Date: March 2007

Common and Data Link Layer Facilities Specification, v1.0

OMG Available Specification
formal/07-03-05

The PIM and PSM for Software Radio Components Specification (formal/07-03-01) is physically partitioned into 5 volumes:

Communication Channel and Equipment (formal/07-03-02)
Component Document Type Definitions (formal/07-03-03)
Component Framework (formal/07-03-04)
Common and Data Link Layer Facilities (formal/07-03-05)
POSIX Profiles (formal/07-03-06)

OBJECT MANAGEMENT GROUP

OMG’s Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents, Report a Bug/Issue (*http://www.omg.org/technology/agreement.htm*).

Table of Contents

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG’s specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org/*.

OMG Specifications

As noted, OMG specifications address middleware, modeling, and vertical domain frameworks. A Specifications Catalog is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications.

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM).

Platform Specific Model and Interface Specifications

- CORBAservices
- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications.

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: pubs@omg.org

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/technology/agreement.htm*.

1 Scope

This specification responds to the requirements set by “Request for Proposals for a Platform Independent Model (PIM) and CORBA Platform Specific Model (PSM)” (swradio/02-06-02) of data link layer facilities that can be utilized in developing waveforms, which promotes the portability of waveforms across Software Defined Radios (SDR). The terms Software Radio and Software Defined Radio are used to describe radios that are implemented with strong emphasis on software.

These facilities are applicable for other domains besides SDR that is why this specification is broken out as a separate volume from the SWRadio specification.

The common and data link layer facilities are described in a Platform Independent Model (PIM) that can be transformed into any technology. The specification provides a mechanism for transforming the elements of the PIM model into the platform specific model for CORBA IDL. This mapping definition is given in the PSM (Chapter 8).

2 Conformance

The interfaces and components as defined in section 7 of this specification are not required to be used for a given platform or application. An Application uses the interfaces and component definitions that meet their needs. Conformance is at the level of usage as follows:

- A PSM implementation (no matter what language) of an interface defined in this specification needs to be conformant to the interface definition as described in the specification.
- A PSM implementation (no matter what language) of a component defined in this specification needs to be conformant to the component definition (ports, interfaces realized, properties, etc.) as described in the specification.

3 References

3.1 Normative References

3.1.1 UML and Profile Specifications

3.1.1.1 UML Language Specification

Unified Modeling Language (UML) Superstructure Specification, Version 2.1.1
Formal OMG Specification, document number: formal/07-02-03
The Object Management Group, February 2007
[http://www.omg.org]

Unified Modeling Language (UML) Infrastructure Specification, Version 2.1.1
Formal OMG Specification, document number: formal/07-02-04
The Object Management Group, February 2007
[http://www.omg.org]

3.1.1.2 OCL Language Specification

Object Constraint Language (OCL) Specification, Version 2.0
Formal OMG Specification, document number: formal/2006-05-01
The Object Management Group, May 2006
[http://www.omg.org]

3.1.1.3 UML Profile for CORBA Specification

UML Profile for CORBA Specification, Version 1.0
Formal OMG Specification, document number: formal/2002-04-01
The Object Management Group, April 2002
[http://www.omg.org]

3.1.1.4 UML Profile for Modeling QoS and FT Characteristics and Mechanisms Specification

UML Profile for Modeling QoS and FT Characteristics and Mechanisms, Version 1.0
Formal OMG Specification, document number: formal/06-05-02
The Object Management Group, May 2006
[http://www.omg.org]

3.1.1.5 MOF 2.0/XMI Mapping Specification

Meta Object Facility (MOF) 2.0 XMI Mapping Specification, Version 2.1
Formal OMG Specification, document number: formal/05-09-01
The Object Management Group, September 2005
[http://www.omg.org]

3.1.2 CORBA Core Specifications

3.1.2.1 CORBA Specification

Common Object Request Broker (CORBA/IIOP), Version 3.0.3
Formal OMG Specification, document number: formal/2004-03-01
The Object Management Group, March 2004
[http://www.omg.org]

3.1.2.2 Real-time CORBA Specification

Real-time - CORBA Specification, Version 1.2
Formal OMG Specification, document number: formal/2005-01-04
The Object Management Group, January 2005
[http://www.omg.org]

3.1.2.3 CORBA/e Specification

CORBA/e Specification
Draft Adopted OMG Specification, document number: ptc/06-05-01
The Object Management Group, May 2006
[http://www.omg.org]

3.1.3 UML Models

3.1.3.1 UML Profile for Component Framework

UML Profile for Component Framework XMI File
Formal OMG document number: dtc/2006-04-09
The Object Management Group, August 2006
[http://www.omg.org]

3.1.3.2 Common and Data Link Layer Facilities PIM

Common and Data Link Layer Facilities PIM XMI File
Formal OMG document number: dtc/2006-04-11
The Object Management Group, August 2006
[http://www.omg.org]

3.2 Non-normative References

3.2.1 UML Profile for Component Framework Specification

Component Framework Specification, Version 1.0
Formal OMG document number: formal/07-03-04
The Object Management Group, March 2007
[http://www.omg.org]

3.2.2 Software Radio Facilities IDL

Software Radio Facilities IDL Files
Formal OMG document number:dtc/2006-04-14
The Object Management Group, August 2006
[http://www.omg.org]

4 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

Common Object Request Broker Architecture (CORBA)

An OMG distributed computing platform specification that is independent of implementation languages.

Component

A component can always be considered an autonomous unit within a system or subsystem. It has one or more ports, and its internals are hidden and inaccessible other than as provided by its interfaces. A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. A component exposes a set of ports that define the component specification in terms of provided and required interfaces. As such, a component serves as a type, whose conformance is defined by these provided and required interfaces (encompassing both their static as well as dynamic semantics).

Facility

The realization of certain functionality through a set of well defined interfaces.

Interface Definition Language (IDL)

An OMG and ISO standard language for specifying interfaces and associated data structures.

Logical Device

A software component that is an abstraction of a hardware device it represents.

Mapping

The Specification of a mechanism for transforming the elements of a model conforming to a particular metamodel into elements of another model that conforms to another (possibly the same) metamodel.

Metadata

The Data that represents models. For example, a UML model; a CORBA object model expressed in IDL; and a relational database schema expressed using CWM.

Metamodel

A model of models.

Meta Object Facility (MOF)

An OMG standard, closely related to UML, that enables metadata management and language definition.

Model

A formal specification of the function, structure and/or behavior of an application or system.

Model Driven Architecture (MDA)

An approach to IT system specification that separates the specification of functionality from the specification of the implementation of that functionality on a specific technology platform.

Platform

A set of subsystems/technologies that provide a coherent set of functionality through interfaces and specified usage patterns that any subsystem that depends on the platform can use without concern for the details of how the functionality provided by the platform is implemented.

Platform Independent Model (PIM)

A model of a subsystem that contains no information specific to the platform, or the technology that is used to realize it.

Platform Specific Model (PSM)

A model of a subsystem that includes information about the specific technology that is used in the realization of it on a specific platform, and hence possibly contains elements that are specific to the platform.

Request for Proposal (RFP)

A document requesting OMG members to submit proposals to the OMG's Technology Committee. Such proposals must be received by a certain deadline and are evaluated by the issuing task force.

Service

A set of functionality with common characteristics.

Unified Modeling Language (UML)

An OMG standard language for specifying the structure and behavior of systems. The standard defines an abstract syntax and a graphical concrete syntax.

UML Profile

A standardized set of extensions and constraints that tailors UML to particular use.

5 Symbols and abbreviated terms

Abbreviation	Definition
API	Application Program Interface
CORBA	Common Object Request Broker Architecture
DLPI	Data Link Protocol Interface
GSM	Global System for Mobiles
I/O	Input/Output
ID	Identification, Identifier
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
ISO	International Standards Organization
MAC	Medium Access Control, a sublayer of the OSI Data Link Layer
OMG	Object Management Group
ORB	Object Request Broker
OSI	Open System Interconnection
PIM	Platform Independent Model
PSM	Platform Specific Model
QoS	Quality of Service
RF	Radio Frequency
SDR	Software Defined Radio
SW	Software
UML	Unified Modeling Language
USB	Universal Serial Bus
UMTS	Universal Mobile Telecommunications System

6 Additional Information

6.1 Relationship to Existing OMG Specifications

XML Interdata Interchange (XMI) - The platform-independent model for this specification designed as a UML model that was converted into XMI model exchange format as defined in the XML Interdata Interchange specification.

6.2 Changes to Adopted OMG Specifications

The specifications contained in this document require no changes to adopted OMG specifications.

6.3 Guide to this Specification

This specification consists of two major parts, contained in chapters 7 and 8.

Chapter 7 contains the data link and physical layer facilities PIM. The interfaces and components defined in the data link and physical layers facilities PIM is done using the stereotypes defined in the UML Profile for Component Framework (reference in 4.1.1.3).

Chapter 8 contains a description of the mapping process from the Platform Independent Model (PIM) to a Platform Specific Model (PSM).

The normative UML model referenced in Section 4.1.5.2 is used to generate the class diagrams shown throughout this specification.

6.4 Acknowledgements

The following organizations (listed in alphabetical order) contributed to this specification:

- BAE Systems
- The Boeing Company
- Blue Collar Objects
- Carleton University
- Communications Research Center Canada
- David Frankel Consulting
- École de Technologie Supérieure
- General Dynamics Decision Systems
- Harris
- ITT Aerospace/Communications Division
- ISR Technologies
- L-3 Communications Corporation
- Mercury Computer Systems
- The MITRE Corporation
- Mobile Smarts
- Northrup Grumman
- PrismTech
- Raytheon Corporation
- Rockwell Collins
- SCA Technica
- Space Coast Communication Systems
- Spectrum Signal Processing

- THALES
- Virginia Tech University
- Zeligsoft
- 88solutions

7 Platform Independent Model

The PIM specified in this section is a non-normative specification of the data link layer facilities. The model referenced in Section 3.1.3.1 is the normative definition. It may be realized using many technologies. The CORBA reference PSM in Chapter 8 is one such realization.

The Data Link Layer Facilities PIM Components are made of:

- Common Platform Facilities – The set of interfaces that all components within a platform can be used. Examples of these types of services are log, naming, and event service.
- Common Layer Facilities - The set of interfaces that all components (regardless of any layering) within the radio can realize. Examples of these types of interfaces are flow control, packet, and stream interfaces.
- Data Link Layer Facilities - The set of interfaces that define Link Layer Control (LLC) and Media Access Control (MAC) layer functionality for communication needs.

7.1 Common Platform Facilities

7.1.1 Lightweight Services

7.1.1.1 NamingService

The NamingService provides a white page capability for component registration and retrieval. This white page capability provides the means to have a centralized repository of component references in the system. Servers (components that provide services) register their component references with the NamingService under a unique name so that clients (components that require these services) can find them. Clients find the desired component references distributed throughout the system by their assigned name as published within this while page capability. Once a client finds the desired server component, the client can start requesting the desired services.

Semantics

The NamingService's NameComponent structure is made up of an id-and-kind pair. The "id" element of each NameComponent is a string value that uniquely identifies a NameComponent.

7.1.1.2 EventService

The EventService decouples the communication between consumer and producer components, where consumer components are unaware of producer components, and vice versa. Consumer components process event data that are produced by producer components. The OMG Lightweight Event Service as required by this specification is restricted to support the canonical Push Model approach where producers push events to event channels and event channels in turn push these events to consumers.

The CosLightweightEventComm package is used by consumers for receiving events and by producers for generating events. A component that consumes events shall implement the CosLightweightEventComm PushConsumer interface. A component that produces events shall implement the CosLightweightEventComm PushSupplier interface and use the CosLightweightEventComm PushConsumer interface for generating the events. A producer component shall handle all cases, without raising any exceptions outside of the producer component, due to the connections to a CosEventComm PushConsumer being nil or an invalid reference. The EventService will have the capability to create event channels. An event channel allows multiple suppliers to communicate with multiple consumers asynchronously. An event channel is both a consumer and a producer of events. For Example, event channels can be standard CORBA objects and communication with an event channel is accomplished using standard CORBA requests.

7.1.1.3 LogService

The OMG Lightweight Log Service Specification contains the interfaces and the types necessary for the use of a log. These interfaces consist of the LogProducer, LogConsumer, and LogAdministrator. Using the LogProducer interface, a log producer may generate log records conformant to this specification. Using the LogConsumer interface, a log consumer may retrieve records from a log. Using the LogAdministrator interface, a log administrator may control the operation of a log. Throughout this specification, use of the term Log, Log Service, or LogService refers to any one of these interfaces based upon the context it is used in. Additionally, these interfaces provide operations that may be used to obtain the status of a log. The OMG Lightweight Log Service Specification also defines the types necessary to control the logging output of a log producer. SWRadioComponents that produce logs are required to implement ConfigureProperty(s) and QueryProperty(s) that allow the component to be configured and queried as to what log records it will output.

A LogService may be provided in a software radio installation. The optional aspect of the LogService is restricted to its implementation and deployment. A software radio provider may deliver a product conformant to this specification without a LogService implementation. For instance, a handheld platform with limited resources may choose not to deploy a LogService as part of its domain. Several Infrastructure components contain requirements to write log records using the log service. Components that are required to write log records are also required to account for the absence of a LogService and otherwise operate normally.

Constraints

A log producer is a SWRadioComponent that produces log records using the LogProducer interface. (A component that calls the writeRecord(s) operation of the LogProducer interface.)

A standard record type is defined for all log producers to use when writing log records. The log producer may be configured via the PropertySet interface to output only specific log levels. Log producers shall implement a ConfigureProperty and a QueryProperty with an ID of “PRODUCER_LOG_LEVEL.” The PRODUCER_LOG_LEVEL ConfigureProperty provides the ability to “filter” the log message output of a log producer. The type of the PRODUCER_LOG_LEVEL ConfigureProperty and QueryProperty shall be a Lightweight LogService LogLevelSequence. The LogLevelSequence will contain all log levels that are enabled. Only the messages that contain an enabled log level shall be sent by a log producer to a Log. Log levels that are not in the LogLevelSequence are disabled.

Log producers shall use their component identifier (identifier attribute of the ComponentIdentifier interface) in the producerId field of the CosLwLog ProducerLogRecord.

Log producers shall operate normally in the case where the connections to a Log are nil or an invalid reference.

Log producers shall output only those log records that correspond to enabled CosLwLog LogLevel values.

7.2 Common Layer Facilities

This section defines the Common Layer Facilities, which provide interfaces that cross cut through facilities that correlate to layers. These interfaces can be viewed as building blocks for waveform components that realize multiple interfaces. Figure 7.1 shows the relationships among the packages contained in the Common Layer Facilities part of the PIM. These packages are given as follows:

- Quality of Service Facilities - The set of interfaces that define the quality of service related functionality.

- Flow Control Facilities - The set of interfaces that control communication flow between senders and receivers.
- Measurement Facilities - The set of interfaces that provide measurement parameters and intervals.
- Error Control Facilities - The set of interfaces that allow the Receiver to tell the Sender about frames damaged or lost during transmission, and coordinates the re-transmission of those frames by the Sender.
- PDU Facilities - The set of interfaces that define the Protocol Data Unit (PDU) used in communication among radio sets as well as inter-component communication within a radio.
- Stream Facilities - The set of interfaces that define the stream concept used in communication among radio sets as well as inter-component communication within a radio..

