OPEN SOURCE SDN

IISOMI 531
UML to YANG Mapping Guidelines

Version 1.0
September 20, 2016

Work in progress!

Disclaimer

Open Networking Foundation
2275 E. Bayshore Road, Suite 103, Palo Alto, CA 94303
http://www.opennetworking.org
http://opensourcesdn.org

Content

List of Figures

List of Tables

Document History

Version	**Date**	**Description of Change**
1.0	Sept. 20, 2016	Initial version.

1 Introduction

This 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 UML Modeling Guidelines defined in [7].
In parallel, a tool which automates the mapping from UML → YANG is being developed in the Open Source SDN community. The current draft version of the tool is available on Github [9]. A video which introduces the UML → YANG mapping tool is provided in [10].
The mapping tool is using YANG Version 1.0 (RFC 6020).

Note:
Mapping in the reverse direction from YANG to UML is possible for the class artifacts but has some issues to be taken into account; see also section 9.

Note:
This version of the guidelines is still a work in progress! Known open issues are marked in yellow and by comments.

2 References

[1] RFC 6020 “YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)”
[2] Guidelines for Authors and Reviewers of YANG Data Model Documents (draft-ietf-netmod-rfc6087bis)
[3] A Guide to NETCONF for SNMP Developers
(by Andy Bierman, v0.6 2014-07-10)
[4] YANG Central (http://www.yang-central.org)
[5] NetConf Central (http://www.netconfcentral.org)
[6] YANG patterns (https://tools.ietf.org/html/draft-schoenw-netmod-yang-pattern)
[7] IISOMI 514 “UML Modeling Guidelines Version 1.2”
(https://community.opensourcesdn.org/wg/EAGLE/dashboard)
[8] OpenModelProfile (https://github.com/OpenNetworkingFoundation/EAGLE-Open-Model-Profile-and-Tools/tree/OpenModelProfile)
[9] EAGLE UML-Yang Mapping Tool
(https://github.com/OpenNetworkingFoundation/EAGLE-Open-Model-Profile-and-Tools/tree/UmlYangTools)
[10] Video to introduce the UML to YANG mapping tool
Youtube: https://www.youtube.com/watch?v=6At3YFrE8Ag&feature=youtu.be
Youku: http://v.youku.com/v_show/id_XMTQ4NDc2NDg0OA==.html
[11] RFC 7950 “The YANG 1.1 Data Modeling Language”

3 Abbreviations

App	Application

C	Conditional
CM	Conditional-Mandatory
CO	Conditional-Optional
DN	Distinguished Name
DS	Data Schema
DSCP	Differentiated Services Codepoint
IM	Information Model
JSON	JavaScript Object Notation
M	Mandatory
MAC	Media Access Control
NA	Not Applicable
O	Optional
OF	Open Flow
Pac	Package
ro	read only
RPC	Remote Procedure Call
rw	read write
SDN	Software Defined Network
SMI	Structure of Management Information
UML	Unified Modeling Language
URI	Uniform Resource Identifier
UUID	Universally Unique Identifier
XOR	Exclusive OR
XMI	XML Metadata Interchange
XML	Extensible Markup Language
YANG	"Yet Another Next Generation".

4 Overview

4.1 Documentation Overview

This document is part of a suite of guidelines. The location of this document within the documentation architecture is shown in Figure 4.1 below:

