DOCSIS® Guidelines

UML Modeling Guidelines

CM-GL-OSS-UML-V01-180627

RELEASED

Notice

DISCLAIMER

This document is furnished on an "AS IS" basis and neither CableLabs nor its members provides any representation or warranty, express or implied, regarding the accuracy, completeness, noninfringement, or fitness for a particular purpose of this document, or any document referenced herein. Any use or reliance on the information or opinion in this document is at the risk of the user, and CableLabs and its members shall not be liable for any damage or injury incurred by any person arising out of the completeness, accuracy, or utility of any information or opinion contained in the document.

CableLabs reserves the right to revise this document for any reason including, but not limited to, changes in laws, regulations, or standards promulgated by various entities, technology advances, or changes in equipment design, manufacturing techniques, or operating procedures described, or referred to, herein.

This document is not to be construed to suggest that any company modify or change any of its products or procedures, nor does this document represent a commitment by CableLabs or any of its members to purchase any product whether or not it meets the characteristics described in the document. Unless granted in a separate written agreement from CableLabs, nothing contained herein shall be construed to confer any license or right to any intellectual property. This document is not to be construed as an endorsement of any product or company or as the adoption or promulgation of any guidelines, standards, or recommendations.

Document Status Sheet

Document Control Number:	CM-GL-OSS-UML-V01-180627			
Document Title:	UML Modeling Guidelines			
Revision History:	V01 – Released 06/27/18			
Date:	June 27, 2018			
Status:	~~Work in Progress~~	~~Draft~~	Released	~~Closed~~
Distribution Restrictions:	~~Author Only~~	~~CL/Member~~	~~CL/ Member/ Vendor~~	Public

Contents

Figures

Tables

1 SCOPE

1.1 Introduction and Overview

When defining software systems, engineers typically engage in a process called “information modeling”. An information model in software engineering is, according to Wikipedia, “a representation of concepts and the relationships, constraints, rules, and operations to specify data semantics for a chosen domain of discourse. Typically, it specifies relations between kinds of things, but may also include relations with individual things. It can provide sharable, stable, and organized structure of information requirements or knowledge for the domain context.”

Many software tools are available on the market today to facilitate information modeling. They are based on Unified Modeling Language (UML), a standard developed by the Object Management Group (OMG). UML enables developers to specify, visualize, construct, and document artifacts of a software system. It defines a general purpose, graphical modeling language that can be applied to any application domain (e.g., communications) and implementation platforms (e.g., J2EE).

The OMG defines the purpose of the UML as:

- Providing system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes.
- Advancing the state of the industry by enabling object visual modeling tool interoperability. However, to enable meaningful exchange of model information between tools, agreement on semantics and notation is required.

This document examines the following UML diagrams and their associated artifacts:

1. Class Diagrams
2. Object Diagrams
3. Component Diagrams
4. Sequence Diagrams
5. Use Case Diagrams

Additional diagrams and their associated artifacts will be added as they are incorporated into CableLabs specifications.

1.2 Purpose of Document

This guidelines document defines industry best practices, or guidelines, on how to do information modeling based on UML. Although this document doesn’t assume use of a specific tool, it does assume that the reader is knowledgeable about UML in general. Information contained within this document is based on UML 2.5, published in March 2015. DOCSIS information modeling uses [MagicDraw], a commercially available UML modeling tool. MagicDraw v18.3 or later uses the UML2.5 specification.

The purpose of UML modeling guidelines is to ensure that all engineers involved in the information modeling process define UML artifacts using a similar approach. The goal is to create information models that are consistent, comprehensive, and complete. There is also a section describing the process of translating information models into implementable data models such as SNMP MIBs, YANG modules, etc.

Note: Most of the diagrams in this document were created using a “freeware” UML modeling tool that runs [Papyrus] on top of [Eclipse]. This tool is used by many SDO’s today and facilitates collaborative development without the expense of UML tool licenses.

2 REFERENCES

2.1 Informative References

[CCAP-OSSIv3.1]	DOCSIS 3.1 CCAP OSSI Specification, CM-SP-CCAP-OSSIv3.1-I12-180509, May 9, 2018, Cable Television Laboratories, Inc.
[Eclipse]	Eclipse Oxygen Release. Internet: https://www.eclipse.org/oxygen
[FRAGMENTS]	Examples of UML Combined Fragments. Internet: https://www.uml-diagrams.org/sequence-diagrams-combined-fragment.html
[IISOMI 514]	ONF IISOMI 514 UML Modeling Guidelines v1.3.04, November 2nd, 2017.
[IISOMI 531]	ONF IISOMI 531 UML to YANG Mapping Guidelines v1.1.07, June 6, 2018.
[IISOMI UML YANG]	EAGLE UML-Yang Mapping Tool. Internet: https://github.com/OpenNetworkingFoundation/EAGLE-Open-Model-Profile-and-Tools
[IPDR/SSDG]	IPDR Service Specification Design Guide, Version 3.8, TM Forum, October 2009.
[IPDR/SP]	IPDR Streaming Protocol (IPDR/SP) Specification, TMF8000-IPDR-IIS-PS, Version 2.7, TM Forum, November 2011.
[MagicDraw]	MagicDraw R18.5. Internet: https://docs.nomagic.com/display/MD185/MagicDraw+Documentation
[Papyrus]	Papyrus Oxygen Release 3.3.0. Internet: https://www.eclipse.org/papyrus/download.html
[OMG UML]	OMG Unified Modeling Language v2.5, March 2015. Internet: http://www.omg.org/spec/UML/2.5.
[RFC 1157]	IETF RFC 1157. Simple Network Management Protocol (SNMP), May 1990.
[RFC 2578]	IETF RFC 2578, Structure of Management Information Version 2 (SMIv2), April 1999.
[RFC 6020]	IETF RFC 6020, YANG - A data modeling language for the Network Configuration Protocol (NETCONF), October 2010.
[RFC 6241]	IETF RFC 6241, Network Configuration Protocol (NETCONF), June 2011
[RFC 7231]	IETF RFC 7231, Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, June 2014
[RFC 8040]	IETF RFC 8040, RESTCONF Protocol, January 2017
[RFC 8259]	IETF RFC 8259, The Javascript Object Notation (JSON) Data Interchange Format, December 2017
[R-OSSI]	Remote PHY OSS Interface Specification, CM-SP-R-OSSI-I09-180509, May 9, 2018, Cable Television Laboratories, Inc.
[R-PHY]	Remote PHY Specification, CM-SP-R-PHY-I10-180509, May 9, 2018, Cable Television Laboratories, Inc.
[W3XSD1.0]	XML Schema Part 1: Structures Second Edition, W3C Recommendation 28, October 2004.