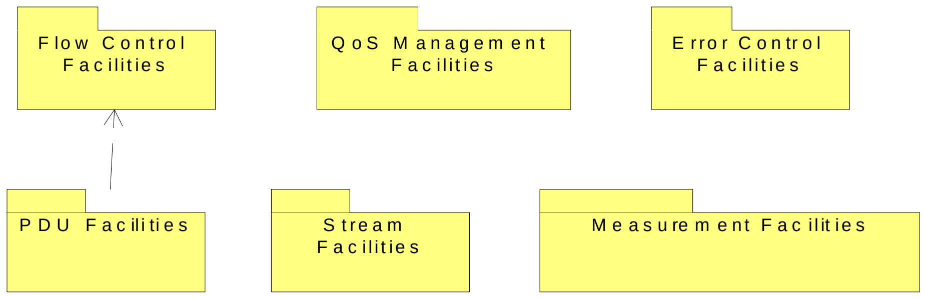


Figure 7.1 - Common Layer Facilities Overview

Types and Exceptions

- `AddressType`
  AddressType is an OctetSequence that represents the source or destination address.
- `SduSizeType (maxSduSize : ULong, minSduSize : ULong)`
  SduSizeType defines the maxSduSize and minSduSize attributes as positive longs. Those two values together define the range of values sduSize can take.

7.2.1 QoS Management Facilities

Quality of Service (QoS) Management Facilities define the facilities that can be used to control quality of service related parameters. The QoS parameters that can be set up are given in the DLPI specification document. Figure 7.2 shows an overview of QoS facilities.

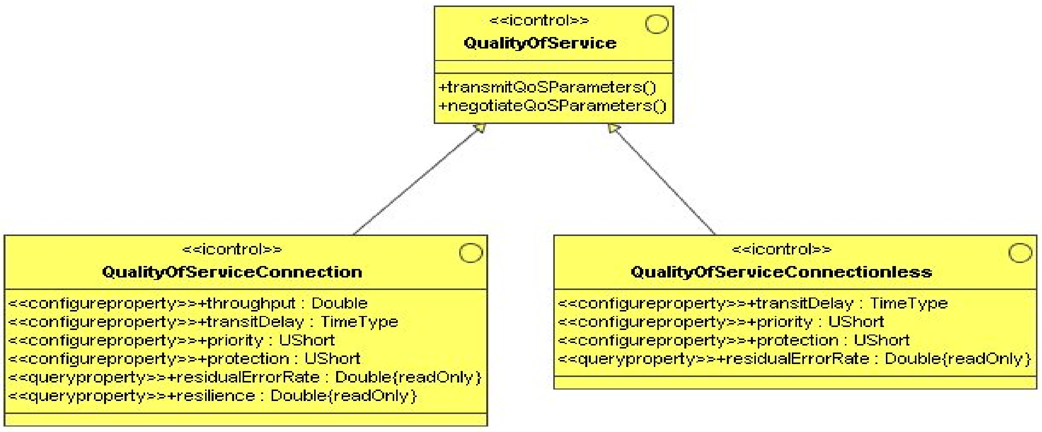

<<icontrol>>
QualityOfService
+transmitQoSParameters()
+negotiateQoSParameters()
<<icontrol>>
QualityOfServiceConnection
<<configureproperty>>+throughput : Double
<<configureproperty>>+transitDelay : TimeType
<<configureproperty>>+priority : UShort
<<configureproperty>>+protection : UShort
<<queryproperty>>+residualErrorRate : Double{readOnly}
<<queryproperty>>+resilience : Double{readOnly}
<<icontrol>>
QualityOfServiceConnectionless
<<configureproperty>>+transitDelay : TimeType
<<configureproperty>>+priority : UShort
<<configureproperty>>+protection : UShort
<<queryproperty>>+residualErrorRate : Double{readOnly}

Figure 7.2 - Quality of Service Facilities Overview

7.2.1.1 QualityOfService

Description

QualityOfService, as shown in Figure 7.2, is the main interface that is used to control the quality of service parameters of a waveform. The parameters that are to be controlled depend on the nature of the established communication link. (Connection-oriented and connectionless). This interface provides the capabilities of signalling and negotiating QoS parameters with waveform components. A component realizing the QualityOfService interface depends on other components that realize

- transmission interfaces (Common Layer Facilities::PDU Facilities) for transferring control and user data,
- flow control interfaces (Common Layer Facilities::Flow Control Facilities) that allow a QoS controller component to change flow control parameters such as data rate, buffer size,
- measurement interfaces (Common Layer Facilities::Measurement Facilities) for monitoring QoS related system performance parameters, and
- error control interfaces (Common Layer Facilities::Error_Control Facilities) for controlling error control coding parameters.

Operations

- `transmitQoSParameters( )`
  This operation signals the quality of service parameters to the requester.
- `negotiateQoSParameters( )`
  This operation provides a generic interface to negotiate the quality of service parameters with the peer receiver/ transmitter.

Semantics

QualityOfService interface depends on the Stream and Pdu interfaces to communicate QoS parameters using transmitQoSParameters operation. A component that plays the AssemblyController role within the same radio set, or the peer receiving radio set may acquire the QoS parameters.

The negotiateQoSParameters operation implies implementation of an encapsulated underlying bidirectional communication protocol. This operation also includes interfacing with error control, flow control and measurement related components within the waveform, to setup their parameters that will meet the quality of service requirements.

7.2.1.2 QualityOfServiceConnection

Description

QualityOfServiceConnection specializes the QualityOfService interface to provide QoS attributes for connection oriented communication establishment. The definition of QualityOfServiceConnection is shown in Figure 7.2.

Attributes

- `<<configureproperty>> throughput : Double`
  Throughput is a connection-mode QoS parameter that has end-to-end significance. It is defined as the total number of Service Data Unit (SDU) bits successfully transferred divided by the greater of both:
  - the time between the first and last data request in a sequence
  - the time between the first and last data indication in the sequence

  Throughput is only meaningful for a sequence of complete SDUs.
- `<<configureproperty>> transitDelay : TimeType`
  The transitDelay attribute indicates the elapsed time between a data request and the corresponding receipt of data. The elapsed time is only computed for SDUs successfully transferred.
- `<<configureproperty>> priority : UShort`
  The specification of priority is concerned with the relationship between connections. This attribute specifies the relative importance of a connection with respect to:
  - The order in which connections are to have their QoS degraded, if necessary.
  - The order in which connections are to be released to recover resources, if necessary. Priority attribute is of UShort type. A lower value means lower priority and vice versa.
- `<<configureproperty>> protection : UShort`
  Protection is the extent to which a provider attempts to prevent unauthorized monitoring or manipulation of user-originated information. Protection is specified by a minimum and maximum protection option within a range of possible protection options. Protection attribute is of UShort type. A lower value means lower protection and vice versa. Protection has local significance only.
- `<<queryproperty>> residualErrorRate : Double`
  Residual Error Rate is the ratio of total incorrect, lost, and duplicated SDUs to the total SDUs transferred between radio sets during a period of time. This property cannot be configured and is used for QoS monitoring purposes only.
- `<<queryproperty>> resilience : Double`
  Resilience is meaningful in connection mode only, and represents the probability of either: provider-initiated disconnects or provider-initiated resets during a time interval of 10,000 seconds on a connection.

Semantics

QualityOfServiceConnection interface inherits transmitQoSParameters and negotiateQoSParameters operations from its base class QualityOfService. Those operations are used respectively transmit and negotiate all of the attributes of the QualityOfServiceConnection interface.

Throughput attribute is specified and negotiated for transmit and receive directions independently at connection establishment. The throughput specification defines the target and minimum acceptable values for a connection. Each specification is an average rate.

Transit delay attribute is negotiated on an end-to-end basis during connection establishment. For each connection, transit delay is negotiated for transmit and receive directions separately by specifying the target value and maximum acceptable value. The transit delay for an individual SDU may be increased if the receiving user flow controls the interface. The average and maximum transit delay values exclude any user flow control of the interface.

Priority attribute is negotiated locally between each user and the provider in connection-mode service. Each user negotiates a particular priority value with the provider during connection establishment. The value is specified by a minimum and a maximum within a given range. This parameter only has meaning in the context of some management entity or structure able to judge relative importance. The priority has local significance only.

Protection attribute is negotiated locally between each user and the provider in connection mode. Provider protects against modification, replay, addition, or deletion of user data. Each user negotiates a particular value with the provider during connection establishment. This attribute only has local significance.

Resilience attribute is not a negotiated QoS parameter. It is set by an administrative mechanism, which is informed of the value by network management.

7.2.1.3 QualityOfServiceConnectionless

Description

QualityOfServiceConnectionless specializes the QualityOfService interface to provide QoS attributes for connectionless communication establishment. Figure 7.2 shows the QualityOfServiceConnectionless definition.

Attributes

- `<<configureproperty>> transitDelay : TimeType`
  This attribute indicates the elapsed time between a data request and the corresponding receipt of data. The elapsed time is only computed for SDUs successfully transferred. This attribute is of TimeType as defined in the Component Framework Profile.

- `<<cconfigureproperty>> priority : UShort`
  This attribute specifies the relative importance of a connectionless communication service. Priority is determined locally for each user in connectionless mode service. A lower value means lower priority and vice versa.

- `<<configureproperty>> protection: UShort`
  Protection is the extent to which a provider attempts to prevent unauthorized monitoring or manipulation of user-originated information. Protection is specified by a minimum and maximum protection option within a range of possible protection options. Protection attribute is of UShort type. A lower value means lower protection and vice versa. Protection has local significance only.

- `<<queryproperty>> residualErrorRate : Double`
  Residual Error Rate is the ratio of total incorrect, lost, and duplicated SDUs to the total SDUs transferred between radio sets during a period of time. This property cannot be configured and is used for QoS monitoring purposes only.

Semantics

In determining the transitDelay attribute, the transmitting radio set selects a particular value within the supported range, and the value may be changed for each SDU submitted for connectionless transmission. The transit delay for an individual SDU may be increased if the receiving user flow controls the interface. The average and maximum transit delay values exclude any user flow control of the interface.

The specification of priority attribute is concerned with the relationship between connectionless data transfer requests. This attribute specifies the relative importance of data units with respect to gaining use of shared resources. The transmitting radio set selects a particular priority value within the supported range, and the value may be changed for each SDU submitted for transmission. This parameter only has meaning in the context of some management entity or structure able to judge relative importance. The priority has local significance only.

Protection attribute has local significance only. Provider protects against modification, replay, addition, or deletion of user data. The transmitting radio set selects a particular value within the supported range, and the value may be changed for each SDU submitted for transmission.

7.2.2 Flow Control Facilities

Flow Control facilities define interfaces that relate to flow control of data transmission and reception. Those facilities control packet flow so that a provider does not transmit more packets than a receiver can process. Flow control is necessary because users and providers are often unmatched in capacity and processing power. The facilities are separated into two interfaces for signaling and control management behavior, namely: FlowControlSignaling and FlowControlManagement interfaces. The goal of flow-control mechanisms is to prevent dropped packets that must be retransmitted. Figure 7.3 shows an overview of Flow Control facilities. Flow Control can be implemented between peer layers for both connection oriented and connectionless communication modes, or at the service boundary between different layers within the same Software Defined Radio (SDR) set.

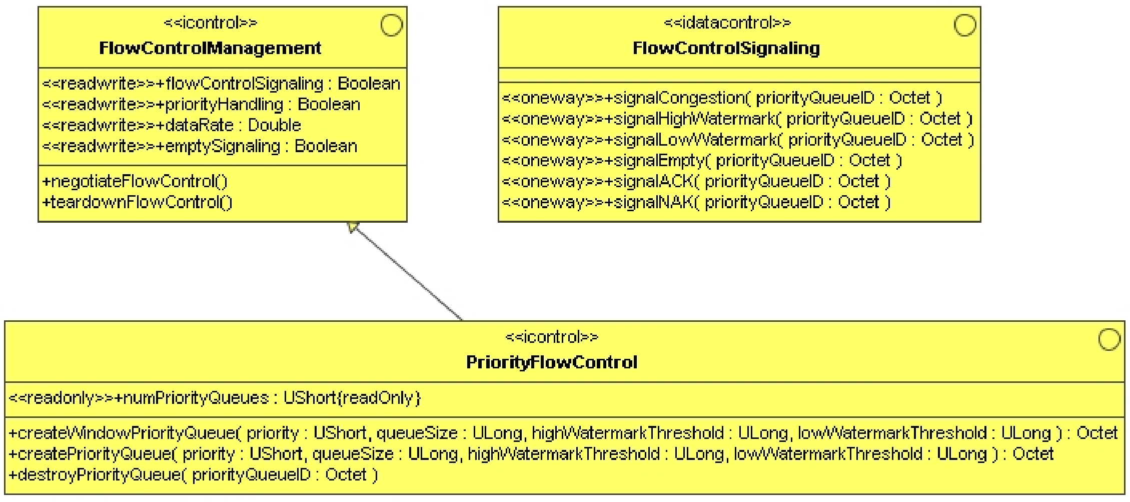

<<icontrol>>
FlowControlManagement
<<readwrite>>+flowControlSignaling : Boolean
<<readwrite>>+priorityHandling : Boolean
<<readwrite>>+dataRate : Double
<<readwrite>>+emptySignaling : Boolean
+negotiateFlowControl()
+teardownFlowControl()
<<idatacontrol>>
FlowControlSignaling
<<oneway>>+signalCongestion(priorityQueueID : Octet)
<<oneway>>+signalHighWatermark(priorityQueueID : Octet)
<<oneway>>+signalLowWatermark(priorityQueueID : Octet)
<<oneway>>+signalEmpty(priorityQueueID : Octet)
<<oneway>>+signalACK(priorityQueueID : Octet)
<<oneway>>+signalNAK(priorityQueueID : Octet)
<<icontrol>>
PriorityFlowControl
<<readonly>>+numPriorityQueues : UShort{readOnly}
+createWindowPriorityQueue(priority : UShort, queueSize : ULong, highWatermarkThreshold : ULong, lowWatermarkThreshold : ULong) : Octet
+createPriorityQueue(priority : UShort, queueSize : ULong, highWatermarkThreshold : ULong, lowWatermarkThreshold : ULong) : Octet
+destroyPriorityQueue(priorityQueueID : Octet)

Figure 7.3 - Flow Control Facilities Definition

7.2.2.1 FlowControlManagement

Description

FlowControlManagement provides an interface for flow control manager component to control and manage flow control related arguments. The interface provides the capabilities of enabling and disabling flow control signaling, enabling priority based queueing, and negotiating flow control parameters with the peer flow controller. Figure 7.4 shows the definition of FlowControlManagement.

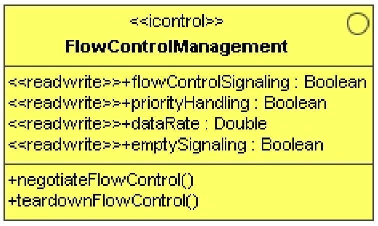

<<icontrol>>
FlowControlManagement
<<readwrite>>+flowControlSignaling : Boolean
<<readwrite>>+priorityHandling : Boolean
<<readwrite>>+dataRate : Double
<<readwrite>>+emptySignaling : Boolean
+negotiateFlowControl()
+teardownFlowControl()

Figure 7.4 - FlowControlManagement Definition

Attributes

- `<<readwrite>> flowControlSignaling: Boolean`
  This attribute indicates whether flow control signalling is currently enabled or not.

- `<<readwrite>> priorityHandling: Boolean`
  This attribute indicates whether priority queue handling is currently enabled or not.

- `<<readwrite>> dataRate : Double`
  Target data rate.

- `<<readwrite>>> emptySignaling : Boolean`
  This attribute indicates whether the flow controller should signal when a queue is empty.

Operations

- `negotiateFlowControl ( )`
  This operation sends a flow control request to the remote radio set in case of a horizontal communication scenario, or to another waveform component in case of a vertical communication scenario. It also sets up flow control related parameters.

- `tearDownFlowControl ( )`
  tearDownFlowControl operation terminates an existing flow control between the user and provider.