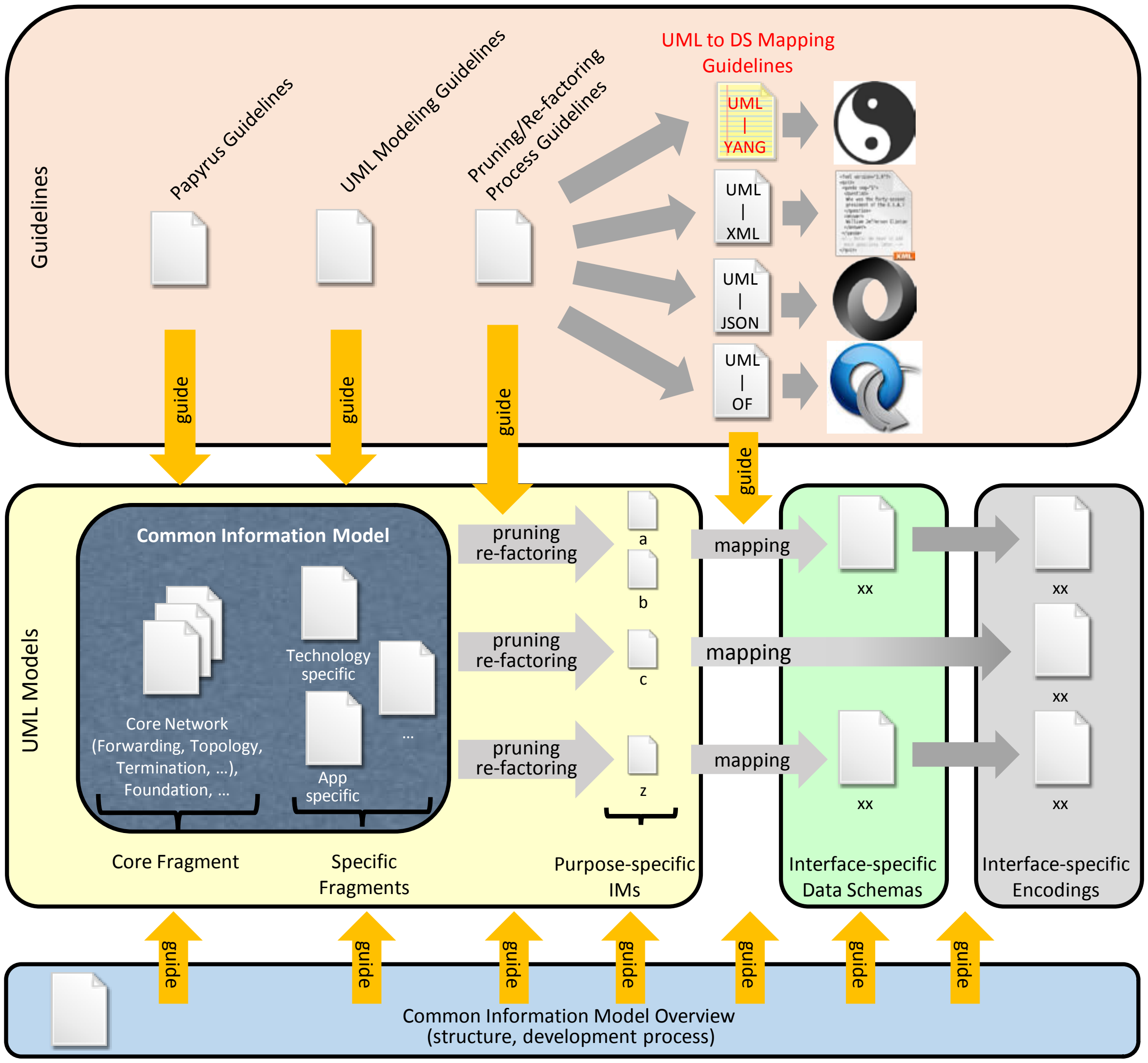

Guidelines
Papyrus Guidelines
UML Modeling Guidelines
Pruning/Re-factoring Process Guidelines
UML to DS Mapping Guidelines
UML | YANG
UML | XML
UML | JSON
UML | OF
guide
UML Models
Common Information Model
Core Network (Forwarding, Topology, Termination, …), Foundation, …
Technology specific
App specific
…
Core Fragment
Specific Fragments
pruning re-factoring
a
b
c
z
Purpose-specific IMs
mapping
xx
Interface-specific Data Schemas
Interface-specific Encodings
Common Information Model Overview (structure, development process)

Figure 4.1: Specification Architecture

5 Mapping Guidelines

The mapping rules are defined in table format and are structured based on the UML artifacts defined in [7]. Two tables are created for every UML artifact. The first table shows the mapping to YANG for the UML artifacts defined in [7]. The second table shows the potential mapping of the remaining YANG substatements which have not been covered in the first table.
Example mappings are shown below the mapping tables.

Open issues are either marked in yellow and/or by comments.
General mapping issues are defined in section 5.11.