2.2 Reference Acquisition

CableLabs Specifications:

- Cable Television Laboratories, Inc., 858 Coal Creek Circle, Louisville, CO 80027 USA; Phone +1 303-661-9100; Fax +1 303-661-9199; Internet: http://www.cablelabs.com

Internet Engineering Task Force:

- IETF, Internet Engineering Task Force (IETF) Secretariat, 48377 Fremont Blvd., Suite 117, Fremont, California 94538, USA; Phone: +1-510-492-4080, Fax: +1-510-492-4001; http://www.ietf.org/

Object Management Group Specifications:

- Object Management Group, 109 Highland Ave, Needham, MA 02494 USA; Phone +1 781-444-0404; Fax +1 781-444-0320; info@omg.org; Internet: http://www.omg.org/

Open Networking Foundation Specifications:

- Open Networking Foundation, 1000 El Camino Real, Suite 100, Menlo Park, CA 94025 USA; info@opennetworking.org; Internet: http://www.opennetworking.org/

3 TERMS AND DEFINITIONS

This document uses the following terms:

Actor	In a Use Case, an actor specifies a role played by a user or any other entity that interacts with the subject
Array	An indexed and bounded collection type. It corresponds to an array type in SQL 1999.
Association	A relationship between classifiers which is used to show that instances of classifiers could be either linked to each other or combined logically or physically into some aggregation.
Artifact	Physical units such as files, scripts, binary executables, etc.
Attribute	Contains a property of an object class.
Bag	A collection which may contain duplicate elements and is unordered.
Class	Classes (also called “object classes”) are used to convey a static representation of an entity, including properties and attributes.
Class Diagram	Show the classes of the system, their interrelationships (including inheritance, aggregation, and association), and the operations and attributes of the classes.
Combined Fragment	A combined fragment is used to group sets of messages together to show conditional flow in a sequence diagram.
Component	A unit of software that provides interfaces that contain operations and consumes interfaces from other software.
Component Diagram	Shows the structural relationships between components in a system.
Data Model	Data Models usually specify items at a lower level of abstraction and include protocol specific constructs. The level of abstraction does not depend on the language being used (e.g., XML, IDL, YANG). Such languages allow modeling both at high and low (i.e. detailed) levels. Note that if the Information Model is too generic, then the derived Data Models may be somewhat undefined, and lacking in interoperability.
Directionality	Represents direction in which an association is navigated. It may be uni-directional or bi-directional.
Extensible Markup Language	A universal file format for storing and exchanging structured data. The CCAP configuration file is created in XML and has a specific schema, generated from a set of YANG modules, which are a physical implementation of an object model created to describe CCAP configuration.
Hypertext Transfer Protocol	An application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers.
Informal Inter-SDO Open Model Initiative	An open source project founded by UML model designers from various SDOs like ETSI NFV, ITU-T, MEF, ONF and TM Forum. The goal is to develop guidelines and tools for a harmonized modeling infrastructure that is not specific to any SDO, technology or management protocol and can then be used by all SDOs.
Information Model	An Information Model (as compared to a Data Model) is an abstraction and only provides a high level view of things of interest (i.e., information) to the business. It aids in understanding the scope and breadth of the business, rather than the depth. An Information Model is a way of representing and structuring information that has advantages over other common artifacts such as a glossary, descriptive document, database or source code. A common Information Model will streamline the processes associated with information exchange, both within a business (e.g., Enterprise) and between the business and its external stakeholders.
Interface	Components are connected to other components through interfaces. A component may either provide an interface, or consume an interface, or both.
Internet Protocol Detail Records	Provides information about Internet Protocol (IP)-based service usage and other activities that can be used by Operational Support Systems (OSS) and Business Support Systems (BSS).
Javascript Object Notation	A lightweight, text-based, language-independent data interchange format. It was derived from the ECMAScript Programming Language Standard. JSON defines a small set of formatting rules for the portable representation of structured data.
Management Information Base	A database of device configuration and performance information which is acted upon by SNMP.
Navigable	Describes the need for an object to access another object by "navigating across the link."

NETCONF	Provides mechanisms to install, manipulate, and delete the configuration of network devices. It uses an Extensible Markup Language (XML)-based data encoding for the configuration data as well as the protocol messages. The NETCONF protocol operations are realized as remote procedure calls (RPCs).
Object Diagram	The Object diagram displays instances of classifiers and links (instances of associations) between them.
Ordered Set	A collection of entities in which the entities have an order, and each entity is unique.
Port	A port is often used to help expose required and provided interfaces of a component.
pyang	A YANG validator, transformer, and code generator, written in Python, that is used to generate the CCAP schema file from the CCAP YANG modules
RESTCONF	An HTTP-based protocol that provides a programmatic interface for accessing data defined in YANG, using the datastore concepts defined in the Network Configuration Protocol (NETCONF).
Sequence Diagram	A sequence diagram is a form of interaction diagram which shows objects as lifelines running down the page, with their interactions over time represented as messages drawn as arrows from the source lifeline to the target lifeline.
Set	A collection of entities in which duplicate are dropped and the entities are not ordered.
Simple Network Management Protocol	An Internet Standard protocol for collecting and organizing information about managed devices on IP networks and for modifying that information to change device behavior.
Subject	A subject of a use case defines and represents boundaries of a software system, physical system or device, subsystem, component or even single class in relation to the requirements gathering and analysis.
Use Case	A set of actions that some system should or can perform in collaboration with one or more external users of the system.
YANG	A data modeling language for t he NETCONF network configuration protocol.

4 ABBREVIATIONS AND ACRONYMS

This document uses the following abbreviations and acronyms.

BSS	Business Support Systems
CBD	Component-Based Development
HTTP	Hypertext Transfer Protocol
IISOMI	Informal Inter-SDO Open Model Initiative
IP	Internet Protocol
IPDR	Internet Protocol Detail Record
IPDR/SP	IPDR Streaming Protocol
JSON	Javascript Object Notation
LCC	Lower Camel Case
MIB	Management Information Base
NETCONF	Network Configuration Protocol
OMG	Object Management Group
ONF	Open Networking Foundation
OSS	Operational Support Systems
RPC	Remote Procedure Calls
SOA	Service-Oriented Architecture
SDO	Standards Developing Organization
SNMP	Simple Network Management Protocol
SMIv2	Structure of Management Information version 2
UCC	Upper Camel Case
UML	Unified Modeling Language
XML	Extensible Markup Language
XSD	XML Schema Definition

5 INFORMATION MODELING USING UML

5.1 Class Diagrams

Class diagrams show the classes of the system, their interrelationships (including inheritance, aggregation, and association), and the operations and attributes of the classes. Class diagrams are used for a wide variety of purposes, including both conceptual/domain modeling and detailed design modeling.

5.1.1 Diagram Notation

Figure 1 illustrates a Class Diagram:

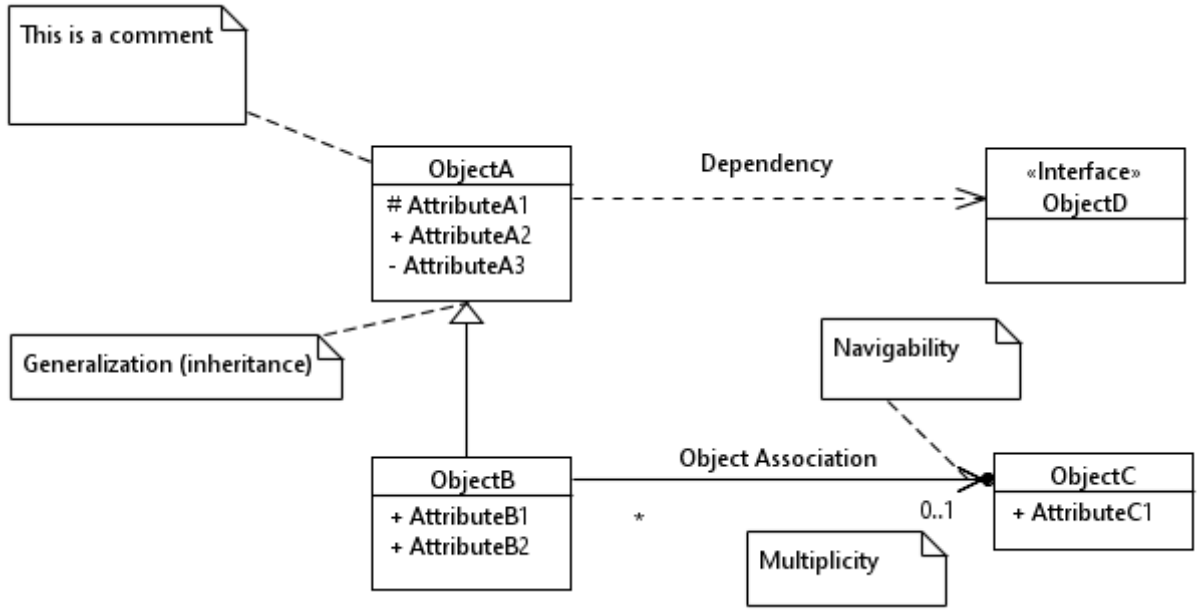

Figure 1 - UML Class Diagram Notation

The concepts presented in the diagram are defined in the sections that follow.

5.1.2 Classes

Classes (also called “object classes”) are used to convey a static representation of an entity, including properties and attributes; i.e., data model, the static part of the model. In other words, defining object classes is about defining entities and not the operations acting on the entities.

Classes are generally represented by a square box with three compartments. The top compartment contains the class name (used here as the object name). The middle compartment contains the list of attributes. The bottom compartment contains the list of operations, or methods. For the purposes of these guidelines, the methods section of the class box is not used (suppressed) and the implementation level details of the attributes are omitted.

An object class has the following properties:

Name	Must use Upper Camel Case (UCC). Each class in the model has a unique name.
Documentation	Contains a brief description of the class. The documentation field in some tools is specified as “Applied comments”
Superclass(es)	Inheritance and multiple inheritances may be used to deal with shared properties.
Abstract	Indicates if the object class can be instantiated or is just used for inheritance. If it is abstract, it cannot be instantiated.

5.1.3 Attributes

Attributes contain the properties of an object class. Note that the roles of navigable association ends become an attribute in the class at the other associated end when this association end is owned by the classifier. Note: The association end can also be owned by the association itself in which case it does not become an attribute. See the section on associations for further information.

The graphical notation used for object class attributes is:

|<list of stereotypes> <visibility> <attribute name>: <attribute type> [<multiplicity>] = <default value>

Note: When no default is relevant, or no default is defined, the "=" is not shown.

The visibility notation which precedes the attribute name and is one of the following symbols:

'+' public (default)

'-' private

'#' protected

If the above notation is omitted from the attribute, the default of public is implied. For the purposes of these guidelines, the protected visibility generally refers to indexes of MIB tables, schema instances, etc.

An attribute has the following properties:

Name	Usually it is recommended to use Lower Camel Case (LCC). However, DOCSIS has a tradition of using UCC for attribute names. Therefore, DOCSIS will continue to follow UCC for attribute names. Also, attribute names typically do not include any of the object name elements since this would cause duplication when the object and attributes are realized in SNMP. The name is unique across all attribute names in the inheritance tree.
Documentation	Contains a brief description of the attribute.
Ordered	For a multi-valued multiplicity; this specifies whether the values in an instantiation of this attribute are sequentially ordered; default is false.
Unique	For a multi-valued multiplicity, this specifies if the values of this attribute instance are unique (i.e., no duplicate attribute values); default is true.

When Unique is true (the default), the collection of values may not contain duplicates. When Ordered is true (false being the default) the collection of values is ordered in a meaningful way. In combination these two allow the type of a property to represent a collection as shown in Table 1:

Table 1 - Collection Types

Ordered	Unique	Collection Type
false	true	Set
true	true	Ordered Set
false	false	Bag
true	false	Array

Type	Refers to a data type. The data type can be a simple type such as UnsignedInt or a defined data type such as EnumBits. DOCSIS data types are defined in Annex F of the [CCAP-OSSIv3.1] specification and Annex C of the [R-PHY] Specification.
Multiplicity	(*, 1, 1..*, 0..1, …) Defines the number of values the attribute can simultaneously have. * is a list attribute with 0, one or multiple values; 1 attribute has always one value; 1..* is a list attribute with at least one value; 0..1 attribute may have no or at most one value; Default value is 1. Other values are possible; e.g., "2..17".
Access	Indicates the attributes accessibility (as mapped to an SNMP object for example). Example values include "key", "read-only", "read-write", and "read-create".

Type Constraints	Lists constraints on the normal data type specified in the "Type". If there are no defined constraints for the attribute, this column is empty. The example below for AttributeA1 lists a constraint on the UnsignedInt Type where the range starts from 1 instead of normally starting from 0 for an UnsignedInt
Units	Lists units for the attribute or "N/A" if the attribute does not have units.
Default	Contains the default value for the attribute or "N/A" if the attribute does not have a default value or in cases where the attribute's description defines rules for the initialization value.

The sections following Table 2 below are attribute descriptions which might include behavioral requirements or references.

Table 2 - ObjectA Example Table Layout

Attribute Name	Type	Access	Type Constraints	Units	Default
AttributeA1	UnsignedInt	key	1..4294967295	N/A	N/A
AttributeA2	AdminString	read-write	SIZE (1..15)	N/A	N/A
AttributeA3	UnsignedByte	read-create		seconds	60

5.1.3.1 AttributeA1

AttributeA1 is a key defined for...

NOTE: Objects which represent a table (in an SNMP MIB realization) and have N number of instances need to include at least one "key" attribute which is used to denote the instance or id. Key attributes are typically denoted with a protected visibility whereas all other attributes are denoted with a public visibility.

5.1.3.2 AttributeA2

AttributeA2 is ...

NOTE: Persistence requirements are documented at the object level, not at the attribute level.

5.1.3.3 AttributeA3

AttributeA3 is ...

5.1.4 Comment

A Comment in a class diagram is a textual annotation attached to any element. This is represented as a note symbol with a dashed line connecting the note with the element.

5.1.5 Relationships Between Classes

There are four types of relationships between classes:

1. *Association* A relationship between classes which is used to show that instances of classes could be either linked to each other or combined logically or physically into some aggregation
2. *Generalization* A taxonomic relationship between a more general class and a more specific class. Each instance of the specific class is also an instance of the general class. The specific class inherits the features of the more general class.
3. *Dependency* A relationship that implies the class at the source end of the relationship has some sort of dependency on the class at the target (arrowhead) end of the relationship.
4. *Realization* A relationship between two classes, in which one class (the client) realizes (implements or executes) the behavior that the other class (the supplier) specifies.

5.1.5.1 Associations

The following examples show the various kinds of association notations. The actual notation may differ from one tool to another, but the concepts described below are applicable to all UML associations.

Figure 2 below shows a bi-directional *navigable* association where each object class has a pointer to the other. The role name becomes the name of the corresponding attribute in the class. The role name must exist and begin with an underscore. I.e., in the example: ClassA will have an attribute named "_classB" pointing to ClassB and vice versa.

Figure 2 - Bidirectional Association Relationship Notation

Next, Figure 3 shows a unidirectional association (shown with an open arrow at the target object class) where only the source object class has a pointer to the target object class and not vice-versa.