Semantics

The negotiateFlowControl and tearDownFlowControl operations indicate an underlying protocol mechanism that allows for two-way handshaking between components in order to negotiate and tear down flow control.

A component realizing FlowControlManagement interface shall communicate with the component that realizes Stream or Pdu interface in order to transmit flow control related data, and with the component that realizes the PriorityQueue interface in order to setup and teardown a priority queue.

7.2.2.2 FlowControlSignaling

Description

FlowControlSignaling provides an interface for sending flow control related signals. The interface provides the signalling capabilities for data congestion, high and low watermark, empty buffer, acknowledgement and negative acknowledgement events. Figure 7.5 shows the definition of FlowControlSignaling.

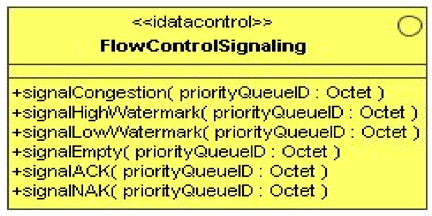

<<idatacontrol>>
FlowControlSignaling
+signalCongestion(priorityQueueID : Octet)
+signalHighWatermark(priorityQueueID : Octet)
+signalLowWatermark(priorityQueueID : Octet)
+signalEmpty(priorityQueueID : Octet)
+signalACK(priorityQueueID : Octet)
+signalNAK(priorityQueueID : Octet)

Figure 7.5 - FlowControlSignaling Definition

Operations

- `<<oneway>>signalCongestion (in priorityQueueID : Octet)`
  This operation signals a congestion (the user cannot handle incoming packets and they are being dropped.
- `<<oneway>>signalHighWatermark (in priorityQueueID : Octet)`
  The signalHighWaterThreshold operation is used to alert the peer entity that high watermark threshold has been reached.
- `<<oneway>>signalLowWatermark (in priorityQueueID : Octet)`
  The signalLowWaterThreshold operation is used to alert the peer entity that low watermark threshold has been reached.
- `<<oneway>>signalEmpty (in priorityQueueID : Octet)`
  The signalEmpty operation signals that the buffer is empty and ready to receive data.
- `<<oneway>>signalACK (in priorityQueueID : Octet)`
  The signalACK operation is used to acknowledge successful reception of a PDU sent by the provider.
- `<<oneway>>signalNAK (in priorityQueueID : Octet)`
  The signalNAK operation is used to acknowledge unsuccessful reception of a PDU sent by the provider.

Semantics

The component that consumes data shall use this interface to indicate to the sender the condition of the data consumer. A component realizing FlowControlSignaling interface shall receive signals from the data consumer through this interface.

7.2.2.3 PriorityFlowControl

Description

PriorityFlowControl interface specializes the FlowControlManagement interface and extends it by adding priority queue handling behavior. This interface can be used to create and destroy both PriorityQueue and WindowedPriorityQueue structures.

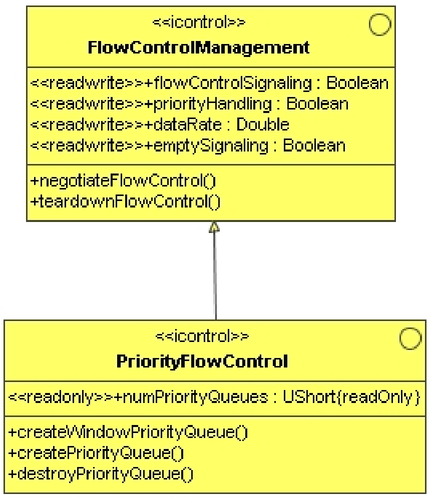

<<icontrol>>
FlowControlManagement
<<readwrite>>+flowControlSignaling : Boolean
<<readwrite>>+priorityHandling : Boolean
<<readwrite>>+dataRate : Double
<<readwrite>>+emptySignaling : Boolean
+negotiateFlowControl()
+teardownFlowControl()
<<icontrol>>
PriorityFlowControl
<<readonly>>+numPriorityQueues : UShort{readOnly}
+createWindowPriorityQueue()
+createPriorityQueue()
+destroyPriorityQueue()

Figure 7.6 - PriorityControlFlow Definition

Attributes

- `<<readonly>> numPriorityQueues : UShort`
  Number of priority queues that the flow controller component is managing. This attribute can only be queried, setting the number of priority queues is done by creating and destroying priority queues using the related operations.

Operations

- `createPriorityQueue (in priority: UShort, in queueSize: ULong, in highWatermarkThreshold: ULong, in lowWaterMarkThreshold: ULong, return Octet)`
  This operation creates a priority queue bound to the flow control manager, and returns the priorityQueueId for it.

- `createWindowedPriorityQueue (in priority: UShort, in queueSize: ULong, in highWatermarkThreshold: ULong, in lowWaterMarkThreshold: ULong, return Octet)`
  This operation creates a windowed priority queue bound to the flow control manager, and returns the priorityQueueId for it.

- `destroyPriorityQueue (in priorityQueueID : Octet)`
  This operation destroys a previously instantiated priority queue. This interface can also be used to destroy a WindowedPriorityQueue, which is a specialization of PriorityQueue.

Types and Exceptions

- `PriorityQueue (priority: UShort, queueSize: ULong, highWatermarkThreshold: ULong, lowWatermarkThreshold: ULong)`
  PriorityQueue provides a type definition for priority queues parameters. Priority queues are used by the flow control mechanism to direct incoming Protocol Data Units (PDU) with different priority tags to corresponding queues. Queues with higher priority get easier access to system resources; while lower priority queue elements wait until higher priority ones are processed. The values of highWatermarkThreshold and lowWatermarkthreshold attributes are application dependent and usually are determined by the flow controller of the QoS controller after negotiating with the remote entity. The interface provides the capability for configuring various parameters of a priority queue such as the queue size, priority level, and high and low watermark levels. The attributes of PriorityQueue type are defined as:

- *priority : UShort*
  Priority defines the relative importance of queues. A lower value means a lower value and vice versa.

- *queueSize : ULong*
  The maximum number of elements the queue can hold.

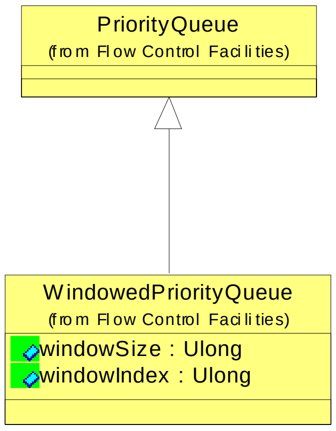
- *highWatermarkThreshold : ULong*
  High watermark threshold shows the number of elements in the queue where a dangerous occupancy is reached in the buffer and probability of dropping PDUs has increased.

- *lowWatermarkThreshold : ULong*
  Low watermark threshold shows the number of elements in the queue where a dangerous occupancy is reached in the buffer and probability of dropping PDUs has increased, although is less than the highWatermarkThreshold point.

- *spaceAvailable : ULong*
  The size of available buffer space in the priority queue in terms of queue elements.

- *priorityQueueID : Octet*
  This attribute is assigned during the instantiation of a priority queue component and is used by other components to uniquely identify the priority queue.

- `WindowedPriorityQueue( windowSize : ULong, windowIndex : ULong)`
  WindowedPriorityQueue specializes the PriorityQueue type in order to provide a mechanism for windowed acknowledgement in a priority queue.

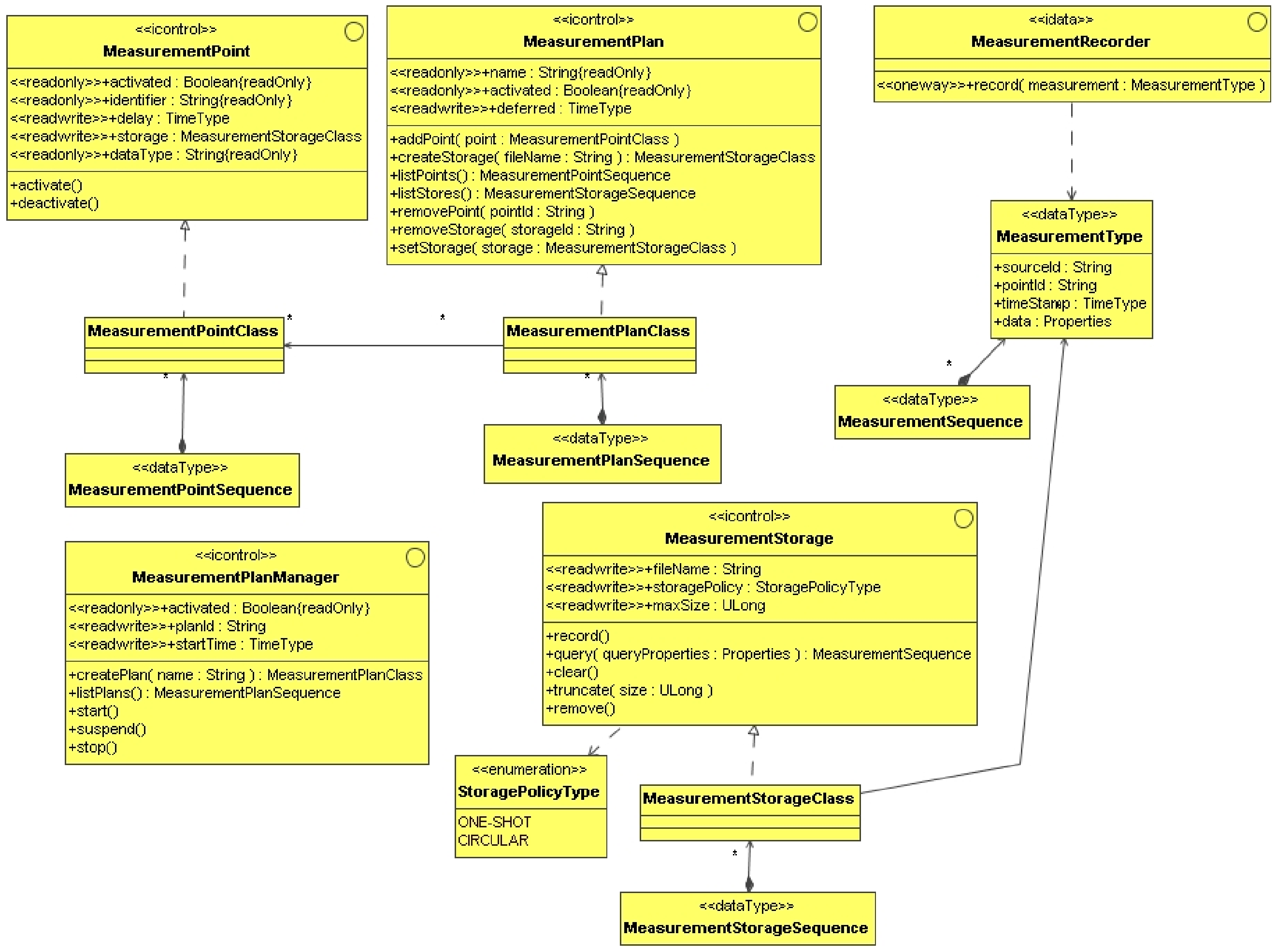


- *windowSize : ULong*
  Size of the window. This attribute can be changed during initialization and/or after the communication has been established.

- *windowIndex: ULong*
  Index of the current data window that is being acknowledged. Every time a window is acknowledged, the index is incremented.

7.2.3 Measurement Facilities

Measurement facilities relate to performing a measurement as requested by a component that has controller functionality over the component that implements the measurement facilities. Any component can be scheduled to perform a measurement, such as traffic volume measurement, bit error rate measurement, voice silence duration measurement, link quality measurement, etc. These measurement plans are communicated to the component through a MeasurementPlan. Measurement Facilities interfaces are shown in Figure 7.7.



<<control>>
MeasurementPoint
<<readonly>>+activated : Boolean{readOnly}
<<readonly>>+identifier : String{readOnly}
<<readwrite>>+delay : TimeType
<<readwrite>>+storage : MeasurementStorageClass
<<readonly>>+dataType : String{readOnly}
+activate()
+deactivate()
<<control>>
MeasurementPlan
<<readonly>>+name : String{readOnly}
<<readonly>>+activated : Boolean{readOnly}
<<readwrite>>+deferred : TimeType
+addPoint(point : MeasurementPointClass)
+createStorage(fileName : String) : MeasurementStorageClass
+listPoints() : MeasurementPointSequence
+listStores() : MeasurementStorageSequence
+removePoint(pointId : String)
+removeStorage(storageId : String)
+setStorage(storage : MeasurementStorageClass)
<<data>>
MeasurementRecorder
<<oneway>>+record(measurement : MeasurementType)
<<dataType>>
MeasurementType
+sourceId : String
+pointId : String
+timeStamp : TimeType
+data : Properties
MeasurementPointClass
MeasurementPlanClass
*
*
*
*
*
<<dataType>>
MeasurementSequence
<<dataType>>
MeasurementPointSequence
<<dataType>>
MeasurementPlanSequence
<<control>>
MeasurementPlanManager
<<readonly>>+activated : Boolean{readOnly}
<<readwrite>>+planId : String
<<readwrite>>+startTime : TimeType
+createPlan(name : String) : MeasurementPlanClass
+listPlans() : MeasurementPlanSequence
+start()
+suspend()
+stop()
<<control>>
MeasurementStorage
<<readwrite>>+fileName : String
<<readwrite>>+storagePolicy : StoragePolicyType
<<readwrite>>+maxSize : ULong
+record()
+query(queryProperties : Properties) : MeasurementSequence
+clear()
+truncate(size : ULong)
+remove()
<<enumeration>>
StoragePolicyType
ONE-SHOT
CIRCULAR
MeasurementStorageClass
*
<<dataType>>
MeasurementStorageSequence

Figure 7.7 - Measurement Facilities Overview

Types and Exceptions

- `MeasurementSequence`
  The MeasurementSequence type represents an unbounded sequence of MeasurementTypes.

- `MeasurementPlanSequence`
  The MeasurementPlanSequence type represents an unbounded sequence of MeasurementPlans.

- `MeasurementPointSequence`
  The MeasurementPointSequence type represents an unbounded sequence of MeasurementPoints.

- `MeasurementStorageSequence`
  The MeasurementStorageSequence type represents an unbounded sequence of MeasurementStorages.

7.2.3.1 MeasurementType

Description

MeasurementType represents the information captured or measured for a MeasurementPoint.

Attributes

- `data: Properties`
  The data attribute represents the measurement data. The measurement point dataType attribute indicates the type of measurement data captured in the measurement.

- `pointId: String`
  The pointId attribute represents the measurement point that made the measurment.

- `sourceId: String`
  The sourceId attribute represents the component that contains the measurement point that made the measurement.

- `timeStamp TimeType`
  The timeStamp represents the time the measurement was made.

Semantics

As MeasurementPoints are activated they create MeasurementTypes that are recorded in a MeasurementStorage.

7.2.3.2 MeasurementPlan

Description

The Measurement Plan interface is used to manage a measurement plan, to configure it, and to manage its measures.

Attributes

- `<<readonly>name: String`
  The name attribute is the name of the measurement plan.

- `<<readonly>>activated: Boolean`
  The activated attribute indicates if the plan is activated. A value of True indicates the plan is activated.

- `<<readwrite>>deferred: TimeType`
  The deferred attribute represents when the activation should take place.

Operations

- `addPoint (in point: MeasurementPoint)`
  The addPoint operation shall add a MeasurementPoint to the plan.