5.1 Mapping of Classes

Classes are mapped in two steps. In the **first step**, all classes are mapped to “grouping” statements. In the **second step** the groupings of all non-abstract classes are “instantiated” in a “list” or “container” statement at which

- real classes having/inheriting at least one attribute identified as “partOfObjectKey” will be instantiated into a “list” statement
- real classes **not** having/inheriting any attribute identified as “partOfObjectKey” will be instantiated into a “container” statement.

Note: For a top-level grouping which augment a YANG tree node the mapping tool must NOT generate a top-level list statement; i.e., no second step in this case.

Table 5.1: Class Mapping
(Mappings required by currently used UML artifacts)

Class → “grouping” statement → “list“ or “container” statement		
UML Artifact	YANG Artifact	Comments
documentation “Applied comments” (carried in XMI as “ownedComment”)	“description” substatement	Multiple “applied comments” defined in UML, need to be collapsed into a single “description” substatement.
superclass(es)	"grouping" statement	Concrete superclasses are then mapped to container/list which uses these groupings.
abstract	"grouping" statement	It is possible that the superclass or abstract class contains the key attribute for the instantiated subclass. When the subclass is instantiated the key value must be identified from within the used grouping of the superclass.
object identifier Note: Attributes used as object identifier are defined in UML by the attribute property “partOfObjectKey”.	list::“key” substatement	It is possible that the superclass or abstract class contains the key attribute for the instantiated subclass.
object identifier list Does not appear in the UML which is used for mapped to YANG.		The splitting of a list attribute (marked as key) into a single key attribute and an additional list attribute will be done in UML during Pruning&Refactoring. I.e., the mapping tool will never get a list attribute which is part of the object identifier.
objectCreationNotification [YES/NO/NA]	“notification” statement	See section 5.8 Goes beyond the simple “a notification has to be sent”; a tool can construct the signature of the notification by reading the created object.
objectDeletionNotification [YES/NO/NA]	“notification” statement	See section 5.8 Goes beyond the simple “a notification has to be sent”; a tool can construct the signature of the notification by providing the object identifier of the deleted object (i.e., not necessary to provide the attributes of the deleted object).

Class → "grouping" statement → "list" or "container" statement		
UML Artifact	YANG Artifact	Comments
multiplicity >1 on association to the class	list::"min-elements" and "max-elements" substatements	min-elements default = 0 max-elements default = unbounded mandatory default = false
«Reference»	"reference" substatement	
«Example»	Ignore Example elements and all composed parts	
lifecycleState	"status" substatement or "description" substatement	See section 5.10.
Proxy Class: See section 6.6. XOR: See section 6.3	"choice" substatement	
support	"if-feature" substatement	Support and condition belong together. If the "support" is conditional, then the "condition" explains the conditions under which the class has to be supported.
condition		
operation	"action" substatement	YANG 1.0 supports only rpc → add prefix to the rpc name; i.e., objectClass::rpc; action requires YANG 1.1
Conditional Pac	"container" statement with "presence" substatement	See section 6.2.

Table 5.2: Class Mapping
(Mappings for remaining YANG substatements)

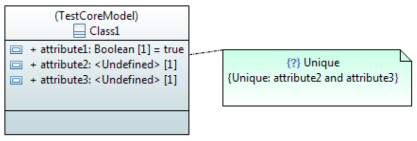
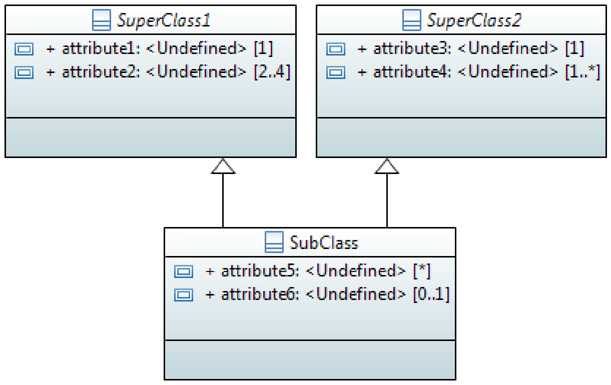
Class → "grouping" statement → "list" or "container" statement		
UML Artifact	YANG Artifact	Comments
	~~"config" substatement~~	not relevant to class
not needed now	~~"must" substatement~~	not relevant to class
not needed now	~~list::"ordered-by" substatement~~	not relevant to class ordered-by default = system
constraint property  (TestCoreModel) Class1 + attribute1: Boolean [1] = true + attribute2: <Undefined> [1] + attribute3: <Undefined> [1] {?} Unique {Unique: attribute2 and attribute3}	list::"unique" substatement	UML is not able to define a group of attributes to be unique as YANG can do using the "unique" substatement. See also EAGLE Issue #66.
{<constraint>}	"when" substatement	