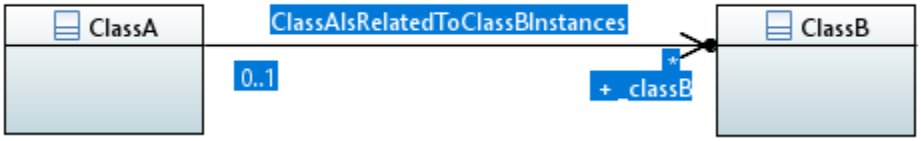

Figure 3 - Unidirectional Association Relationship Notation

Figure 4 displays a non-navigable association where each object class does not have a pointer to the other; i.e., such associations are just for illustration purposes.

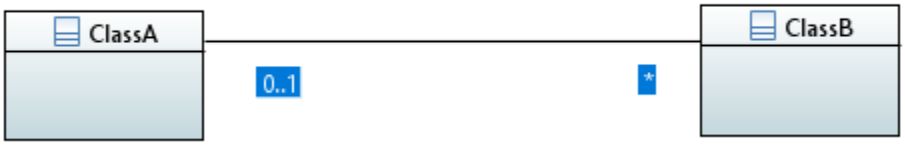

Figure 4 - Non-Navigable Association Relationship Notation

An aggregation is a special type of association in which objects are assembled or configured together to create a more complex object. Aggregation protects the integrity of an assembly of objects by defining a single point of control called aggregate, in the object that represents the assembly. The example in Figure 5 below defines a shared aggregation. In a shared aggregation, if the aggregate is deleted, all its parts remain.

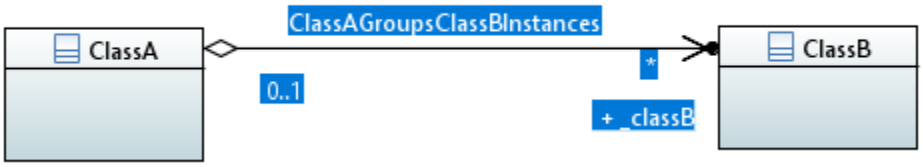

Figure 5 - Shared Aggregation Association Relationship Notation

A composite aggregation association is a strong form of aggregation that requires a part instance be included in at most one composite at a time. In a composite aggregation, if the aggregate is deleted, all its parts are deleted as well; i.e., the lifecycle of ClassB is tied to the lifecycle of ClassA.

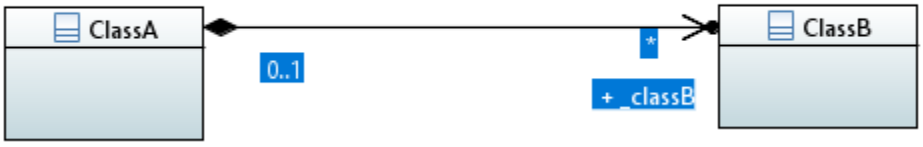

Figure 6 - Composite Aggregation Association Relationship Notation

An association has the following properties:

Name	Follows UCC and is unique across all association names defined in the whole model. The format is "<Class1Name><VerbPhrase><Class2Name>" where the verb phrase creates a sequence that is *readable and meaningful. For example, "ClassADefinesClassB".*
Documentation	Contains a brief description of the association.

Member Ends of an Association may contain the following information:

Name	Follows LCC with an underscore "_" prefix and identifies the role that the object plays at this end of the association. Only navigable association ends have role names and follow the definitions made for attributes.
Owner	Either "Association" or "Classifier". Arrow notation is used to denote association end navigability. All class-owned association ends (classifier) are navigable. By convention, all association-owned ends (association) in the model are not navigable. Note: The Owner of a navigable Member End must be the Classifier to become an attribute in the object class.
Navigable	True" or "False"
Aggregation	• none • shared • composite
Multiplicity	Identifies the number of object instances that can participate in an instance of the association.

5.1.5.2 Generalizations

A generalization indicates a relationship in which one class (the child) inherits from another class (the parent). That implies that all the attributes that are defined in the ParentClass are also defined in the ChildClass. The ChildClass will most likely have other attributes that will distinguish it from the ParentClass.

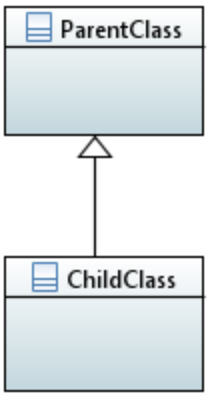

Figure 7 - Generalization Notation

5.1.5.3 Dependencies

Dependency is a directed relationship which is used to show that a UML class requires, needs or depends on another class for specification or implementation. Because of this, dependency is called a supplier - client relationship, where supplier provides something to the client, and thus the client is in some sense incomplete while semantically or structurally dependent on the supplier element(s). Modification of the supplier may impact the client elements.

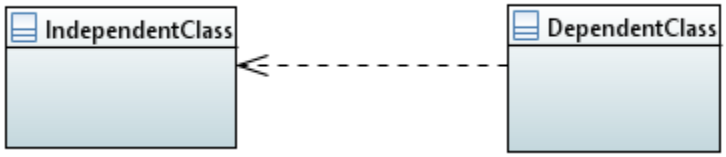

Figure 8 - Dependency Notation

5.1.5.4 Realizations

A *realization* relates two classes at different semantic levels (as an example an analysis class and a design class.) It denotes the implementation of the functionality defined in one class by another class. To show the relationship in UML, a broken line with an unfilled solid arrowhead is drawn from the class that defines the functionality of the class that implements the function.

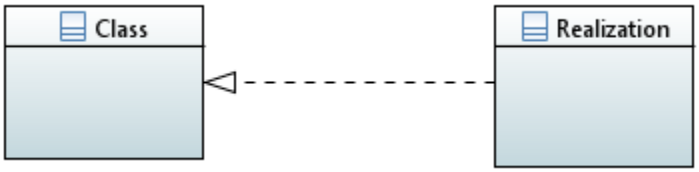

Figure 9 - Realization Notation

5.2 Object Diagrams

The notion of “Object Diagrams” no longer exists in the UML 2.5 specification. They were defined in previous UML specifications. However, this section is included as many of the tools on the market today still provide Object Diagrams. The Object diagram displays instances of classifiers and links (instances of associations) between them. In many tools these are referred to as Instance Diagrams.

5.2.1 Diagram Notation

Figure 10 shows an Object Instance Diagram for an instantiation (myObjectA) of ObjectA

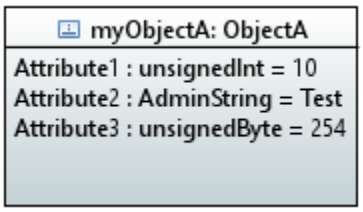

Figure 10 - Object Instance Diagram for ObjectA

Instances of objects are represented in UML 2.5 by “Instance Specifications”. An instance specification specifies the existence of an entity in a modeled system and completely or partially describes the entity. In this case the instance is based on a class. The compartments within an Instance Specification are called “Slots”. One defines the value that an attribute takes in an instance by specifying it in the Slot.

5.3 Component Diagrams

The UML component diagram shows the structural relationships between “components” in a system. This type of diagrams is used in Component-Based Development (CBD) to describe systems with Service-Oriented Architecture (SOA). Components are autonomous, encapsulated units within a system or subsystem that provide, or require, one or more “interfaces”. Components may contain “artifacts” which are physical units such as files, scripts, binary executables, etc. This document often refers to resources within a component.