- `createStorage (in fileName: String, return MeasurementStorage`
  The createStorage operation creates a new MeasurementStorage for a plan.
- `listPoints (return MeasurementPointSequence)`
  The listPoints operation shall return all MeasurementPoint(s) attached to this plan by either through the addPoint operation.
- `listStores (return MeasurementStoreSequence)`
  The listStoresoperation shall return all MeasurementStore(s) attached to this plan by either through the create or by the set operations.
- `removePoint (in pointId : String)`
  The removePoint operation shall remove a MeasurementPoint from the plan as specified by the input.
- `removeStorage (in storageId : String)`
  The removeStorage operation shall remove a MeasurementStorage from the plan as specified by the input.
- `setStorage (in storage: MeasurementStorage`
  The setStorage operation sets the current storage for the plan.

7.2.3.3 MeasurementPoint

Description

The MeasurementPoint interface is used to manage a measurement point, to set and to get its configuration, to control its activation and its deactivation, and to set its storage.

Attributes

- `<<readonly>> activated: Boolean`
  The activated attribute indicates if the MeasurementPoint is activated or not. A value of True means the MeasurementPoint is activated.
- `<<readonly>> identifier: String`
  The identifier attribute uniquely identifies a MeasurementPoint.
- `<<readwrite>> delay : TimeType`
  The delay attribute indicates the delay for deferred/immediate measurement.
- `<<readwrite>> storage : MeasurementStorage`
  The storage attribute indicates the current MeasurementStorage associated with measurement point.
- `<<readonly>> dataType : String`
  The dataType attribute indicates the type of data issued from Measurement Point.

Operations

- `activate( )`
  The activate operation activates the MeasurementPoint to start collecting MeasurementTypes.
- `deactivate( )`
  The deactivate operation deactivates the MeasurementPoint from collecting MeasurementTypes.

7.2.3.4 MeasurementPlanManager

Description

The MeasurementPlan interface is used to control a measurement plan.

Attributes

- `<readonly>> activated: Boolean`
  The activated attribute indicates if a MeasurementPlan is activated or not. A value of True indicates a plan is activated.
- `<<readwrite>> planId: String`
  The planId attribute indicates the MeasurementPlan that can be activated or is activated.
- `<<readwrite>> startTime: TimeType`
  The startTime attribute indicates the time to activate the plan.

Operations

- `createPlan (in name: String, return MeasurementPlan)`
  The createPlan operation shall create a MeasurementPlan with the specified input name.
- `listPlans (return MeasurementPlanSequence)`
  The listPlans operation shall return all MeasurementPlans that have been created, which have not been removed since their creation.
- `start ()`
  The start operations shall activate or restart to execute the plan as specified by the planId attribute.
- `stop ()`
  The stop operation shall stop the plan measurement execution.
- `suspend ()`
  The suspend operation shall halt the plan measurement execution.

7.2.3.5 MeasurementRecorder

Description

The MeasurementRecorder interface is used to record measurements.

Operations

- `record:(in in_measurement: MeasurementType)`
  The record operation records a MeasurementType.

7.2.3.6 MeasurementStorage

Description

The MeasurementStorage interface is used to control and retrieve measurements.

Attributes

- `<<readwrite>> fileName: String`
  The fileName attribute indicates the name of file that actually stores records.
- `<<readwrite>> storagePolicy: StoragePolicyType`
  The storagePolicy attribute indicates the storage policy.
- `<<readwrite>> maxSize: ULong`
  The maxSize attribute indicates the allocated size for measurement storage.

Operations

- `clear ()`
  The clear operation shall clear all recorded measurements from storage.
- `query (in queryProperties: Properties, return MeasurementSequence)`
  The query operation enables to retrieve a set of measurements based upon the input queryProperties.
- `record ()`
  The record operation enables to record a measure.
- `remove ()`
  This operation deletes the storage. The component is no longer available for service.
- `truncate (in size : ULong)`
  The truncate operation truncates storage file to new size. The truncate operate shall set the maxSize attribute to the input size value.

Types and Exceptions

- `<<enumeration>>StoragePolicyType (ONESHOT, CIRCULAR)`
  The StoragePolicyType indicates how the storage should be performed.

7.2.4 Error Control Facilities

Error Control facilities allow the Data User (consumer) to tell the Data Provider about the protocol data units damaged or lost during transmission, and coordinate the re-transmission of those data units by the Provider. Since the Flow Control Facilities provide the User's acknowledgement (ACK) of correctly-received data units, it is closely linked to error control. The Error Control interface attributes are communicated to the component through a class of type ErrorControlType. Error Control Facilities also provides a mechanism for receiving status asynchronously by the StatusSignal interface.

7.2.4.1 Error_Control

Description

The Error_Control interface provides a mechanism to establish error control related facilities at both the Provider and User sides of communication. The error control mechanism can be used to change the error control parameters that affect any layer of the waveform.

Attributes

- `<<readwrite>> errorControlParams: ErrorControlParamsType`

This attribute defines which error control attributes are currently enabled to execute for the existing communication link.

Operations

- `checkFrameError( )`
  The checkFrameError operation provides a mechanism to check the incoming data unit against any errors. This operation may be implemented by cyclic redundancy check (CRC) algorithm, or any other algorithm that introduces some redundancy to the SDU and checks for authenticity at the receiver side.

- `checkSequenceNumber( )`
  This operation provides a mechanism to check the sequence number of the received PDU against what has been estimated.

- `estimateSequenceNumber( )`
  The estimateSequenceNumber operation provides a mechanism to estimate the sequence number for the next PDU that is expected.

- `forwardErrorCorrection( )`
  This operation allows the user to correct some of the errors that occurred during reception. Forward error correction works without any feedback mechanism or reporting back to the original sender. The channel coding type of redundancy introduced to the SDU allows the receiver to correct some of the bit errors introduced by the physical channel.

- `requestRetransmit( )`
  The requestRetransmit operation allows the user to request a retransmission of a recently arrived PDU, which contained an error.

- `reportReceptionError( )`
  This operation provides a mechanism to report an error at the reception to the provider port. It is different from the requestRetransmit operation in the sense that it only reports the error and does not request the data to be retransmitted. This operation is more suitable for radio links that have low latency requirements (like video stream).

Types and Exceptions

- `ErrorControlParamsType (ARQStopWait: Boolean, errorControl: Boolean, forwardErrorCorrection: Boolean, slidingWindowARQ: Boolean)`
  The ErrorControlParamsType is a type that defines the error control attributes that can be enabled as a part of the error control facility. It contains ARQStopWait, errorControl, forwardErrorCorrection, and slidingWindowARQ Boolean attributes.

Constraints

If errorControl parameter of errorControlParams attribute is set to be False (no error control at all), then all other parameters of the errorControlParams shall be set to False.

7.2.4.2 StatusSignal

Description

The StatusSignal interface provides a mechanism to asynchronously indicate a status from one component to another component. Figure 7.8 shows the definition of StatusSignal.

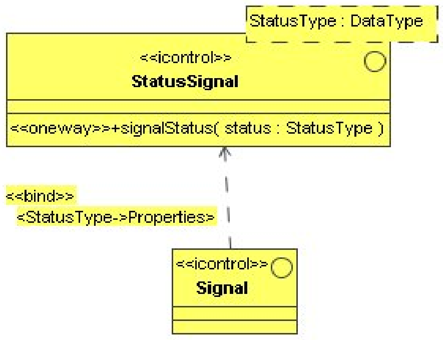


Figure 7.8 - StatusSignal Definition

Operations

- `<<oneway>>signalStatus(in status : statusType)`
  The signalStatus operation provides a mechanism to send a status.

Semantics

The StatusSignal is a template interface. To use this interface one must form a new interface by binding to this interface with a specific StatusType.

7.2.4.3 Signal

Description

The Signal interface provides a generic mechanism to asynchronously indicate a status from one component to another component. Figure 7.8 shows the definition of Signal.

Semantics

The Signal interface is formed from StatusSignal by binding the Component Framework::BaseTypes::Properties for the StatusType template parameter.

7.2.5 Protocol Data Unit Facilities

This facility defines the Protocol Data Unit (PDU) concept that can be used as the smallest data unit element in any waveform layer. PDUs are data elements that are used to store protocol data, and query certain attributes that relate to the usage of PDUs in the waveform protocol. Packet terminology is very specific to Logical Link Layer, so in order to make this concept applicable to any waveform layer that carries data in small units, packet has been renamed as a Protocol Data Unit. The PDU facilities define BasePdu, SimplePdu, Pdu, DataPdu, and PriorityPdu interfaces as shown in Figure 7.9. In order to provide flexibility of usage, those interfaces are specified as parameterized classes. This package also provides two concrete interface recommendations that realize DataPdu and Pdu through binding concrete data types as parameters. The operations and attributes for the interfaces are not shown in Figure 7.9.

.

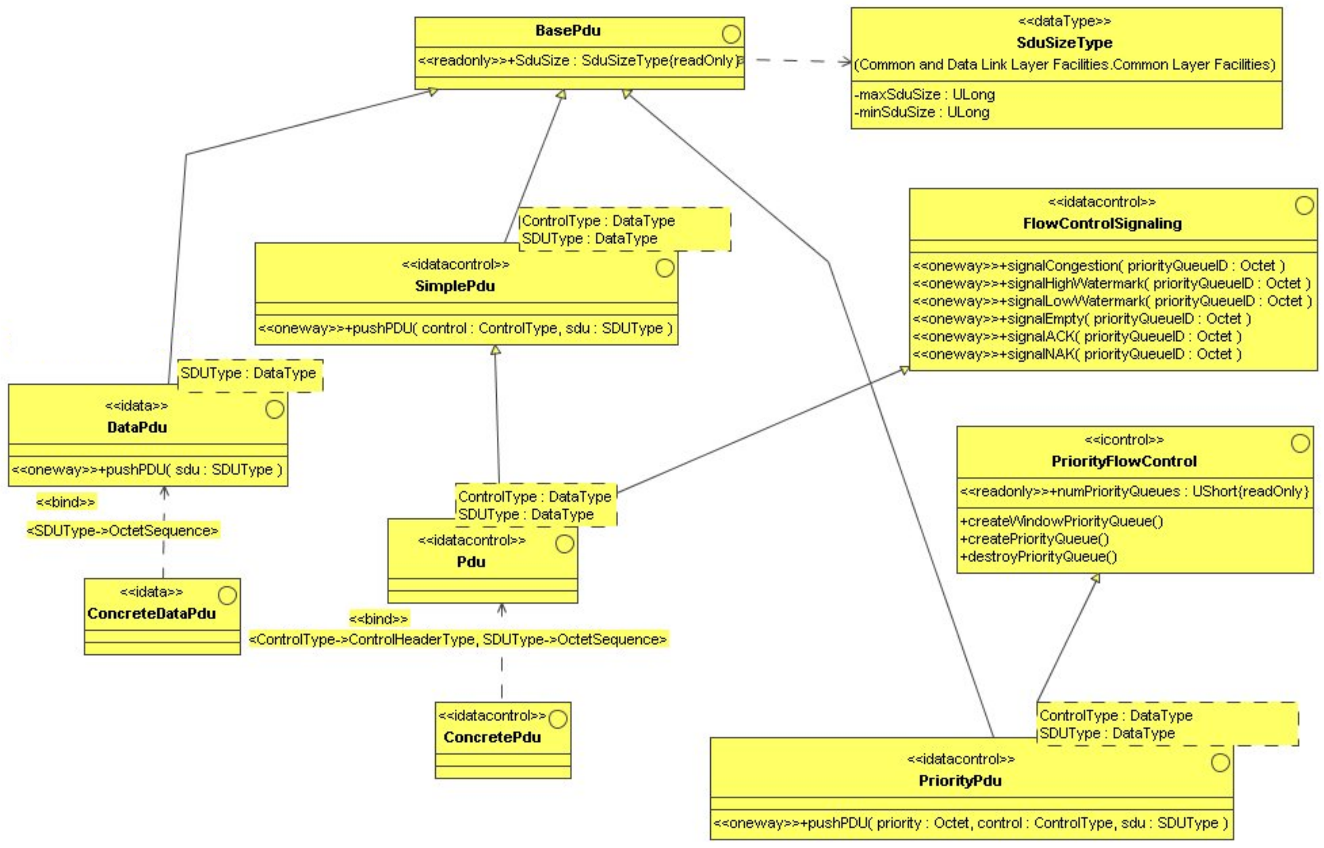

BasePdu
<<readonly>>+SduSize : SduSizeType{readOnly}
<<dataType>>
SduSizeType
(Common and Data Link Layer Facilities.Common Layer Facilities)
-maxSduSize : ULong
-minSduSize : ULong
ControlType : DataType
SDUType : DataType
<<idatacontrol>>
SimplePdu
<<oneway>>+pushPDU(control : ControlType, sdu : SDUType)
<<idatacontrol>>
FlowControlSignaling
<<oneway>>+signalCongestion(priorityQueueID : Octet)
<<oneway>>+signalHighWatermark(priorityQueueID : Octet)
<<oneway>>+signalLowWatermark(priorityQueueID : Octet)
<<oneway>>+signalEmpty(priorityQueueID : Octet)
<<oneway>>+signalACK(priorityQueueID : Octet)
<<oneway>>+signalNAK(priorityQueueID : Octet)
SDUType : DataType
<<idata>>
DataPdu
<<oneway>>+pushPDU(sdu : SDUType)
<<bind>>
<SDUType->OctetSequence>
<<idata>>
ConcreteDataPdu
ControlType : DataType
SDUType : DataType
<<idatacontrol>>
Pdu
<<bind>>
<ControlType->ControlHeaderType, SDUType->OctetSequence>
<<idatacontrol>>
ConcretePdu
<<icontrol>>
PriorityFlowControl
<<readonly>>+numPriorityQueues : UShort{readOnly}
+createWindowPriorityQueue()
+createPriorityQueue()
+destroyPriorityQueue()
ControlType : DataType
SDUType : DataType
<<idatacontrol>>
PriorityPdu
<<oneway>>+pushPDU(priority : Octet, control : ControlType, sdu : SDUType)

Figure 7.9 - PDU Facilities Overview

7.2.5.1 BasePdu

Description

The BasePdu interface is an abstract class that can be specialized by any PDU definition, whether it is used for data, control, or both. This interface forms the basis for SimplePdu and PriorityPdu interfaces. It only defines common attributes that any PDU can have, and does not specify any operations. Those types are defined as dependencies of the BasePdu interface. This interface can be used for both vertical and horizontal communication links. Figure 7.9 shows the definition of BasePdu interface.

Attributes

- `<<readonly>> SduSizeType`
  The SduSize attribute is of type SduSizeType and it specifies the minimum and maximum payload size that can be stored in a single PDU.

7.2.5.2 SimplePdu

Description

The SimplePdu interface is a parametrized class that specializes the BasePdu interface and adds a pushPDU behavior to it. SimplePdu interface is shown in Figure 7.9.

Operations

- `<<oneway>>pushPDU(in control : ControlType, in sdu : SDUType )`
  The pushPDU operation is used to create and send protocol data units through the existing communication link.

7.2.5.3 Pdu

Description

The Pdu interface is a parametrized class which specializes SimplePdu interface and can be implemented using different header and service data unit (SDU) types. Pdu interface also specializes FlowControlSignaling interface, so it supports flow control signalling. This interface can be used for both vertical and horizontal communication links. Figure 7.9 shows the definition of Pdu interface.

7.2.5.4 PriorityPdu

Description

The PriorityPdu interface is a parameterized class that specializes the BasePdu and PriorityFlowControl interfaces. A component realizing the PriorityPdu interface shall contain priority queuing behavior besides the functionalities of a Pdu interface. PriorityPdu also defines a pushPDU behavior which also takes priority information into account. PriorityPdu interface is shown in Figure 7.9.

Operations

- `<<oneway>>pushPDU(in priority : Octet, in control : ControlType, in sdu : SDUType )`
  The pushPDU operation is used to create and send protocol data units through the existing communication link.