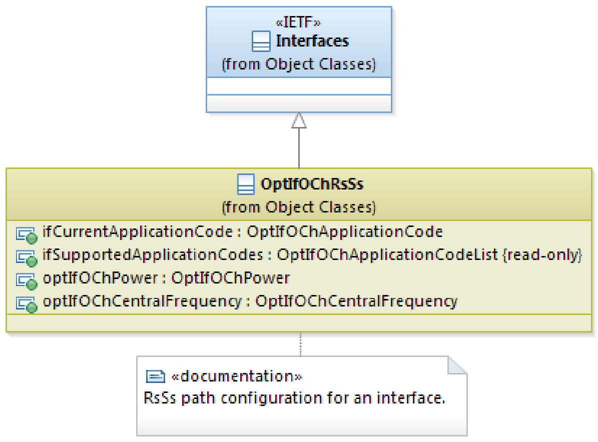
Table 5.3: Class Mapping Example

<table>
<tr>
<td>
SuperClass1
+ attribute1: <Undefined> [1]
+ attribute2: <Undefined> [2..4]
SuperClass2
+ attribute3: <Undefined> [1]
+ attribute4: <Undefined> [1..*]
SubClass
+ attribute5: <Undefined> [*]
+ attribute6: <Undefined> [0..1]

</td>
<td>

```
grouping SuperClass1 {
    leaf attribute1 {
        …
        mandatory true;
    }
    leaf-list attribute2 {
        …
        min-elements 2;
        max-elements 4;
    }
}
grouping SuperClass2 {
    leaf attribute3 {
        …
        mandatory true;
    }
    leaf-list attribute4 {
        …
        min-elements 1;
    }
}
grouping SubClass {
    leaf-list attribute5 {
        …
    }
    leaf attribute6 {
        …
    }
}
container SubClass {
    …
    uses SubClass
    uses SuperClass1;
    uses SuperClass2;
}
```

</td>
</tr>
</table>


«IETF»
Interfaces
(from Object Classes)
OptIfOChRsSs
(from Object Classes)
ifCurrentApplicationCode : OptIfOChApplicationCode
ifSupportedApplicationCodes : OptIfOChApplicationCodeList {read-only}
optIfOChPower : OptIfOChPower
optIfOChCentralFrequency : OptIfOChCentralFrequency
«documentation»
RsSs path configuration for an interface.

from IETF draft-dharini-netmod-g-698-2-yang-04:

```
module ietf-opt-if-g698-2 {
    namespace "urn:ietf:params:xml:ns:yang:ietf-opt-if-
    g698-2";
    prefix ietf-opt-if-g698-2;

    import ietf-interfaces {
        prefix if;
    }

    …

augment "/if:interfaces/if:interface" {
    description "Parameters for an optical interface";
    container optIfOChRsSs {
        description "RsSs path configuration for an
        Interface";
        container ifCurrentApplicationCode {
            description "Current Application code of the
                        interface";
        uses optIfOChApplicationCode;
        }

        container ifSupportedApplicationCodes {
            config false;
            description "Supported Application codes of
                        the interface";
            uses optIfOChApplicationCodeList;
        }

        uses optIfOChPower;
        uses optIfOChCentralFrequency;
    }
}
…
```

5.2 Mapping of Attributes

Table 5.4: Attribute Mapping
(Mappings required by currently used UML artifacts)

Attribute → "leaf" (for single-valued attribute) or "leaf-list" (for multi-valued attribute) statement		
UML Artifact	YANG Artifact	Comments
documentation "Applied comments" (carried in XMI as "ownedComment")	"description" substatement	Multiple "applied comments" defined in UML, need to be collapsed into a single "description" substatement.