5.3.1 Component

In UML2, a component is merely a specialized version of the class concept. A component is drawn as a rectangle with optional compartments stacked vertically. A high-level, abstracted view of a component in UML 2 can be modeled as just a rectangle with the component's name and the component stereotype text and/or icon. The component stereotype's text is «component» and the component stereotype icon is a rectangle with two smaller rectangles protruding on its left side. Components in this document are illustrated as shown in Figure 11:

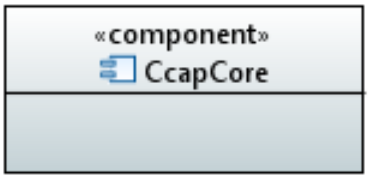

Figure 11 - Example of Component

A component has the following properties:

Name	Must use Upper Camel Case (UCC) and is unique across all component names defined in the whole model.
Documentation	Contains a brief description of the component.
Abstract	Indicates whether a component is instantiated or used for illustration purposes.
Required (Interfaces)	Indicates the “interfaces” consumed by this component. There may be multiple interfaces.
Provided (Interfaces)	Indicates the “interfaces” provided by this component. There may be multiple interfaces.

5.3.2 Artifacts

An artifact is a classifier that represents some physical entity, a piece of information that is used or is produced by a software development process, or by deployment and operation of a system. An artifact is denoted by a rectangle showing the artifact name, the «artifact» keyword and a document icon. An example of an artifact is given in Figure 12 below:

Figure 12 - Example of Artifact

5.3.3 Interfaces

Interfaces represent the places where the groups of classes in the component communicate with other system components. An interface is the definition of a collection of one or more methods, and zero or more attributes, ideally one that defines a cohesive set of behaviors. Interfaces are defined by extending symbols from the component box.

- Provided interface is shown by a straight line from the component box with an attached circle. These represent the interfaces where a component produces information used by the required interface of another component. This is often referred to as the "lollipop" notation.
- Required interface is shown by a straight line from the component box with an attached half circle. These represent the interfaces where a component requires information to perform its proper function. This is often referred to as the "socket" notation.

The interfaces provided represent the formal contract of services the component provides to its consumers/clients. Figure 13 illustrates Component A that "provides" interface1, and Component B that has interface 1 as a "required" interface. Note: the tool used to create the diagram doesn't show the "required" interface as a line with a half circle. It is displayed as a straight line to the interface.

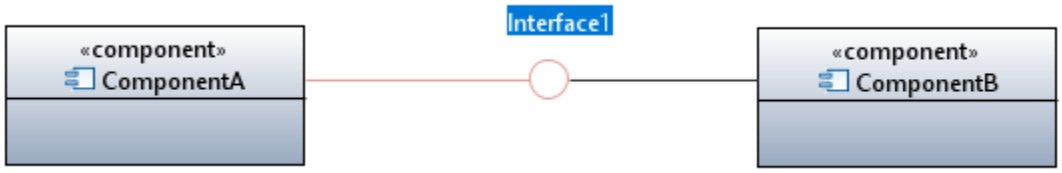

Figure 13 - Example of Interfaces

5.3.4 Dependencies

Components may be dependent on other components. A dependency is a relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation. Dependencies are drawn using dashed arrows. Figure 14 illustrates a dependency between components:

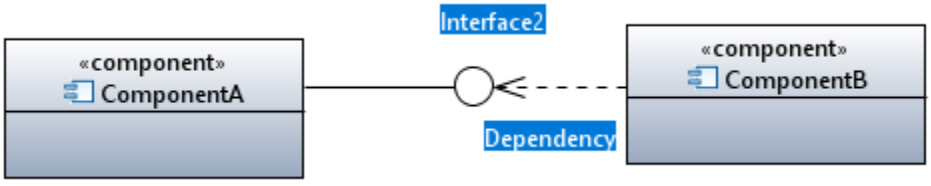

Figure 14 - Example of Dependency

5.3.5 Ports

A port is often used to help expose required and provided interfaces of a component. It specifies a separate interaction point between the component and the environment and can be used to group together several interfaces. A Port may specify the services a classifier provides (offers) to its environment as well as the services that a classifier expects (requires) of its environment. Ports are represented using a square along the edge of the system or a component.

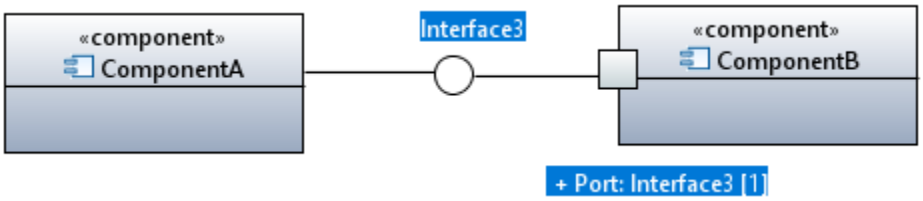

Figure 15 - Example of a Port

5.4 Sequence Diagrams

A sequence diagram is a form of interaction diagram which shows objects as lifelines running down the page, with their interactions over time represented as messages drawn as arrows from the source lifeline to the target lifeline. Sequence diagrams are good at showing which objects communicate with which other objects; and what messages trigger those communications.

Sequence diagrams are used by software developers and business people alike to understand requirements for a new system or to document an existing process. They are useful to:

- Represent the details of a UML use case.
- Model the logic of a sophisticated procedure, function, or operation.
- See how tasks are moved between objects or components of a process.
- Plan and understand the detailed functionality of an existing or future scenario.

5.4.1 Lifelines

Lifeline notation elements are placed across the top of the sequence diagram. Lifelines represent either roles or object instances that participate in the sequence being modeled. They indicate the object's presence over time. Lifelines are drawn as a box with a dashed line descending from the center of the bottom edge. The lifeline's name is placed inside the box.