7.2.5.5 DataPdu

Description

The BasePdu interface is a parametrized class which specializes BasePdu interface and can be implemented using different SDU types. DataPdu does not have any header information, so it can be used when there is a stream of data to be transferred in frames, with no header requirements. This interface can be used for both vertical and horizontal communication links. Figure 7.9 shows the definition of DataPdu interface.

Operation

- `<<oneway>>pushPDU(in sdu : SDUType )`
  The pushPDU operation is used to create and send protocol data units through the existing communication link.

7.2.5.6 ConcretePdu

Description

ConcretePdu interface realizes the Pdu by binding the SDUType with UML Profile for SWRadio::Application and Device Components::BaseTypes::OctetSequence and ControlType with ControlHeaderType.

Types and Exceptions

- `ControlHeaderType (sourceAddress: AddressType, destinationAddress: AddressType, priority: Long, sduSize: sduSizeType, sequenceNumber: Long)`
  ControlHeaderType is defined in this package in order to provide a concrete PDU definition. This class defines the sourceAddress and destinationAddress fields for the PDU, priority attribute as a ULong, sduSize as the allowed minimum and maximum values and the sequenceNumber, which shows the sequence number of a PDU in a given stream of data packets.

7.2.5.7 ConcreteDataPdu

Description

ConcreteDataPdu interface provides a concrete interface by realizing the parameterized DataPdu and binding the SDUType with UML Profile for SWRadio::Application and Device Components::BaseTypes::OctetSequence.

7.2.6 Stream Facilities

The stream building block defines interfaces to establish and control data streams. These data streams can be used for various purposes and may have different implementations. They can be used for in-band signalling such as transmitting data along with some waveform command and control signals embedded in the stream. They can be used for out-of-band streaming which may be implemented as non-standard CORBA streams.

7.2.6.1 Stream

Description

The Stream interface, as shown in Figure 7.10, defines stream communication capabilities. The interface provides capabilities of establishing and releasing a stream, as well as setting up local parameters at the initialization time of.

Attributes

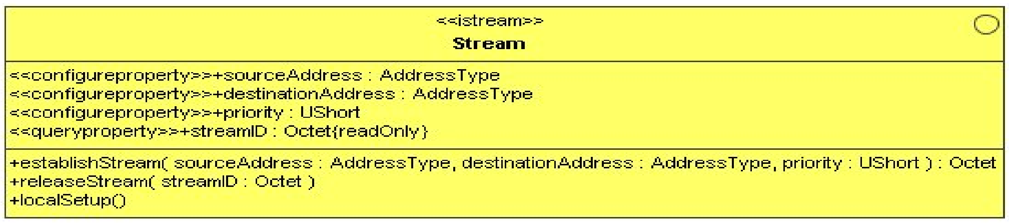

<<istream>>
Stream
<<configureproperty>>+sourceAddress : AddressType
<<configureproperty>>+destinationAddress : AddressType
<<configureproperty>>+priority : UShort
<<queryproperty>>+streamID : Octet{readOnly}
+establishStream(sourceAddress : AddressType, destinationAddress : AddressType, priority : UShort) : Octet
+releaseStream(streamID : Octet)
+localSetup()

Figure 7.10 - Stream Definition

- `<<configureproperty>> sourceAddress : AddressType`
  Source address attribute designates the stream provider. This may refer to a port definition, or in the horizontal communication scenario, an address provided by a high layer protocol such as IP.

- `<<configureproperty>> destinationAddress : AddressType`
  Destination address attribute designates the stream user. This may refer to a port definition, or in the horizontal communication scenario, an address provided by a high layer protocol such as IP.

- `<<configureproperty>> priority : UShort`
  Priority attribute specifies the priority level of the established stream. High priority streams are allocated more resources, relatively less latency and high quality of service operation is expected from a high priority stream implementation.

- `<<queryproperty>> streamID : Octet`
  streamID attribute is the unique ID that the system assigns to the stream. This attribute can only be queried, since it is set by the establishStream operation.

Operations

- `establishStream (in sourceAddress : AddressType, in destinationAddress : AddressType, in priority : UShort, return streamID : Octet)`
  The establishStream operation is used to establish a prioritized data stream by handshaking the stream parameters with the remote component.

- `localSetup( )`
  This operation sets up the local parameters required for setting up a communication stream. These parameters are discussed in the quality of Service building block.

- `releaseStream (in streamID : Octet)`
  The releaseStream operation is used to release the currently established stream. This operation can be a simple teardown of the stream, or a connection termination with acknowledging the peer end, depending on the implementation.

7.3 Data Link Layer Facilities

7.3.1 Link Layer Control Facilities

This section defines the Link Layer Control (LLC) facilities. LLC layer provides facilities to upper layers, for management of communication links between two or more radio sets. LLC layer definition is mainly based on the DLPI specification. DLPI specifies an SCA conformant API that is an instantiation of the ISO Data Link Service Definition DIS 8886 and Logical Link Control DIS 8802-2 (LLC). Where the two standards do not conform, DIS 8886 prevails.

The LLC interface supports three modes of communication: connection, connectionless, and acknowledged connectionless. The connection mode is circuit-oriented and enables data to be transferred over a pre-established connection in a sequenced manner. After the link parameters are negotiated and the link is established, data provider can send a data stream through the link. Data may be lost or corrupted in this service mode, however, due to provider-initiated resynchronization or connection aborts.

The connectionless mode is message-oriented and supports data transfer in self-contained units (PDUs) with no logical relationship required between units. Because there is no acknowledgement of each data unit transmission, this service mode can be unreliable in the most general case. However, a specific logical link provider can provide assurance that messages will not be lost, duplicated, or reordered.

The acknowledged connectionless mode provides the means by which a data link user can send data and request the return of data at the same time. By this way, the transmitter knows which data packets made it through, and retransmits the required packets. Although the exchange service is connectionless, in-sequence delivery is guaranteed for data sent by the initiating station. The data unit transfer is point-to-point.

For each of these communication modes, established link should be controlled locally using local link management interfaces. For this purpose, the LLC facilities are sub-packaged into four different categories.

Types and Exceptions

- `ConnectionIDType (sourceAddress: AddressType, destinationAddress: AddressType, priority: UShort , sapAddress: SAPAddressType, linkService: LinkServiceType)`
  ConnectionIDType completely specifies a logical link that is established at the LLC layer. It specifies the sourceAddress and destinationAddress for radio sets, the sapAddress that the logical link is bound to within the local radio set, as well as the linkService type (connection, connectionless, ack connectionless).
- `<<enumeration>>LinkServiceType (CONNECTION, CONNECTIONLESS, ACKCONNECTIONLESS)`
  The LinkServiceType indicates the type of Data Link Layer service.
- `SAPAddressType (sap:  ULong, address:AddressType)`
  The SAPAddressType contains the SAP address information, where the local link is bound to within the local radio set.

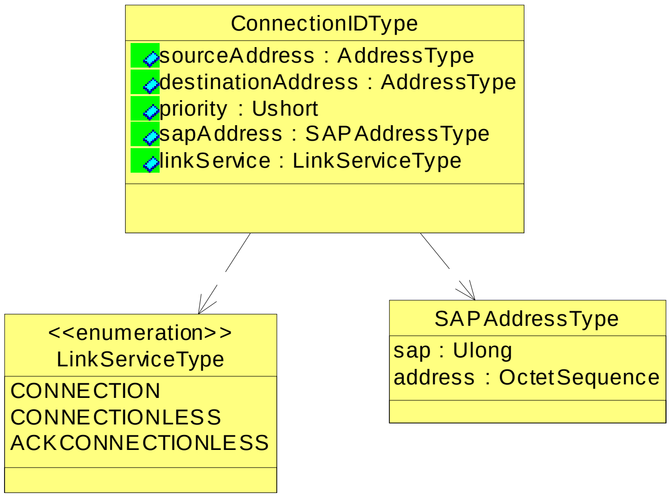


Figure 7.11 - ConnectionIDType Definition

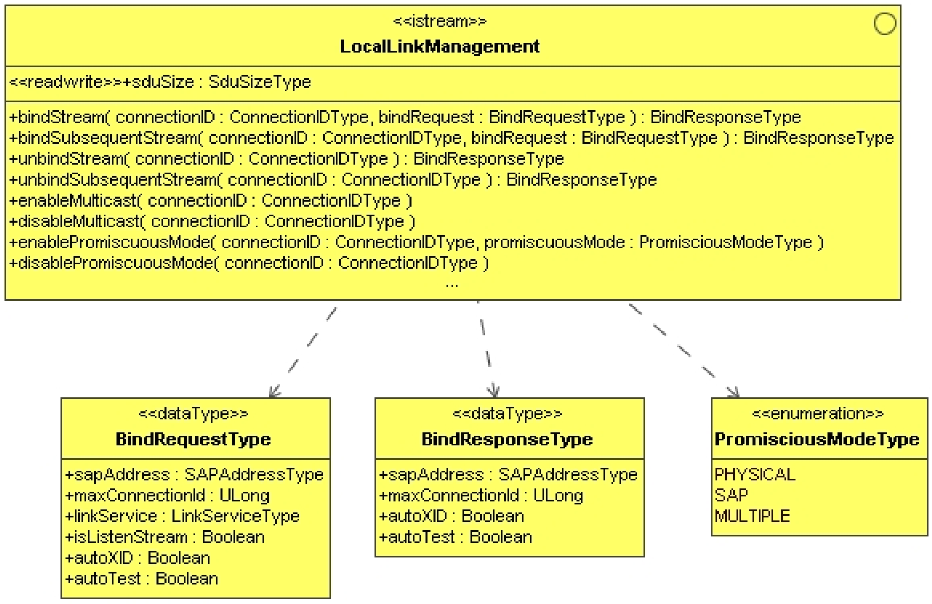
7.3.1.1 Local Link Management Package

This package provides a mechanism to manage the properties of communication links that are instantiated or established by the LLC. The local management services apply to all modes of service. These services, which fall outside the scope of standards specifications, define the method for initializing a stream that is connected to a logical link provider. Logical link provider information reporting services are also supported by the local management facilities. This package consists of a single interface, LocalLinkManagement, and several other type definitions that the interface depends upon.

7.3.1.1.1 LocalLinkManagement

Description

LocalLinkManagement interface provides functionality to control local parameters that are related to link establishment, binding, information reporting, as well as managing connection properties. LocalLinkManagement is defined in Figure 7.12. Every logical link is referenced by a ConnectionID that describes the service SAPs that the link is bound to.



<<istream>>
LocalLinkManagement
<<readwrite>>+sduSize : SduSizeType
+bindStream(connectionID : ConnectionIDType, bindRequest : BindRequestType) : BindResponseType
+bindSubsequentStream(connectionID : ConnectionIDType, bindRequest : BindRequestType) : BindResponseType
+unbindStream(connectionID : ConnectionIDType) : BindResponseType
+unbindSubsequentStream(connectionID : ConnectionIDType) : BindResponseType
+enableMulticast(connectionID : ConnectionIDType)
+disableMulticast(connectionID : ConnectionIDType)
+enablePromiscuousMode(connectionID : ConnectionIDType, promiscuousMode : PromisciousModeType)
+disablePromiscuousMode(connectionID : ConnectionIDType)
...
<<dataType>>
BindRequestType
+sapAddress : SAPAddressType
+maxConnectionId : ULong
+linkService : LinkServiceType
+isListenStream : Boolean
+autoXID : Boolean
+autoTest : Boolean
<<dataType>>
BindResponseType
+sapAddress : SAPAddressType
+maxConnectionId : ULong
+autoXID : Boolean
+autoTest : Boolean
<<enumeration>>
PromisciousModeType
PHYSICAL
SAP
MULTIPLE

Figure 7.12 - LocalLinkManagement Definition

Attributes

- `<<readwrite>> sduSize : sduSizeType`
  This attribute specifies the minimum and maximum service data unit size the LLC resource can transfer. If incoming data is less than this amount, the LLC resource waits to transmit until more data comes in (or until timeout occurs, depending on the implementation).

Operations

- `bindStream(in connectionID : ConnectionIDType, in bindRequest : BindRequestType, return BindResponseType) : {raises = (InvalidPort,ServiceUsageError,SystemError)}`
  The bindStream operation associates a SAP with a stream. The SAP is identified by a SAP address. It requests that the logical link provider bind a SAP to a stream. It also notifies the logical link provider to make the stream active with respect to the SAP for processing connectionless and acknowledged connectionless data transfer and connection establishment requests. Protocol-specific actions taken during activation should be described in logical link provider specific addenda.

- `bindSubsequentStream(in connectionID : ConnectionIDType, in bindRequest : BindRequestType, return BindResponseType) : {raises = (InvalidPort,ServiceUsageError,SystemError)}`

Certain logical link providers require the capability of binding a stream on multiple SAP addresses. BindSubsequentStream operation provides that added capability. The logical link provider returns the bound SAP address in the same primitive. The logical link provider indicates failure by raising an exception.

- `disableMulticast(in connectionID : ConnectionIDType)`
  disableMulticast operation requests the logical link provider to disable specific multicast addresses on a per stream basis.

- `disablePromiscuousMode(in connectionID : ConnectionIDType)`
  This operation requests the provider to disable promiscuous mode on a per Stream basis.

- `enableMulticast(in connectionID : ConnectionIDType)`
  enableMulticast operation requests the logical link provider to enable specific multicast addresses on a per stream basis.

- `enablePromiscuousMode(in connectionID : ConnectionIDType, in promiscouosMode : PromiscuousModeType)`
  This operation requests the provider to enable promiscuous mode on a per Stream basis, either at the physical level or at the SAP level.

- `getInfo(in connectionID : ConnectionIDType, return InfoType): {raises = (InvalidPort, SystemError)}`
  This operation requests information of the provider about the currently established connection. The connectionID parameter identifies a stream (defined as a user connected to the provider). The operation may raise the InvalidPort or SystemException exception.

- `unbindStream(in connectionID : ConnectionIDType) : {raises = (InvalidPort,ServiceUsageError,SystemError)}`
  The unbindStream operation requests the logical link provider to unbind all SAP(s) from a stream. This operation also unbinds all the subsequently bound SAPs that have not been unbound.

- `unbindSubsequentStream(in connectionID : ConnectionIDType) : {raises = (InvalidPort,ServiceUsageError,SystemError)}`
  The unbindSubsequentStream requests the logical link provider to unbind the subsequently bound SAP.

Types and Exceptions

- `BindRequestType`
  This class defines the BindRequest header attributes. This header is passed to the LLC when a connection is required to be bound to a SAP. The attributes of BindRequestType are: sapAddress, maxConnectionId, linkService (type of link service, connection, ack connection or connectionless), isListenStream (Boolean), autoXID (Boolean), autoTest (Boolean)

- `BindResponseType`
  This class defines the BindResponse header attributes. The attributes of BindResponseType are: sapAddress, maxConnectionId, autoXID (Boolean), autoTest (Boolean)

- `InfoType (currentState: StateType,mode: ServiceModeType [1..*], broadcastAddress: CF::OctetSequence broadcastAddress, address, SAPAddressType)`
  The InfoType states information of the provider about the currently established connection.

- `<<enumeration>>PromisciousModeType (PHYSICAL, SAP, MULTIPLE)`
  Promiscuous mode can be enabled on a per connection basis, either at the physical level (PHYSICAL) or at the SAP level, or at a multiple (MULTIPLE) level.