Attribute → “leaf” (for single-valued attribute) or “leaf-list” (for multi-valued attribute) statement		
UML Artifact	YANG Artifact	Comments
type	“type” substatement (built-in or derived type)	
readOnly	“config” substatement (false)	config default = true
isOrdered	leaf-list::“ordered-by” substatement ("system" or "user") The leaf-list::"description" substatement may suggest an order to the server implementor.	ordered-by default = system
isUnique	No unique sub-statement in leaf-list. Can this be defined via a single leaf argument in the unique sub-statement of the containing list statement?	Only relevant for multi-valued attributes. YANG 1.0: The values in a leaf-list MUST be unique. YANG 1.1: In configuration data, the values in a leaf-list MUST be unique. I.e., YANG 1.1 allows non-unique values in non-configuration leaf-lists.
Multiplicity (carried in XMI as lowerValue and upperValue)	leaf only: “mandatory” substatements [0..1] => no mapping needed; is leaf default [1] => mandatory substatement = true leaf-list only: “min-elements” and “max-elements” substatements [0..x] => no mapping needed; is leaf-list default [1..x] => min-elements substatement = 1 [0..3] => max-elements substatement = 3	min-elements default = 0 max-elements default= unbounded mandatory default = false
defaultValue	"default" substatement	If a default value exists and it is the desired value, the parameter does not have to be explicitly configured by the user. When the value of “defaultValue” is “NA”, the tool ignores it and doesn’t print out “default” substatement.
isInvariant	“extension” substatement → ompExt:isInvariant	See extensions YANG module in section 8.2.
valueRange	For string typed attributes: “pattern”, and/or “length” substatement of “type” substatement. For integer and decimal typed attributes: “range” substatement of “type” substatement. For all other typed attributes and for string or integer or decimal typed attributes where the UML definition is not compliant to YANG: “description” substatement.	The tool should provide a warning at the output of the mapping process notifying when one or more UML valueRange definitions are contained in the description substatement of the corresponding leaf or leaf-list. When the value of “valueRange” is “null”, “NA”, “See data type”, the tool ignores it and doesn’t print out “range” substatement.

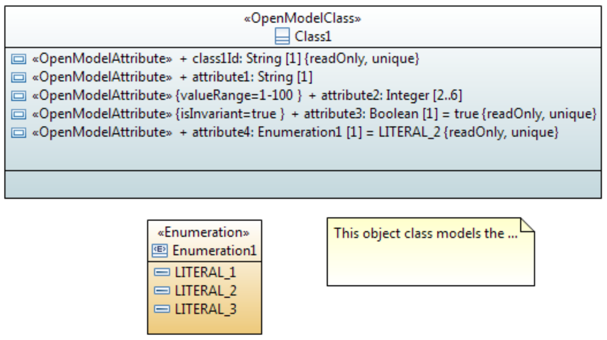
passedByReference	if passedByReference = true →
	type leafref {
	path “/<object>/<object identifier>"
	if passedByReference = false →
	either “list” statement (key property, multiple instances) or “container” statement (single instance)
		Relevant only to attributes that have a class defined as their type.

Attribute → "leaf" (for single-valued attribute) or "leaf-list" (for multi-valued attribute) statement		
UML Artifact	YANG Artifact	Comments
partOfObjectKey >0	list::"key" substatement	It is possible that the (abstract) superclass contains the key attribute for the instantiated subclass.
unit	"units" substatement	
«Reference»	"reference" substatement	
«Example»	Ignore Example elements and all composed parts	
lifecycleState	"status" substatement or "description" substatement	See section 5.10.
support	For conditional support only: "if-feature" substatement	Support and condition belong together. If the "support" is conditional, then the "condition" explains the conditions under which the class has to be supported.
condition	"when" substatement if condition can be formalized as XPath expression (i.e., it is conditioned by the value of another attribute)	

Table 5.5: Attribute Mapping
(Mappings for remaining YANG substatements)

Attribute → "leaf" (for single-valued attribute) or "leaf list" (multi-valued) statement		
UML Artifact	YANG Artifact	Comments
Operation exception error notification?	"must" substatement	
{<constraint>}	"when" substatement	