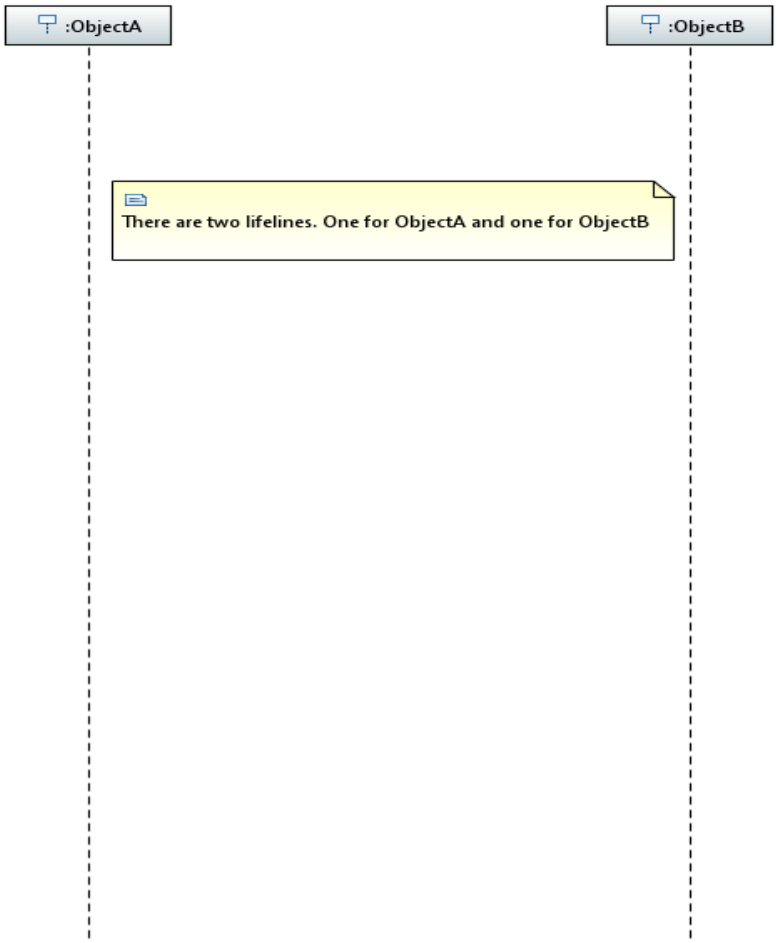

Figure 16 - Lifelines

5.4.2 Activation or Execution Occurrence

Activation boxes represent the time an object needs to complete a task. When an object is busy executing a process or waiting for a reply message, a thin gray (or in some tools opaque) rectangle is placed vertically on its lifeline.

Figure 17 - Activation or Execution Occurrence

5.4.3 Messages

Messages are arrows that represent communication between objects. The various types of messages are described in Figure 18.

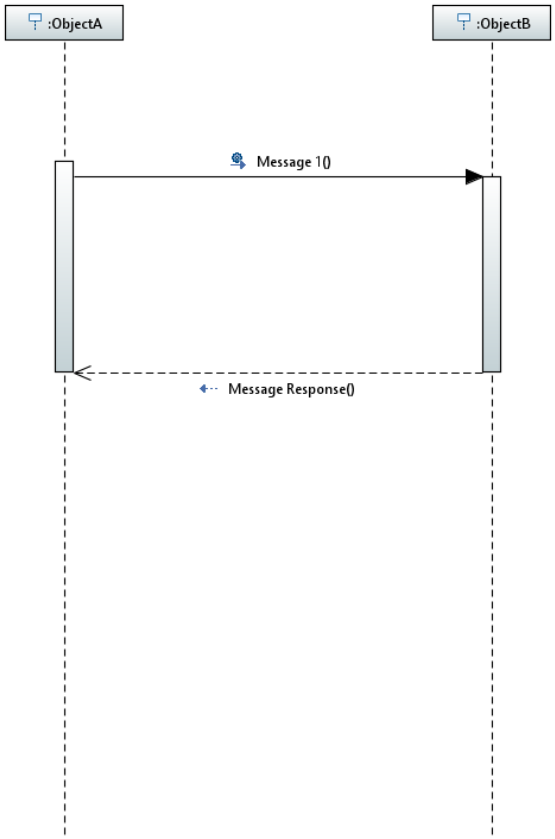

Figure 18 - Messages

Synchronous Message

A synchronous message requires a response before the interaction can continue. It's usually drawn using a line with a solid arrowhead pointing from one object to another.

Asynchronous Message

Asynchronous messages don't need a reply for interaction to continue. Like synchronous messages, they are drawn with an arrow connecting two lifelines; however, the arrowhead is usually open and there's no return message depicted.

Reply or Return Message

A reply message is drawn with a dotted line and an open arrowhead pointing back to the original lifeline.

Self-Message

A message an object sends to itself, usually shown as a U- shaped arrow pointing back to itself.

Create Message

This is a message that creates a new object. Similar to a return message, it's depicted with a dashed line and an open arrowhead that points to the rectangle representing the object created.

Delete Message

This is a message that destroys an object. It can be shown by an arrow with an x at the end.

5.4.4 Combined Fragments

A combined fragment is used to group sets of messages together to show conditional flow in a sequence diagram. The UML 2.5 specification identifies 11 interaction types for combined fragments. Three basic combined fragments of the eleven are described below. For examples of notation of combined fragments, see [FRAGMENTS].

Alternatives

Alternatives are used to designate a mutually exclusive choice between two or more message sequences. Alternatives allow the modeling of the classic "if then else" logic. An alternative combination fragment element is drawn using a frame. The word "alt" is placed inside the frame's namebox. The larger rectangle is then divided into what UML 2 calls operands. Operands are separated by a dashed line. Each operand is given a guard to test against, and this guard is placed towards the top left section of the operand on top of a lifeline. If an operand's guard equates to "true," then that operand is the operand to follow.

Option

The option combination fragment is used to model a sequence that, given a certain condition, will occur; otherwise, the sequence does not occur. An option is used to model a simple "if then" statement. The option combination fragment notation is similar to the alternation combination fragment, except that it only has one operand and there never can be an "else" guard. To draw an option combination, you draw a frame. The text "opt" is placed inside the frame's namebox, and in the frame's content area the option's guard is placed towards the top left corner on top of a lifeline. Then the option's sequence of messages is placed in the remainder of the frame's content area.

Loops

Loops are used to model a repetitive sequence. The loop combination fragment is very similar in appearance to the option combination fragment. You draw a frame, and in the frame's namebox the text "loop" is placed. Inside the frame's content area, the loop's guard is placed towards the top left corner, on top of a lifeline. Then the loop's sequence of messages is placed in the remainder of the frame's content area. In a loop, a guard can have two special conditions tested against in addition to the standard Boolean test. The special guard conditions are minimum iterations written as "minint = the number" (e.g., "minint = 1") and maximum iterations written as "maxint = the number" (e.g., "maxint = 5"). With a minimum iterations guard, the loop must execute at least the number of times indicated, whereas with a maximum iteration guard the number of loop executions cannot exceed the number.

5.5 Use Case Diagrams

Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors).

Use case diagrams are used to specify:

- (external) requirements, required usages of a system under design or analysis (subject) - to capture what the system is supposed to do;
- the functionality offered by a subject – what the system can do;
- requirements the specified subject poses on its environment - by defining how the environment should interact with the subject so that it will be able to perform its services.

5.5.1 Diagram Notation

The following diagram illustrates use cases associated with Fault Management. The actor is the “Operator” and the subject is the “Network Device”.