- `<<enumeration>>ServiceErrorType (ERROR_INVALID_STATE, ERROR_UNSUPPORTED, ERROR_BAD_ADDRESS, ERROR_BAD_CORRELATION,ERROR_NOT_ENABLED, ERROR_TOO_MANY,`

`ERROR_NO_ACCESS, ERROR_BOUND, ERROR_NO_AUTO, ERROR_NO_XIDAUTO, ERROR_NO_TESTAUTO, ERROR_BAD_DATA, ERROR_NO_ADDRESS, ERROR_BAD_SAP, ERROR_BAD_QOS_PARAMETERS, ERROR_UNDELIVERABLE)`
The ServiceErrorType indicates the service error for a Link Layer component.

- `<<enumeration>>ServiceModeType (CODLS, CLDLS, ACLDLS)`
  ServiceModeType indicates the service mode for a Link Layer component.
- `<<enumeration>>StateType (UNATTACHED, UNBOUND, IDLE, OUTCON_PENDING, INCON_PENDING, CONN_RES_PENDING, DATAXFER, USER_RESET_PENDING, PROV_RESET_PENDING, RESET_RES_PENDING, DISCON_PENDING_OUTCON, DISCON_PENDING_INCON, DISCON_PENDING_DATAXFER, DISCON_PENDING_USER_RESET, DISCON_PENDING_PROV_RESET)`
  The StateType indicates the state for a Link Layer component.
- `<<exception>>InvalidPort`
  The InvalidPort indicates an invalid port request.
- `<<exception>>ServiceUsageError (ServiceErrorType qualifier)`
  The ServiceErrorType exception indicates the service usage error for a Link Layer component.
- `<<exception>>SystemError (errorNo: ULong)`
  SystemError exception indicates a system exception has occurred and the error number indicates the system problem.

7.3.1.2 Connectionless Link Package

This package provides facilities to provide connectionless mode communication for an LLC layer. The connectionless mode is message-oriented and supports data transfer in self-contained units with no logical relationship required between units. This package consists of a main interface, ConnectionlessLink and several other type definitions that the interface depends upon.

The connectionless mode package does not use the connection establishment and release phases of the connection-mode service. The local management phase is still required to initialize a stream. Once initialized, however, the connectionless data transfer phase is immediately entered. Because there is no established connection, however, the connectionless data transfer phase requires the LLC user to identify the destination of each protocol data unit to be transferred. The destination LLC user is identified by the address associated with that user. Since there is no acknowledgement of each PDU transmission, this service mode can be unreliable in the most general case. However, a specific link layer or MAC provider can provide flow and error control mechanisms to assure that messages will not be lost, duplicated, or reordered.

7.3.1.2.1 ConnectionlessLink Component

Description

ConnectionlessLink component as shown in Figure 7.13, provides functionality to control parameters that are related to connectionless link establishment, and management as well as preparing and sending protocol data units. After the connection is established, data can be transferred using ConnectionlessLink component for unacknowledged connectionless communication scenario. This component realizes the QualityOfServiceConnectionless interface for quality of service related facilities, FlowControlSignaling for flow control interfaces, and Indicator and RequestPdu interfaces for PDU based communication. Connectionless link component can provide facilities for segmentation and reassembly of protocol data, as well as concatenation and padding of PDU's to match the protocol specification. Every logical link is referenced by a ConnectionID that describes the port(s) that the link is bound to. Local link management interface establishes the links and bounds them to vertical (internal) streams. This component can have a user role, a provider role, or both; depending on the waveform scenario.

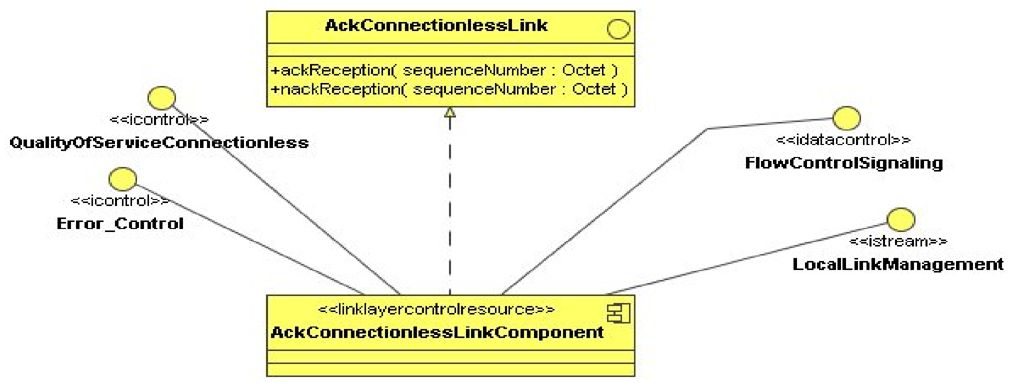

AckConnectionlessLink
+ackReception(sequenceNumber : Octet)
+nackReception(sequenceNumber : Octet)
<<icontrol>>
QualityOfServiceConnectionless
<<icontrol>>
Error_Control
<<idatacontrol>>
FlowControlSignaling
<<istream>>
LocalLinkManagement
<<linklayercontrolresource>>
AckConnectionlessLinkComponent

Figure 7.13 - ConnectionlessLinkComponent Definition

Constraints

ConnectionlessLinkComponent shall provide one ControlPort, at least one input DataControl port and at least one output DataControl port.

7.3.1.2.2 IndicatorPdu

Description

IndicatorPdu interface, as shown in Figure 7.14, realizes the PriorityPdu interface from the Common Layer Facilities::PDU Facilities, by binding ControlHeaderType to MediumAccessControlHeaderType and SDUType to OctetSequence.

.

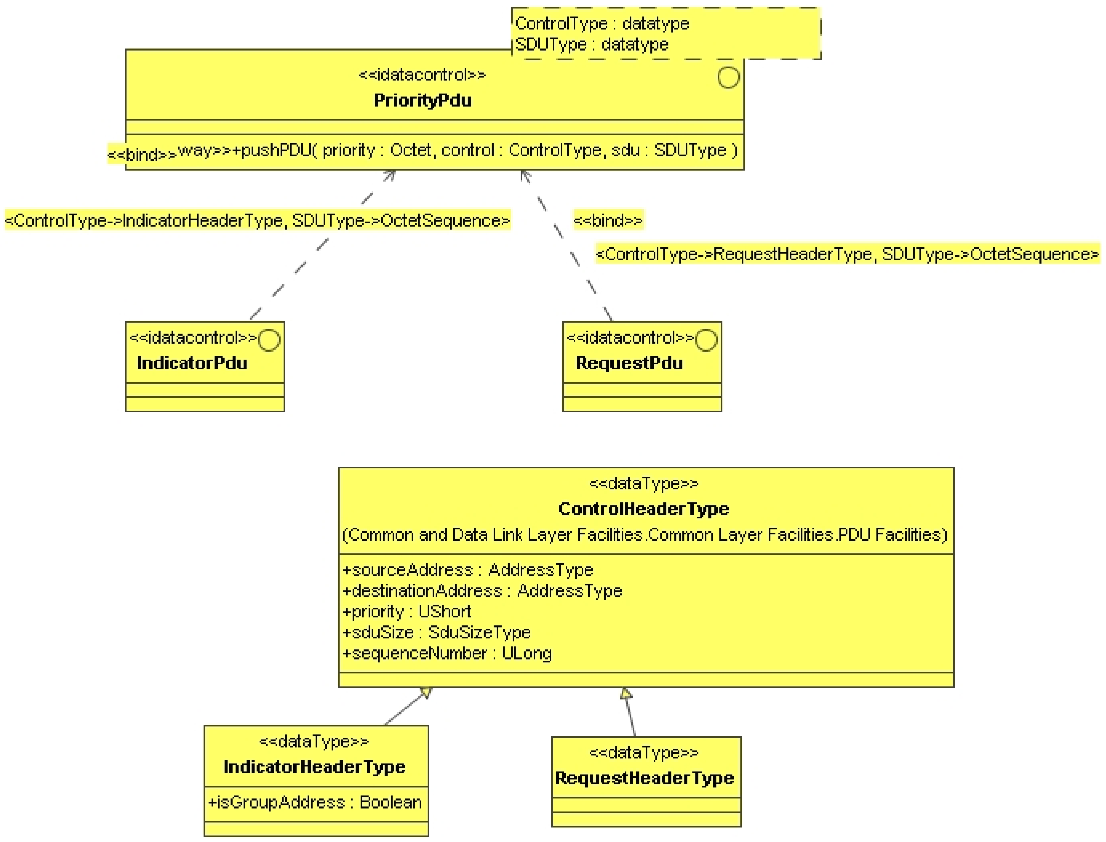

ControlType : datatype
SDUType : datatype
<<idatacontrol>>
PriorityPdu
<<bind>>way>>+pushPDU(priority : Octet, control : ControlType, sdu : SDUType)
<ControlType->IndicatorHeaderType, SDUType->OctetSequence>
<<bind>>
<ControlType->RequestHeaderType, SDUType->OctetSequence>
<<idatacontrol>>
IndicatorPdu
<<idatacontrol>>
RequestPdu
<<dataType>>
ControlHeaderType
(Common and Data Link Layer Facilities.Common Layer Facilities.PDU Facilities)
+sourceAddress : AddressType
+destinationAddress : AddressType
+priority : UShort
+sduSize : SduSizeType
+sequenceNumber : ULong
<<dataType>>
IndicatorHeaderType
+isGroupAddress : Boolean
<<dataType>>
RequestHeaderType

Figure 7.14 - IndicatorPdu and RequestPdu Definitions

Types and Exceptions

- `IndicatorHeaderType (isGroupAddress: Boolean)`
  Indicator header is used to conveys one SDU from the LLC provider to the LLC user. IndicatorHeaderType inherits from ControlHeaderType defined in the Common Layer Facilities::PDU Facilities package.

  isGroupAddress attribute defines whether the destination address is a group address.

7.3.1.2.3 RequestPdu

Description

RequestPdu interface, as shown in Figure 7.14, realizes the Pdu interface from the Common Layer Facilities::PDU Facilities, by binding ControlHeaderType to MediumAccessControlHeaderType and SDUType to OctetSequence.

Types and Exceptions

- `RequestHeaderType`
  This header type conveys one SDU from the LLC user to the LLC provider for transmission to a peer LLC user. RequestHeaderType inherits from ControlHeaderType defined in the Common Layer Facilities::PDU Facilities package.

7.3.1.3 Acknowledged Connectionless Link Package

This package provides facilities to provide acknowledged connectionless mode communication for LLC layer. The acknowledged connectionless mode is message-oriented and supports data transfer in self-contained units with no logical relationship required between units. Although the exchange service is connectionless, in-sequence delivery is guaranteed for data sent by the initiating station. The acknowledged connectionless mode provides the means by which a data link user can send data and request the return of data at the same time. The data unit transfer is point-to-point. This package consists of a main interface, AckConnectionlessLink and several other type definitions that the interface depends upon.

The acknowledged connectionless mode package also does not use the connection establishment and release phases of the connection-mode service. The local management phase is still required to initialize a stream. Once initialized, the acknowledged connectionless data transfer phase is immediately entered. Because there is no established connection, the LLC user is required to identify the destination of each protocol data unit to be transferred. The destination LLC user is identified by the address associated with that user.

Acknowledged connectionless data transfer guarantees that ‘data units will be delivered to the destination user in the order in which they were sent. A data link user entity can send a data unit to the destination LLC user, request a previously prepared data unit from the destination LLC user, or exchange data units.

7.3.1.3.1 AckConnectionless

Description

AckConnectionlessLink interface as shown in Figure 7.15, provides the extra functionality to control parameters that are related to acknowledged connectionless link establishment and management.

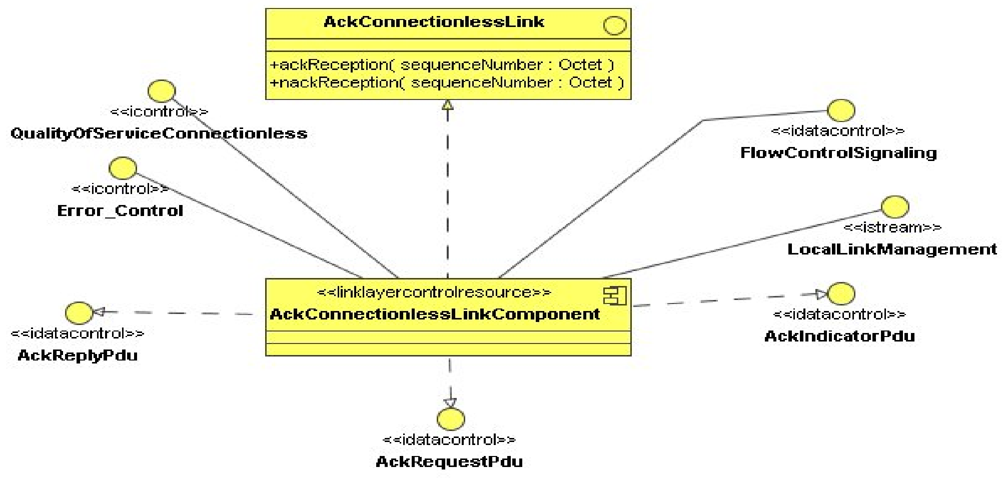


Figure 7.15 - AckConnectionlessLink Definition

Operations

- `ackReception (in sequenceNumber : Octet)`
  Acknowledgement of received PDU.

- `nakReception(in sequenceNumber : Octet)`
  Negative acknowledgement of PDU. This operation indicates that an expected data packet was not received at all, or it was received in error.

Types and Exceptions

- `<<enumeration>> PacketIndicatorType (PI_ONEWAY, PI_TWOWAY)`
  PacketIndicatorType specifies whether one way or two way packet indication will be used.

7.3.1.3.2 AckReplyPdu

Description

AckReplyPdu interface, as shown in Figure 7.16, realizes the PriorityPdu interface from the Common Layer Facilities::PDU Facilities, by binding ControlHeaderType to RequestHeaderType and SDUType to OctetSequence. This PDU interface shall be used when replying to a data request.