Table 5.6: Attribute Type Mapping Example


«OpenModelClass»
Class1
«OpenModelAttribute» + class1Id: String [1] {readOnly, unique}
«OpenModelAttribute» + attribute1: String [1]
«OpenModelAttribute» {valueRange=1-100 } + attribute2: Integer [2..6]
«OpenModelAttribute» {isInvariant=true } + attribute3: Boolean [1] = true {readOnly, unique}
«OpenModelAttribute» + attribute4: Enumeration1 [1] = LITERAL_2 {readOnly, unique}
«Enumeration»
Enumeration1
LITERAL_1
LITERAL_2
LITERAL_3
This object class models the ...

```
grouping Class1 {
    description "This class models the...";

    leaf class1Id {
        type string;
        mandatory true;
        config false;
    }

    leaf attribute1 {
        type string;
        mandatory true;
    }

    leaf-list attribute2 {
        type int8 {
            range "1-100";
        }
        min-elements 2;
        max-elements 6;
    }

    leaf attribute3 {
        type boolean;
        default true;
        config false;
        ompExt:isInvariant
    }

    leaf attribute4 {
        type enumeration {
            enum LITERAL_1;
            enum LITERAL_2;
            enum LITERAL_3;
        }
        default LITERAL_2;
        config false;
    }

}
list Class1 {
    key class1Id;
    uses Class1;
}
```

5.3 Mapping of Data Types

Various kinds of data types are defined in UML:

- Primitive Data Types (not further structured; e.g., Integer, MAC address)
- Complex Data Types (containing attributes; e.g., Host which combines ipAddress and domainName)
- Enumerations

They are used as the type definition of attributes and parameters.

5.3.1 Generic Mapping of Primitive Data Types

Table 5.7: Primitive Data Type Mapping

Primitive Data Type → "typeDef" statement		
UML Artifact	YANG Artifact	Comments
documentation "Applied comments" (carried in XMI as "ownedComment")	"description" substatement	Multiple "applied comments" defined in UML, need to be collapsed into a single "description" substatement.
type	"type" substatement (built-in type or derived type)	
defaultValue	"default" substatement	If a default value exists and it is the desired value, the parameter does not have to be explicitly configured by the user. When the value of "defaultValue" is "NA", the tool ignores it and doesn't print out "default" substatement.
unit	"units" substatement	
«Reference»	"reference" substatement	
«Example»	Ignore Example elements and all composed parts	
lifecycleState	"status" substatement or "description" substatement	See section 5.10.

5.3.2 Generic Mapping of Complex Data Types

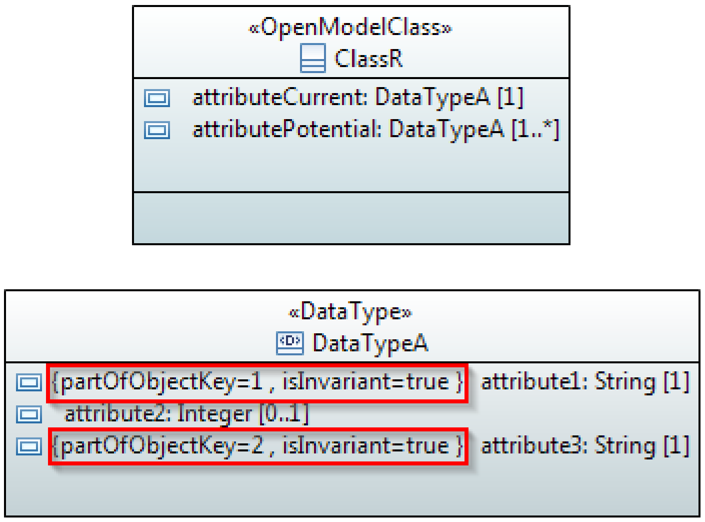
Table 5.8: Complex Data Type Mapping

Complex Data Type containing only one attribute → "typedef" statement; see Table 5.7 Complex Data Type containing more than one attribute → "grouping" statement		
UML Artifact	YANG Artifact	Comments
documentation "Applied comments" (carried in XMI as "ownedComment")	"description" substatement	Multiple "applied comments" defined in UML, need to be collapsed into a single "description" substatement.
not used	"action" substatement	
XOR: See section 6.3	"choice" substatement	
«Reference»	"reference" substatement	

