS**
  - READ.
- **NOTIFICATIONS**
  - CreateUnderlayingRegistration.
  - DeleteUnderlayingRegistration.
- **NAME BINDINGS**
  - UnderlayingRegistrationsNameBinding.

B.62.1. READ ACTION

- **BEHAVIOUR** Return the information of the registration to one or more N-1 DIFs in a READ_R message.
- **OUTPUT OBJECT VALUE**
  - underlayingRegistrationDescriptor, AE_REGISTRATION, encapsulates the data of the registration (AE name/instance, list of DIF names where the AE is registered).
- **RESULT** 0 if success, -1 if not.

B.62.2. CreateUnderlayingRegistration NOTIFICATION

- **BEHAVIOUR** Emitted when an UnderlayingRegistration object is created.
- **OBJECT VALUE**

  - createNotification, SEQUENCE, attributes for create UnderlayingRegistration notification
  - entityName, STRING, the name of the AE of the IPC Process registered to the N-1 DIF(s) (can be null).
  - entityInstance, STRING, the instance of the AE of the IPC Process registered to the N-1 DIF(s) (can be null).
- **REGISTERED AS** CreateObjectNotification

B.62.3. DeleteUnderlayingRegistration NOTIFICATION

- **BEHAVIOUR** Emitted at UnderlayingRegistration object destruction.
- **OBJECT VALUE**
  - deleteNotification, SEQUENCE, attributes for delete UnderlayingRegistration notification
  - entityName, STRING, the name of the AE of the IPC Process registered to the N-1 DIF(s) (can be null).
  - entityInstance, STRING, the instance of the AE of the IPC Process registered to the N-1 DIF(s) (can be null).
- **REGISTERED AS** DeleteObjectNotification

B.62.4. UnderlayingRegistrationsNameBinding NAME BINDING

- **CONTAINER CLASS** UnderlayingRegistrations.
- **CONTAINED CLASS** UnderlayingRegistration.
- **NAMED WITH ATTRIBUTE** entityName.
- **CREATE** CREATE.
- **DELETE** DELETE.

CREATE ACTION

- **BEHAVIOUR**
  1. Create the UnderlayingRegistration RIB object. Causes the target IPC Process to register an AE to one or more DIFs.
- **INPUT OBJECT VALUE**

  - registrationRequest, AE_REGISTRATION, the data related to the registration request. If the registration object already exists and the registration request contains the names of DIFs where the AE is not yet registered, the IPCP will attempt to register the specified AE to the new set of N-1 DIFs. The N-1 DIF names have to be listed under UnderlayingDIFs.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

DELETE ACTION

- **BEHAVIOUR**
  1. Causes the targeted AE to cancel the registration to one or more N-1 DIFs.
- **RESULT** 0 if the operation is successful, a negative error code indicating the reason of failure otherwise.

B.63. UnderlayingRegistrations MANAGED OBJECT CLASS

- **DERIVED FROM** Top.
- **BEHAVIOUR** This class is just a container with no attributes.
- **ATTRIBUTES** - .
- **ACTIONS** - .
  - NOTIFICATIONS* - .
- **NAME BINDINGS**
  - IPCResourceManager.

B.63.1. IPCResourceManager NAME BINDING

- **CONTAINER CLASS** IPCResourceManager.
- **CONTAINED CLASS** UnderlayingRegistrations.
- **NAMED WITH ATTRIBUTE** Static, always "**underregs**".