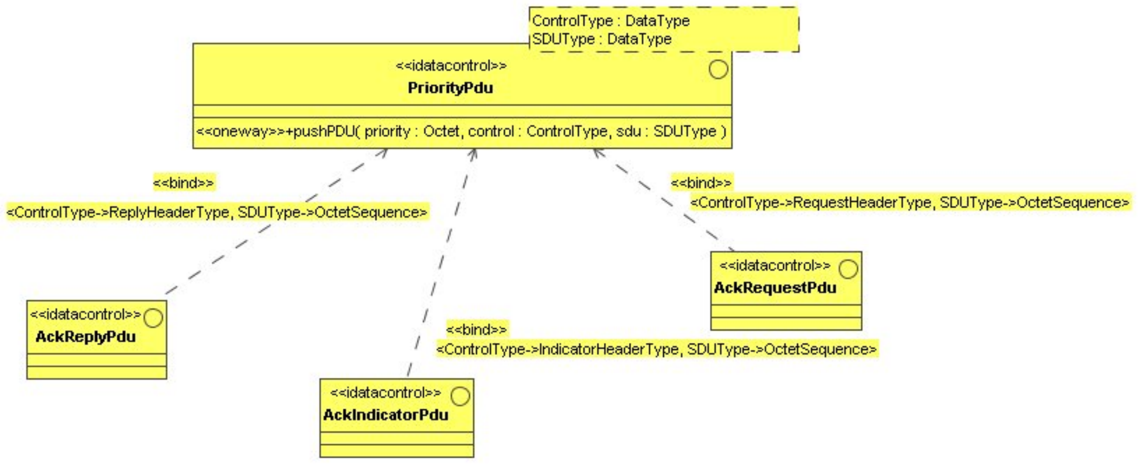

ControlType : DataType
SDUType : DataType
<<idatacontrol>>
PriorityPdu
<<oneway>>+pushPDU(priority : Octet, control : ControlType, sdu : SDUType)
<<bind>>
<ControlType->ReplyHeaderType, SDUType->OctetSequence>
<<bind>>
<ControlType->RequestHeaderType, SDUType->OctetSequence>
<<idatacontrol>>
AckRequestPdu
<<idatacontrol>>
AckReplyPdu
<<bind>>
<ControlType->IndicatorHeaderType, SDUType->OctetSequence>
<<idatacontrol>>
AckIndicatorPdu

Figure 7.16 - AckReplyPdu, AckIndicatorPdu, and AckRequestPdu Definitions

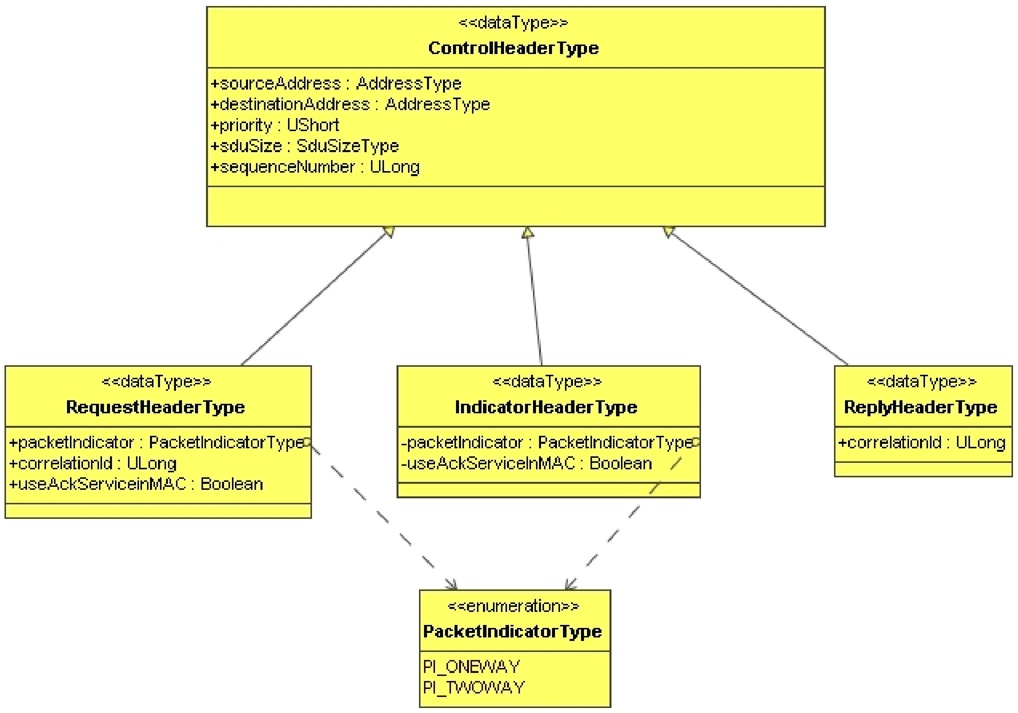


Figure 7.17 - AckConnectionLink Header Types

Types and Exceptions

- `replyHeaderType`
  Conveys an SDU to the LLC provider from the LLC user to be held by the LLC provider and sent out at a later time when requested to do so by the peer LLC provider. See replyHeaderType shown in Figure 7.17.

7.3.1.3.3 AckIndicatorPdu

Description

AckIndicatorPdu interface, as shown in Figure 7.16, realizes the PriorityPdu interface from the Common Layer Facilities::PDU Facilities, by binding ControlHeaderType to IndicatorHeaderType and SDUType to OctetSequence. This PDU interface shall be used when indicating successful or unsuccessful data request or transfer.

Types and Exceptions

- `indicatorHeaderType`
  This header type is passed from the LLC provider to the LLC user to indicate either a successful request of an SDU from the peer data link user entity, or exchange of SDUs with a peer data link user entity. See indicatorHeaderType shown in Figure 7.17.

7.3.1.3.4 AckRequestPdu

Description

AckRequestPdu interface, as shown in Figure 7.16, realizes the PriorityPdu interface from the Common Layer Facilities::PDU Facilities, by binding ControlHeaderType to RequestHeaderType and SDUType to OctetSequence. This PDU interface shall be used when indicating successful or unsuccessful data request or transfer.

Types and Exceptions

- `requestHeaderType`
  (packetIndicator: PacketIndicatorType, correlationID: ULong, useAckServiceInMac: Boolean)
  This header type is passed to the LLC provider by the LLC user to request that an SDU be returned from a peer LLC provider or that SDUs be exchanged between stations using acknowledged connectionless mode data unit exchange procedures. See requestHeaderType shown in Figure 7.16.

7.3.1.3.5 AckConnectionlessLinkComponent

Description

AckConnectionlessLinkComponent, as shown in Figure 7.15, realizes AckConnectionlessLink, ErrorControl, QualityOfServiceConnectionless, FlowControlSignaling, and LocalLinkManagement interfaces. With those relationships, this component provides functionality to control parameters that are related to acknowledged connectionless link establishment and management. After the connection is established, data can be transferred using AckIndicatorPdu, AckRequestPdu, and AckReplyPdu interfaces for acknowledged connectionless communication scenario. ErrorControl is realized for detecting and reporting errors in the reception or transmission. Acknowledged connectionless link component may realize facilities for segmentation and reassembly of protocol data, as well as concatenation and padding of PDU's to match the protocol specification. Every logical link is referenced by a ConnectionID that describes the port(s) that the link is bound to. Local link management interface establishes the links and bounds them to vertical (internal) streams. A component realizing the AckConnectionlessLink API can have a user role, a provider role, or both; depending on the waveform scenario.

Constraints

AckConnectionlessLinkComponent shall provide one ControlPort, at least one input DataControl port, and at least one output DataControl port.

7.3.1.4 Connection Link Package

This package provides facilities to provide connection mode communication for LLC layer. The connection mode is circuit switched and supports data transfer in streams. The connection-mode service is characterized by four phases of communication: local management, connection establishment, data transfer, and connection release. Local management functionality is provided by the local management package defined earlier. The rest of the functionality is defined in this package.

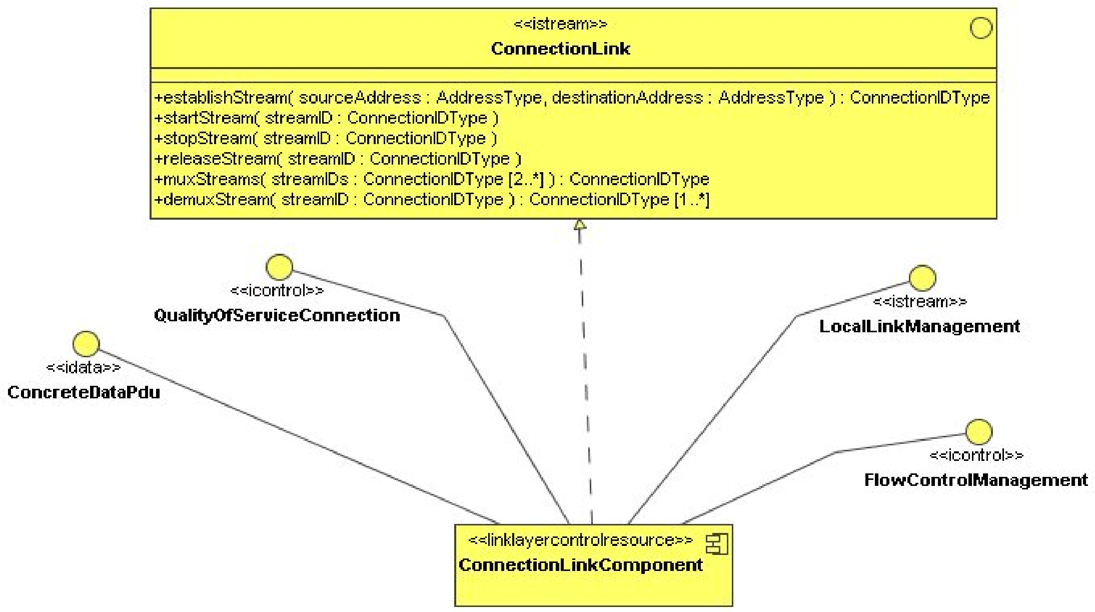

<<istream>>
ConnectionLink
+establishStream(sourceAddress : AddressType, destinationAddress : AddressType) : ConnectionIDType
+startStream(streamID : ConnectionIDType)
+stopStream(streamID : ConnectionIDType)
+releaseStream(streamID : ConnectionIDType)
+muxStreams(streamIDs : ConnectionIDType [2..*]) : ConnectionIDType
+demuxStream(streamID : ConnectionIDType) : ConnectionIDType [1..*]
<<icontrol>>
QualityOfServiceConnection
<<idata>>
ConcreteDataPdu
<<istream>>
LocalLinkManagement
<<icontrol>>
FlowControlManagement
<<linklayercontrolresource>>
ConnectionLinkComponent

Figure 7.18 - ConnectionLink Definition

7.3.1.4.1 ConnectionLink

Description

ConnectionLink interface, as shown in Figure 7.18, provides functionality to control parameters that are related to connection-oriented link establishment, and management as well as enabling and disabling data streams. After the connection is established, data can be transferred using ConnectionLink interface for connection-oriented communication scenario. This interface inherits the

- QualityOfServiceConnection interface for quality of service related facilities,
- IServiceAccessPoint for performing vertical communication tasks,
- FlowControl for flow control interfaces, and
- ITransmission for controlling data and control streams.

Every logical link is referenced by a ConnectionID that describes the SAPs that the link is bound to. Local link management interface establishes the links and binds them to vertical (internal) streams. A component realizing the ConnectionLink API can have a user role, a provider role, or both; depending on the waveform scenario. This interface encompasses all of the possibilities.

Operations

- `establishStream(in sourceAddress : AddressType, in destinationAddress : AddressType, return : ConnectionIDType)`
  This operation allows the LLC service user to initialize a stream.

- `startStream(in streamID : ConnectionIDType)`
  This operation starts data transfer through a previously established stream.
- `stopStream(in streamID : ConnectionIDType)`
  This operation stops data transfer through the given stream.
- `releaseStream(in streamID : ConnectionIDType)`
  The releaseStream operation destroys the stream and releases all of the resources associated with it.
- `muxStreams(in streamIDs: ConnectionIDType [2..n], return ConnectionIDType)`
  This operation multiplexes multiple (two or more) streams into a single stream. This can be done by both the receiving or transmitting entity.
- `demuxStream(in streamID : ConnectionIDType, return ConnectionIDType[1..n])`
  This operation demultiplexes a stream that is composed of multiple data streams.

7.3.1.4.2 ConnectionLinkComponent

Description

ConnectionLinkComponent as shown in Figure 7.18, provides functionality to control parameters that are related to connection oriented link establishment, and management as well as enabling and disabling data streams. After the connection is established, data can be transferred using ConnectionLink interface for connection oriented communication scenario. This interface inherits the

- QualityOfServiceConnection interface for quality of service related facilities,
- FlowControlManagement for flow control interfaces,
- LocalLinkManagement for link management tasks,
- ConnectionLink for managing connection oriented streams, and
- ConcreteDataPdu for transferring data over a stream on a frame-by-frame basis with no control information.

Every logical link is referenced by a ConnectionID that describes the SAPs that the link is bound to. Local link management interface establishes the links and bounds them to vertical (internal) streams. A component realizing the ConnectionLink API can have a user role, a provider role, or both; depending on the waveform scenario. This interface encompasses all of the possibilities.

Constraints

ConnectionLinkComponent shall provide one ControlPort and at least one StreamPort.

7.3.2 Medium Access Control Layer Facilities

This section defines the MAC Layer facilities. MAC Layer provides facilities to upper layers, for both data transmission and control purposes. In that manner, LLC layer, Radio Resource Control (RRC) layer, and other layers that can by-pass the waveform stack to communicate with the MAC layer. MAC layer uses the facilities offered by the physical layer in order to perform medium access control tasks. DLPI specification, OSI reference model X.200e, IEEE 802 series, 3GPP UMTS and GSM specifications were used when defining this interface.

The MAC Facilities define interactions between a user of the MAC layer, termed a Service User, and a MAC layer, termed a Service Provider. The MAC Facilities declare operations that can be invoked by a Service User on a Service Provider for pushing data or sending non-real-time control signals (for configuration purposes). There are also callback operations that can be invoked by a Service Provider on a Service User to report event occurrences. A MAC component communicates with a SAP in order to transfer data and control information between components within the same radio set (Vertical communication). It also provides interfaces to communicate with the remote radio set MAC layer (Horizontal communication).

Due to the complexity and variety of waveforms, defining a single MAC API capable of satisfying all waveform requirements would result in significant processing and memory inefficiencies. For these reasons, most of the main MAC layer interface is defined as a bundle of building blocks as defined in the Common Layer Facilities. Several services provided by a MAC interface are listed as follows:

- Flow control and priority queueing (from Common Layer Facilities::Flow Control Facilities)
- Quality of Service (from Common Layer Facilities::Quality of Service Facilities)
- Error Control (from Common Layer Facilities::Error Control Facilities)
- Measurement and reporting of requested traffic parameters (from Common Layer Facilities::Measurement Facilities)
- Handling of data and control channels
- Scheduling of transmission (Common Radio Facilities::Scheduling Facilities)
- Reordering, Assembly, Multiplexing of data

Figure 7.19 shows an example medium access layer component definition for a CDMA system. The example CDMA parameter type is bound to the MediumAccessParameter definition.

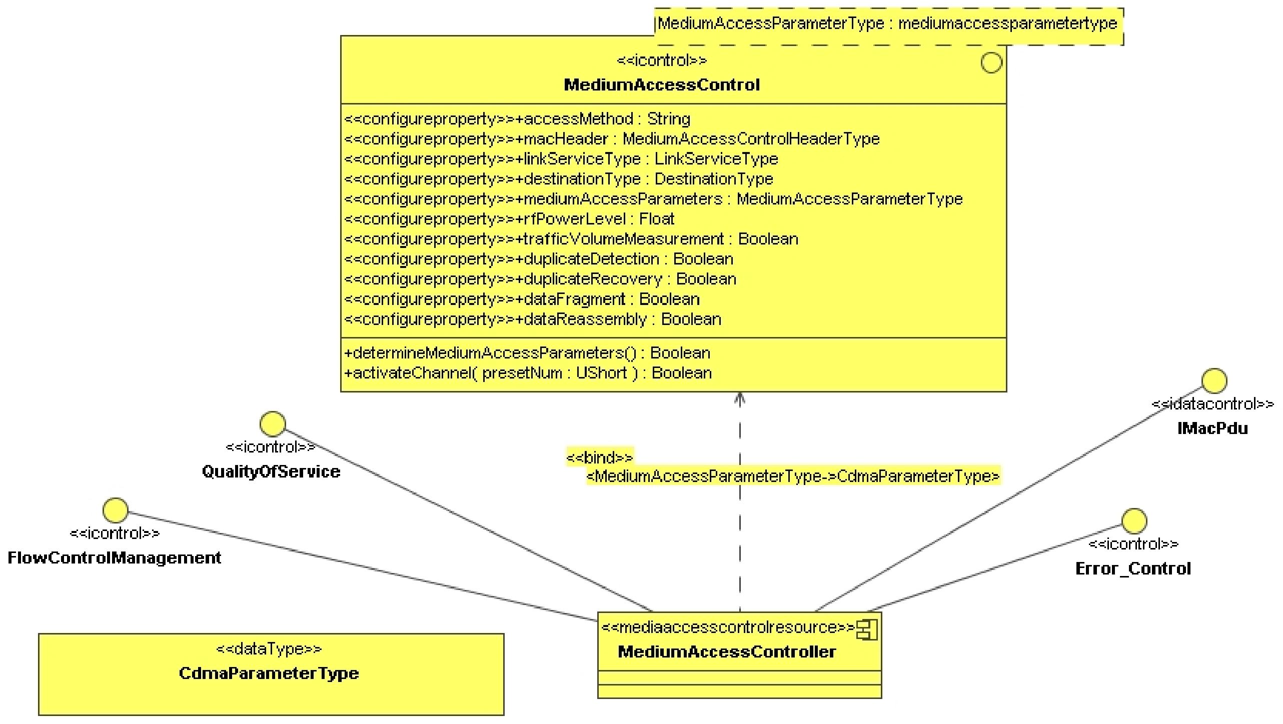

MediumAccessParameterType : mediumaccessparametertype
<<icontrol>>
MediumAccessControl
<<configureproperty>>+accessMethod : String
<<configureproperty>>+macHeader : MediumAccessControlHeaderType
<<configureproperty>>+linkServiceType : LinkServiceType
<<configureproperty>>+destinationType : DestinationType
<<configureproperty>>+mediumAccessParameters : MediumAccessParameterType
<<configureproperty>>+rfPowerLevel : Float
<<configureproperty>>+trafficVolumeMeasurement : Boolean
<<configureproperty>>+duplicateDetection : Boolean
<<configureproperty>>+duplicateRecovery : Boolean
<<configureproperty>>+dataFragment : Boolean
<<configureproperty>>+dataReassembly : Boolean
+determineMediumAccessParameters() : Boolean
+activateChannel(presetNum : UShort) : Boolean
<<idatacontrol>>
IMacPdu
<<icontrol>>
QualityOfService
<<bind>>
<MediumAccessParameterType->CdmaParameterType>
<<icontrol>>
FlowControlManagement
<<icontrol>>
Error_Control
<<mediaaccesscontrolresource>>
MediumAccessController
<<dataType>>
CdmaParameterType

Figure 7.19 - MAC Facilities Overview

7.3.2.1 MediumAccessControl

Description

The MediumAccessControl Facility, as shown in Figure 7.19, is a parameterized interface that defines operations for activating (or selecting) a transport channel and setting the control mode of the medium access parameters. The MediumAccessParameterType is the parameter type that is dependent on the physical medium and the multiple access mechanism. The MediumAccessControl interface is realized by the MediumAccessController component.