Complex Data Type containing only one attribute → “typedef” statement; see Table 5.7 Complex Data Type containing more than one attribute → “grouping” statement		
UML Artifact	YANG Artifact	Comments
«Example»	Ignore Example elements and all composed parts	
lifecycleState	“status” substatement or “description” substatement	See section 5.9.
complex attribute	“uses”, “container” or “list” substatement	

Note: Leaf and leaf-list can only use built-in types, typeDef types or enumerations in their type substatement; i.e., not groupings. Complex data types with more than one item (e.g., name value pair) can only be defined using groupings. Groupings can only be used by grouping, container and list statements.

Table 5.9: Complex Data Type Mapping Example


«OpenModelClass»
ClassR
attributeCurrent: DataTypeA [1]
attributePotential: DataTypeA [1..*]
«DataType»
DataTypeA
{partOfObjectKey=1 , isInvariant=true } attribute1: String [1]
attribute2: Integer [0..1]
{partOfObjectKey=2 , isInvariant=true } attribute3: String [1]

```
container ClassR {
      uses ClassR;
      …
}
-------------------------------------
grouping ClassR
      …
      container attributeCurrent {
            …
            uses DataTypeA;
      }
      list attributePotential {
            key "attribute1 attribute3";
            …
            min-elements 1;
            uses DataTypeA;
      }
}
-------------------------------------
grouping DataTypeA {
      …
      leaf attribute1 {
            …
            type string;
            mandatory true;
      }
      leaf attribute2 {
            …
            type int64;
      }
      leaf-list attribute3 {
            …
            type string;
            mandatory true;
      }
}
```

5.3.3 Mapping of Common Primitive and Complex Data Types

A list of generic UML data types is defined in a “CommonDataTypes” Model Library. This library is imported to every UML model to make these data types available for the model designer.

Table 5.10: Common Primitive and Complex Data Type Mapping

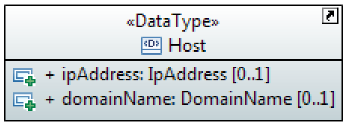
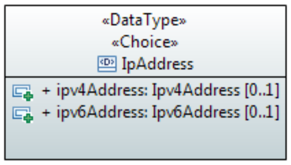
UML CommonDataTypes → YANG Built-In Types, ietf-yang-types, ietf-inet-types		
UML Artifact	YANG Artifact	Comments
UML Primitive Types		The following YANG Built-In types are currently not used: • binary | Any binary data • bits | A set of bits or flags • decimal64 | 64-bit signed decimal number
Boolean	Built-In Type::boolean	
«LENGTH_8_BIT» Integer	Built-In Type::int8	
«LENGTH_16_BIT» Integer	Built-In Type::int16	
«LENGTH_32_BIT» Integer	Built-In Type::int32	
«LENGTH_64_BIT» Integer	Built-In Type::int64	
Integer		If bitLength = NA
«UNSIGNED, LENGTH_8_BIT» Integer	Built-In Type::uint8	
«UNSIGNED, LENGTH_16_BIT» Integer	Built-In Type::uint16	
«UNSIGNED, LENGTH_32_BIT» Integer	Built-In Type::uint32	
«UNSIGNED, LENGTH_64_BIT» Integer	Built-In Type::uint64	