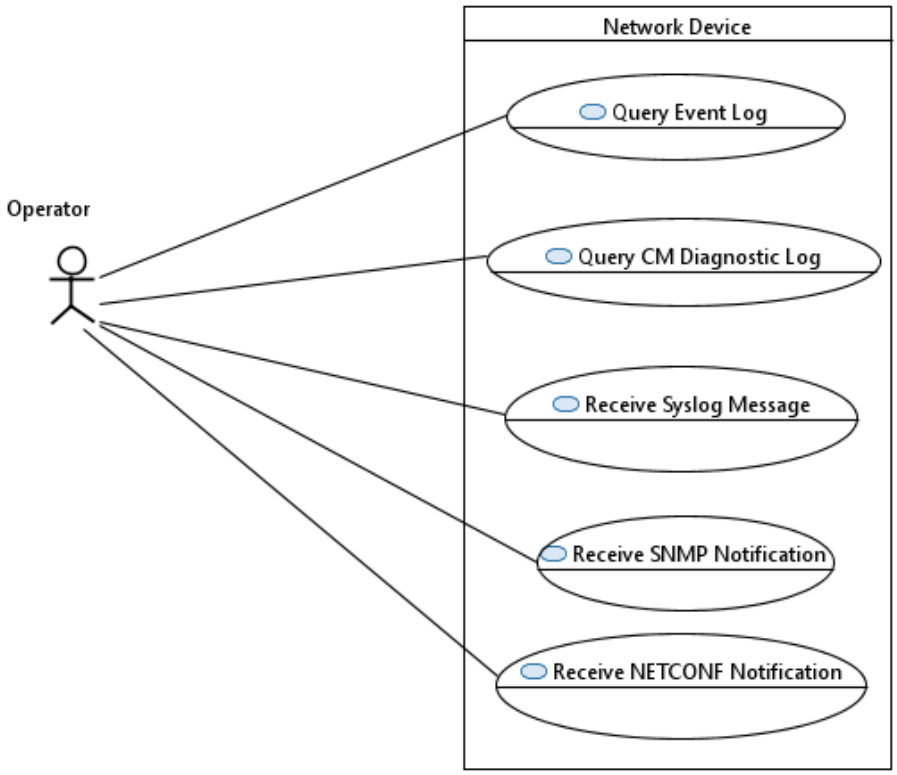

Figure 19 - Fault Management Use Case Diagram

5.5.2 Subject

A subject of a use case defines and represents boundaries of a software system, physical system or device, subsystem, component or even single class in relation to the requirements gathering and analysis.

A subject (sometimes called a system boundary) is represented by a rectangle with subject's name, associated keywords, and stereotypes in the top left corner.

5.5.3 Actor

An actor specifies a role played by a user or any other entity that interacts with the subject. An actor models a type of role played by an entity that interacts with the subject (e.g., by exchanging signals and data), but which is external to the subject.

Actors may represent roles played by human users, external hardware, or other subjects. Actors do not necessarily represent specific physical entities but merely particular facets (i.e., “roles”) of some entities that are relevant to the specification of its associated use cases. A single physical instance may play the role of several different actors and a given actor may be played by multiple different instances.

Actors are represented by a "stick man" icon with the name of the actor above or below the icon. Actor names should follow the capitalization and punctuation guidelines for classes. The names of abstract actors should be shown in italics. All actors have names. Actors may also be represented by icons.

5.5.4 Use Case

A use case is a sequence of actions or events steps that typically define the interactions between a role (actor) and a system (subject) to achieve a goal. A use case provides some observable and valuable result to the actors or other stakeholders of the system.

Every use case has a name. A use case is represented as an ellipse containing the name of the use case.

5.5.5 Relationships

In UML, a relationship is a connection between model elements. A UML relationship is a type of model element that adds semantics to a model by defining the structure and behavior between the model elements. Use case diagrams have two types of important relationships, “include” and “extend”.

An “include” relationship is a relationship in which one use case (the base use case) includes the functionality of another use case (the inclusion use case).The include relationship supports the reuse of functionality in a use-case model.

An “extend” relationship is used to specify that one use case (extension) extends the behavior of another use case (base). This type of relationship reveals details about a system or application that are typically hidden in a use case.

6 INFORMATION MODEL TRANSLATIONS

The Information Model approach is based on an object oriented modeling approach well known in the industry for capturing requirements and analyzing the data in a protocol independent representation. This approach defines requirements with use cases to describe the interactions between the operations support systems and the network element. The management information is represented in terms of managed objects (i.e., UML classes) along with their attributes and the interactions between these encapsulated objects (or also referred to as entities in some representations). The diagrams developed to capture these managed objects and their attributes and associations are UML Class Diagrams. The collection of UML Class Diagrams and Use Case Diagrams are referred to as the DOCSIS 3.1 Information Models. With the introduction of several new, complex features in DOCSIS 3.0 and DOCSIS 3.1 architecture, and the operator needs for a more proactive and efficient approach to management information, information modeling methodologies offer the ability to reuse the same definitions when new protocols are introduced in the future.

While the purpose of an information model is for capturing system and architectural designs via visual diagrams, data models represent real-world implementations of information models. For example, data models specify how data can be encoded and sent over a communications network using a network management protocol. Once an information model is developed and standardized, data models may be derived from the information model. Data models become an implementation realization of the information model, often for a specific interface of an open system.

Managed objects are represented in a protocol specific form referred to as a management data model. The management data models when using SNMP [RFC 1157] are described using the Structure of Management Information Version 2 (SMIv2) [RFC 2578] and the design of these models is determined by the capabilities of the protocol. The management data models when using NETCONF [RFC 6241] are described using the YANG data modeling language [RFC 6020]. The management data models when using RESTCONF [RFC 8040] are described using YANG data modeling language and encoded using XML or JSON [RFC 8259]. The management data models when using IPDR/SP [IPDR/SP] are described using the IPDR Service Definition Schemas [IPDR/SSDG. The management data models when using XML configuration file download are described using XML Schema [W3XSD1.0].

6.1 UML-to-YANG

IISOMI is developing specifications for formal translation of UML to YANG [IISOMI 531]. That document defines the guidelines for mapping protocol-neutral UML information models to YANG data schemas. The UML information model to be mapped has to be defined based on the IISOMI UML Modeling Guidelines [IISOMI 514].

In parallel, a tool which automates the mapping from UML-to-YANG is being developed in the Open Source SDN community. The current draft version of the tool is available on GitHub [IISOMI UML YANG].

6.2 UML-to-SMIv2

There is no formal mapping specification from UML to SMIv2 (for SNMP MIBs). The translation can be done manually since the UML class diagrams generally contain all required information needed to author the corresponding MIB elements.

6.2.1 Converting Classes

Converting UML class diagrams into SNMP MIBs is relatively straightforward. Classes without keys (and are not contained within classes defining keys) map to SMIv2 scalar groups while classes with keys (or are contained within a class defining keys) map to SMIv2 tables.

6.2.2 Example Mappings

The following examples illustrates mapping UML classes into MIB elements. Refer to *CCAP Core RPD Control Information Model* in [R-OSSI] for class definition details.