For the best software re-use practice, several facilities that are provided by a MAC component are defined as common interfaces in the common layer facilities. Those facilities can be realized by components other than a MAC component, and they may be used in a non-OSI specific waveform layer implementation. The MediumAccessControl interface provides extra MAC layer specific functionality.

Attributes

- `<<configureproperty>> accessMethod: String`
  This attribute defines the access method mechanism MAC component is using. Possible values are defined in the AccessMethodType class definition. This attribute defines the access method mechanism the MAC component is using. Some possible values are: CSMACD (Carrier Sense Multiple Access / Collision Detect), ETHER (Ethernet), ISDN, ATM, LOOP (Software Loopback), etc. For a full listing, see DLPI specification.

- `<<configureproperty>> macHeader: MediumAccessControlHeaderType`
  MacHeader attribute defines the in-band control parameters that will be embedded in the MAC PDU header. Possible fields are defined in the MediumAccessControlHeaderType definition.

- `<<configureproperty>> linkServiceType: LinkServiceType`
  This attribute provides a mechanism for setting the link service type (connectionless, ack connectionless, connection oriented). LLC layer can set this parameter, and request MAC services related to the link type.

- `<<configureproperty>> destinationType: DestinationType`
  This attribute determines whether the destination is a single entity (unicast), multiple entities (multicast), or the entire network of radio sets (broadcast).

- `<<configureproperty>> mediumAccessParameters: MediumAccessParameterType`
  This is an abstract definition of a mediumAccessParameter type. Implementation of this parameter is dependent upon the waveform that implements the MAC component. It may consist of the spreading and scrambling codes on case of WCDMA, allowed time slots for TDMA, frequency bandwidth and hop-set for hopping FDMA, etc.

- `<<configureproperty>> rfPowerLevel: Float`
  rfPowerLevel attribute is used to get/set the RF power output level. MAC component can communicate the RF power level to the physical layer API, if instructed by a higher layer component. Also in certain MAC layer specifications, MAC layer has the ability to extract power control bits from the incoming MAC PDU and set the RF power level accordingly.

- `<<configureproperty>> trafficVolumeMeasurement: Boolean`
  This attribute specifies whether the traffic control measurement is enabled or not. Measurement parameters are communicated to the MAC layer using the Measurement Facilities.

- `<<configureproperty>> duplicateDetection: Boolean`
  duplicateDetection attribute is used to specify whether duplicate PDU detection is enabled in the MAC layer or not.

- `<<configureproperty>> duplicateRecovery: Boolean`
  duplicateRecovery attribute is used to specify whether duplicate PDU recovery is enabled in the MAC layer or not.

- `<<configureproperty>> dataFragment: Boolean`
  dataFragment attribute is used to specify whether data fragmenting is enabled in the transmission chain of the MAC layer or not. If data fragmenting is enabled, related parameters (SDU size, etc.) should be defined in the mediumAccessParameters attribute.

- `<<configureproperty>> dataReassembly: Boolean`
  dataReassembly attribute is used to specify whether data reassembling is enabled in the reception chain of the MAC layer or not. If data reassembly is enabled, related parameters (SDU size, etc.) should be defined in the mediumAccessParameters attribute.

Operations

- `determineMediumAccessParameters (return Boolean)`
  This operation is realized differently depending on the medium the waveform is trying to access to. It can be the ethernet address for an ethernet type connection, or spreading code for UMTS waveform.

- `activateChannel (in presetNum: UShort, return Boolean)`
  Invoked by a Service User on a Service Provider to pass the number of a selected preset channel. The number refers to a preset channel such as the emergency, guard or primary channel. If the Service Provider knows the PresetNum and succeeds to set the corresponding channel it returns the value true, otherwise it returns the value false.

Types and Exceptions

- `<<enumeration>> DestinationType`
  DestinationType class defines the type of destination that is being addressed. Possible values are:
  - UNICAST: For addressing a single recipient.
  - MULTICAST: For addressing multiple recipients.
  - BROADCAST: For addressing entire network of possible recipients.

- `<<abstract>> MediumAccessParameterType`
  Implementation of this class is dependent upon the waveform that implements the MAC component. It may consist of the spreading and scrambling codes on case of WCDMA, allowed time slots for TDMA, frequency bandwidth and hop-set for hopping FDMA, etc.

Semantics

Transmission Security is either implemented by the LLC or the MAC.

7.3.2.2 MacPdu

Description

MacPdu interface, as shown in Figure 7.20, realizes the Pdu interface from the Common Layer Facilities::PDU Facilities, by binding ControlHeaderType to MediumAccessControlHeaderType, and SDUType to OctetSequence.

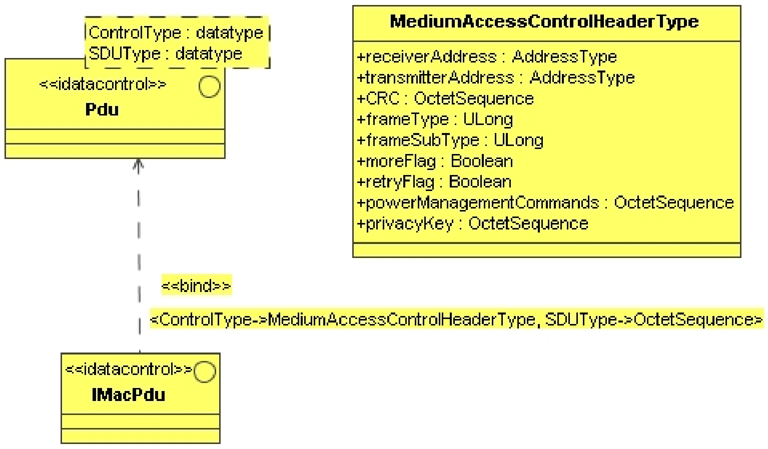

ControlType : datatype
SDUType : datatype
<<idatacontrol>>
Pdu
MediumAccessControlHeaderType
+receiverAddress : AddressType
+transmitterAddress : AddressType
+CRC : OctetSequence
+frameType : ULong
+frameSubType : ULong
+moreFlag : Boolean
+retryFlag : Boolean
+powerManagementCommands : OctetSequence
+privacyKey : OctetSequence
<<bind>>
<ControlType->MediumAccessControlHeaderType, SDUType->OctetSequence>
<<idatacontrol>>
IMacPdu

Figure 7.20 - MacPdu Definition

Types and Exceptions

- `MediumAccessControlHeaderType (receiverAddress : AddressType, transmitterAddress : AddressType, CRC : OctetSequence, frameType : ULong, frameSubType : ULong, moreFlag : Boolean, retryFlag : Boolean, powerManagementCommands : OctetSequence, privacyKey : OctetSequence)`
  MediumAccessControlHeaderType class inherits and extends the ControlHeaderType class as defined in the PDU Facilities. Attributes defined by this class are:
  - receiverAddress: Address information of the receiver. This field may be different than the destinationAddress defined by the control header, in case of retransmission/bridging of a PDU over multiple radio sets before reaching its final destination.
  - transmitterAddress: Address information for the transmitter. This field may be different than the sourceAddress defined by the control header, in case of retransmission/bridging of a PDU over multiple radio sets before reaching its final destination.
  - CRC: cyclic redundancy check code for error checking.
  - frameType: this is an abstract definition and defines the type of frame that is being transferred.
  - frameSubType: this is an abstract definition and defines the sub-type of frame (if it exists) that is being transferred.
  - moreFlag: specifies whether there is more data that will be sent as a part of the current transmission.
  - retryFlag: specifies whether current packet is a retransmission or not.
  - powerManagementCommands: this abstract attribute is used to convey the power management commands to the receiver. Power management is especially required in spread spectrum systems in order to overcome the near/far problem.
  - privacyKey: This key is used in case transmission involves security features.

7.3.2.3 MediumAccessController Component

Description

The MediumAccessController Component, as shown in Figure 7.19, realizes MediumAccessControl, ErrorControl, FlowControlManagement, Measurement, MacPdu, and QualityOfService interfaces. Any extra functionality that is not defined by the interfaces from the common layer facilities package is defined by the MediumAccessControl interface. In order to realize MediumAccessControl interface the implementer shall bind a specific medium access parameter type to MediumAccessParameterType. For example, in a code division multiple access (CDMA) system, users are distinguished by their orthogonal spreading sequences, therefore the MediumAccessParameterType is bound to CdmaParameterType for a CDMA MediumAccessController component as shown in Figure 7.19. Through realizing above mentioned interfaces, this component provides operations for activating (or selecting) a transport channel and setting the control mode of the medium access parameters. MAC Facilities provide Service Users with methods to send non-real-time control and data between software resources and methods to signal the Service User that an event has occurred. Real-time control and signals are communicated via the packet interface. The MediumAccessControlResource is defined in the UML Profile.

Constraints

MediumAccessController component shall provide one ControlPort, at least one input DataControl port and at least one output DataControl port.

8 Platform Specific Model (PSM)

The Common and DataLink Layer Facilities PSM consists of CORBA that is based upon the PIM in Chapter 7. The PIM to PSM transformation rules are not universal rules for creating *any* PSM, but only used for the purpose of this specification. This section defines a non-normative reference PSM. Non-CORBA PSMs may also be fully compliant to this specification as a whole.

The rule set for transforming UML packages, interfaces, types, and exceptions into CORBA constructs are as follows:

1. UML interfaces and interface extensions are map to CORBA interfaces. The CORBA interface names are without the prefix “I” in the interface name as used in the UML profile for SWRadio and in the PIM Facilities.
2. UML attributes with readonly and readwrite map to CORBA attributes in CORBA interfaces.
3. UML attributes with configureproperty, queryproperty, and testproperty do not map to CORBA attributes in CORBA interfaces. Instead XML definitions are used that follow the Property types as defined in UML Profile for SWRadio::Application and Device Components::Properties section.
4. UML classes without operations that are not stereotyped and used for type definitions map to CORBAStruct stereotypes in the CORBA interfaces and modules. The parent classes do not get translated into CORBA types instead the parent class attributes are added to the subclass in the CORBA definition.
5. UML <<datatype>> map to CORBA basic types. Primitive types are mapped to CORBA primitive types and primitive sequence types are mapped to CORBA Typedef of primitive sequence types.
6. UML exceptions and exception extensions map to CORBA exceptions. There is no specializations of exceptions in CORBA so the (UML Profile for SWRadio::Application and Device Components::BaseTypes) SystemException definition does not appear in the generated SWRadio CORBA interfaces but all the specialization exceptions of SystemException are in the SWRadio CORBA interfaces with the same attributes as defined for SystemException.
7. UML attributes that have a cardinality of many [*] map to a CORBA Typedef of sequence types.
8. UML operations and <<optional>> operations map to operations in the SWRadio CORBA interfaces.
9. Transformations are only performed for concrete classes, not for template classes. Concrete classes that bind to template classes are used in the PSM.
10. For Interfaces that reference a component stereotype for a type, the "component" qualifier is removed from the name. For Example, FileManagerComponent would become FileManager as the type for the parameter or attribute.
11. UML attributes with constant stereotype map to CORBA constants in CORBA interfaces.
12. Basic types (e.g., Any, Object) map to CORBA types.

The top most CORBA is called DfSWRadio that maps to the PIM Facilities package. The packages (e.g., Common Layer Facilities) directly beneath PIM Facilities map to CORBA modules but without facilities in there. In some cases these packages have further CORBA modules. This occurs when a package has more than one interface. The DfSWRadio maps to existing IDL definition used in industry, therefore the IDL does not follow all of the OMG CORBA guidelines (e.g., operation, attribute, and parameter names), in order to reduce impact on industry.

Annex A Software Radio Reference Sheet

The Software Radio specification responds to the requirements set by “Request for Proposals for a Platform Independent Model (PIM) and CORBA Platform Specific Model (PSM)” (swradio/02-06-02). The original specification (dtc/05-10-02) has been reorganized into 5 volumes, as follows:

Volume 1. Communication Channel and Equipment

This specification describes a UML profile for communication channel. The profile provides definitions for creating communication channel and communication equipment definitions. The specification also provides radio control facilities and physical layer facilities PIM for defining interfaces and components for managing communication channels and equipment for a radio set or radio system. Along with the profile and facilities is a platform specific model transformation rule set for transforming the communication channel into an XML representation and CORBA interfaces for the radio control facilities.

Volume 2. Component Document Type Definitions

This specification defines the content of a standard set of Data Type Definition (DTD) files for applications, components, and domain and device management. The complete DTD set is contained in Section 7, Document Type Definitions. XML files that are compliant with these DTD files will contain information about the service components to be started up when a platform is power on and information for deploying installed applications.

Volume 3. Component Framework

This specification describes a UML profile for component framework. The profile provides definitions for applications, components (properties, ports, interfaces, etc.), services, artifacts, logical devices, and infrastructure domain management components. In the profile are also library packages that contain interfaces for application, service, logical device, and infrastructure domain management components. Along with the profile is a platform specific model transformation rule set for transforming the profile model library interfaces into CORBA interfaces.

Volume 4. Common and Data Link Layer Facilities

This specification describes a set of facilities PIM for application and component definitions. The set of facilities are common and data link layer facilities that can be utilized in developing waveforms and platform components, which promote the portability of waveforms across Software Defined Radios (SDR). Along with the facilities PIM is a platform specific model transformation rule set for transforming the facilities into CORBA interfaces.

Volume 5. POSIX

This specification defines the application environment profiles for embedded constraint systems, based on Standardized Application Environment Profile - POSIX® Realtime Application Support (AEP), IEEE Std 1003.13-1998.

INDEX