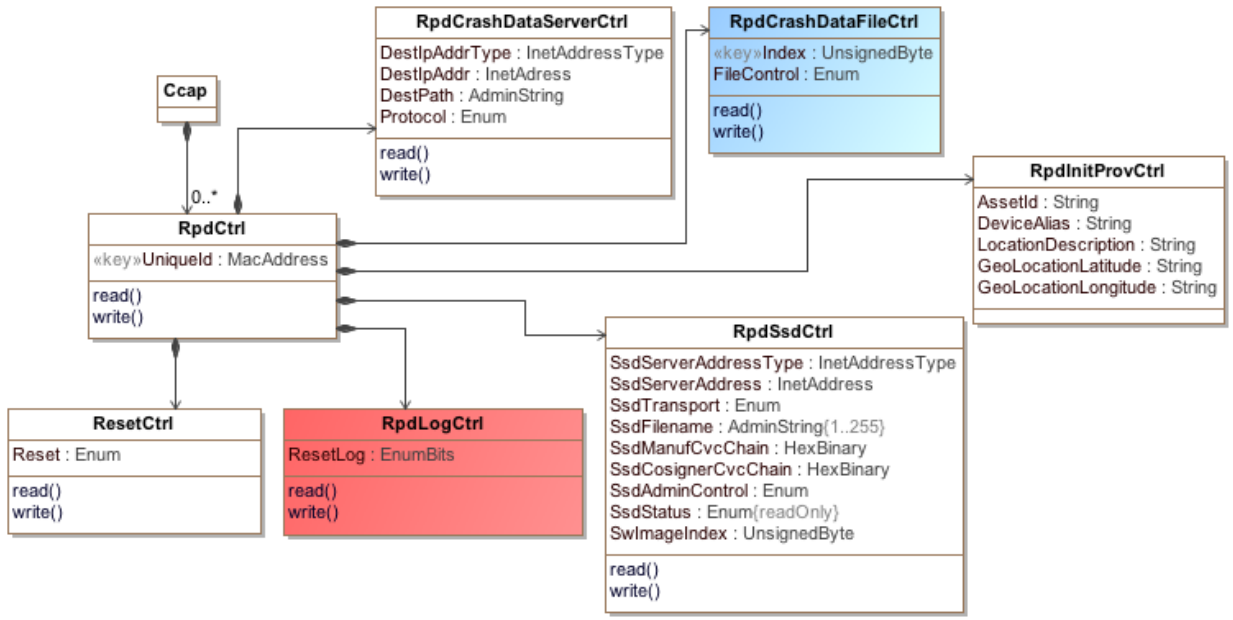

Figure 20 - Example Class Diagram

The red highlighted class in Figure 20 can be mapped into an SNMP MIB Table as shown below. Note that while the class does not contain a key, it is contained within the RpdCtrl class which has a key. Therefore the MIB inherits this key structure through the UML containment association.

```
-- ---------------------------------------------------------------------
-- RPD Log Control Table
-- ---------------------------------------------------------------------
docsRphyCtrlRpdLogCtrlTable OBJECT-TYPE
        SYNTAX      SEQUENCE OF DocsRphyCtrlRpdLogCtrlEntry
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "This table provides RPD log management functions."
        ::= { docsRphyCtrlRpdMibObjects 2 }


docsRphyCtrlRpdLogCtrlEntry OBJECT-TYPE
        SYNTAX     DocsRphyCtrlRpdLogCtrlEntry
        MAX-ACCESS not-accessible
        STATUS     current
        DESCRIPTION
            "The conceptual row of docsRphyRpdLogCtrlTable."
        INDEX { docsRphyRpdDevInfoUniqueId }
        ::= { docsRphyCtrlRpdLogCtrlTable 1 }


DocsRphyCtrlRpdLogCtrlEntry ::= SEQUENCE
       {
       docsRphyCtrlRpdResetLog              BITS
       }

docsRphyCtrlRpdResetLog OBJECT-TYPE
                        SYNTAX      BITS {
                      localEventLog(0),
                      eventPendingQueue(1)
                        }
        MAX-ACCESS  read-write
        STATUS      current
        DESCRIPTION
            "This attribute controls resetting an RPD's local event log
```

```
            and/or the pending event queue. Reading this value returns
            the value zero. Setting an individual bit to 1 resets the
            corresponding RPD log or queue."
        ::= { docsRphyCtrlRpdLogCtrlEntry 1 }
```

The blue highlighted class in Figure 20 can be mapped into an SNMP MIB Table as shown below. The class contains a key, and it is contained within the RpdCtrl class which has a key. Therefore the MIB inherits this key structure through the UML containment association while also defining the key within the class.

```
-- ---------------------------------------------------------------------
-- RPD Crash Data File Control Table
-- ---------------------------------------------------------------------
docsRphyCtrlRpdCrashDataFileCtrlTable OBJECT-TYPE
        SYNTAX      SEQUENCE OF DocsRphyCtrlRpdCrashDataFileCtrlEntry
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "This table provides CCAP Core control attributes needed to
            initiate an RPD upload of a crash analysis file to the Server"
        ::= { docsRphyCtrlRpdMibObjects 4 }


docsRphyCtrlRpdCrashDataFileCtrlEntry OBJECT-TYPE
        SYNTAX     DocsRphyCtrlRpdCrashDataFileCtrlEntry
        MAX-ACCESS not-accessible
        STATUS     current
        DESCRIPTION
            "The conceptual row of docsRphyCtrlRpdCrashDataFileCtrlTable."
        INDEX { docsRphyRpdDevInfoUniqueId,
                        docsRphyCtrlRpdCrashDataFileCtrlIndex }
        ::= { docsRphyCtrlRpdCrashDataFileCtrlTable 1 }


DocsRphyCtrlRpdCrashDataFileCtrlEntry ::= SEQUENCE
       {
       docsRphyCtrlRpdCrashDataFileCtrlIndex                          Unsigned32,
       docsRphyCtrlRpdCrashDataFileCtrlFileControl
            RphyCrashDataFileControlType
       }

docsRphyCtrlRpdCrashDataFileCtrlIndex OBJECT-TYPE
        SYNTAX      Unsigned32
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "This key attribute uniquely identifies a particular
             device crash analysis file."
        ::= { docsRphyCtrlRpdCrashDataFileCtrlEntry 1 }

docsRphyCtrlRpdCrashDataFileCtrlFileControl OBJECT-TYPE
        SYNTAX      RphyCrashDataFileControlType
        MAX-ACCESS  read-write
        STATUS      current
        DESCRIPTION
                 "This attribute controls the action taken by the device regarding
             the file selected by Index attribute. When a value is written to
             this attribute for a given instance of the RpdCrashDataFileCtrl
             object, the device is required to take that action on the associated
             crash data file.

             Possible actions are:

             other(1): This value is returned when the attribute is read.
                               This value is not writeable.

                 upload(2): If the FileControl attribute of the
                               RpdCrashDataFileCtrl object is set to 'upload',
                               the CCAP Core MUST signal the RPD to initiate an upload
                               to the server with the parameters specified in the
                               'DestIpAddr', 'DestIpAddrType', and 'DestPath'
```

```
                        attributes of the RpdCrashDataServerCtrl object.

            cancelUpload(3): If the FileControl attribute of the
                        RpdCrashDataFileCtrl object is set to 'cancelUpload',
                        the CCAP Core MUST signal the RPD to cancel a
                        pending upload.

            deleteFile(4): If the FileControl attribute of the
                        RpdCrashDataFileCtrl object is set to 'deleteFile',
                        the CCAP Core MUST signal the RPD to delete the file
                        from its memory.

            uploadAndDelete(5): If the FileControl attribute of the
                        RpdCrashDataFileCtrl object is set to 'uploadAndDelete',
                        the CCAP Core MUST signal the RPD to upload the
                        selected file and upon successful completion of the
                        upload, delete the file from its memory."

    ::= { docsRphyCtrlRpdCrashDataFileCtrlEntry 2 }
```

6.3 UML-to-IPDR

There is no formal mapping specification from UML to IPDR Service Definition Schemas. The translation can be done manually since the UML class diagrams generally contain all required information needed to author the corresponding schemas.

6.4 UML-to-XML

There are many tools available for translating UML to XML. However, DOCSIS currently maps UML to YANG and then translates YANG into XSD using pyang. Refer to *Converting YANG to XSD* in [CCAP-OSSIv3.1] for additional information. Direct mapping from UML to XML is left for further study.

6.5 UML-to-JSON

Direct mapping from UML to JSON is left for further study.

Appendix I Acknowledgements

CableLabs wishes to heartily thank those individuals and their organizations that contributed to drafting this document.

Contributor	Company Affiliation
Steve Burroughs	CableLabs
Brian Hedstrom	OAM Technology Consulting, LLC

Steve Burroughs, Lead Architect, CableLabs

* * *