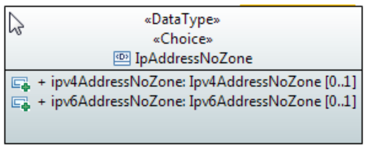
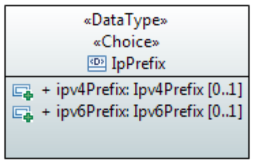
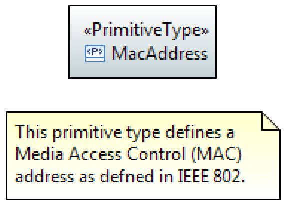
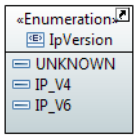
UML CommonDataTypes → YANG Built-In Types, ietf-yang-types, ietf-inet-types		
UML Artifact	YANG Artifact	Comments
Real (Not used so far. See also float and double below.)	Built-In Type::decimal64	YANG foresees a corresponding built-in type "decimal64" (RFC6020 sect. 9.3) but, for this built-in type, YANG requires mandatory to express also the accuracy with the "fraction-digit" sub-statement (RFC6020 sect. 9.3.4), which indicates the expected number of significant decimal digits. "fraction-digit" could range from 1 to 18. Based on the value assigned to the "fraction-digit", the range of real numbers that can be expressed changes significantly. RFC6020 in sect. 9.3.4 shows the supported ranges based on the value chosen for "fraction-digit". Here things work in such a way that, the larger the range you want to express, the lower the accuracy in terms of decimal part. It's not even so immediate to identify a conventional, "nominal" level of accuracy, since this actually depends on the specific context of application. To achieve this, we should identify a level of accuracy that we are sure suits always to all possible cases. So, even if we have a 1:1 correspondence of built-in type between UML and YANG, an automatic conversion to provide the correct mapping couldn't be so straightforward as it appears at a first glance.

«LENGTH_32_BIT» Real (float)	typedef float {
	  type decimal64 {
	   fraction-digits 7;
	  }
	}	

«LENGTH_64_BIT» Real (double)	typedef double {
	  type decimal64 {
	   fraction-digits 16;
	  }
	}	

String	Built-In Type::string	
~~Unlimited Natural~~		currently not used

UML CommonDataTypes → YANG Built-In Types, ietf-yang-types, ietf-inet-types		
UML Artifact	YANG Artifact	Comments
Counter and Gauge Types		
«COUNTER, LENGTH_32_BIT» Integer	ietf-yang-types::counter32	
«COUNTER, LENGTH_64_BIT» Integer	ietf-yang-types::counter64	
«GAUGE, LENGTH_32_BIT» Integer	ietf-yang-types::gauge32	
«GAUGE, LENGTH_64_BIT» Integer	ietf-yang-types::gauge64	
«ZERO_COUNTER, LENGTH_32_BIT» Integer	ietf-yang-types::zero-based-counter32	
«ZERO_COUNTER, LENGTH_64_BIT» Integer	ietf-yang-types::zero-based-counter64	
Date and Time related Types		
DateTime	ietf-yang-types::date-and-time	
~~Timestamp~~	~~ietf-yang-types::timestamp~~	Not needed
~~Timeticks~~	~~ietf-yang-types::timeticks~~	hundredths of seconds since an epoch, best mapped to dateTime Not needed
Domain Name and URI related Types		
DomainName	ietf-inet-types::domain-name	
 «DataType» Host + ipAddress: IpAddress [0..1] + domainName: DomainName [0..1]	ietf-inet-types::host	
Uri	ietf-inet-types::uri	
Address related Types		
 «DataType» «Choice» IpAddress + ipv4Address: Ipv4Address [0..1] + ipv6Address: Ipv6Address [0..1]	ietf-inet-types::ip-address	
Ipv4Address	ietf-inet-types::ipv4-address	
Ipv6Address	ietf-inet-types::ipv6-address	

UML CommonDataTypes → YANG Built-In Types, ietf-yang-types, ietf-inet-types		
UML Artifact	YANG Artifact	Comments
 «DataType» «Choice» IpAddressNoZone + ipv4AddressNoZone: Ipv4AddressNoZone [0..1] + ipv6AddressNoZone: Ipv6AddressNoZone [0..1]	ietf-inet-types::ip-address-no-zone	
Ipv4AddressNoZone	ietf-inet-types::ipv4-address-no-zone	
Ipv6AddressNoZone	ietf-inet-types::ipv6-address-no-zone	
Ipv4Prefix	ietf-inet-types::ipv4-prefix	
Ipv6Prefix	ietf-inet-types::ipv6-prefix	
 «DataType» «Choice» IpPrefix + ipv4Prefix: Ipv4Prefix [0..1] + ipv6Prefix: Ipv6Prefix [0..1]	ietf-inet-types::ip-prefix	
MacAddress  «PrimitiveType» MacAddress This primitive type defines a Media Access Control (MAC) address as defned in IEEE 802.	ietf-yang-types::mac-address	
~~PhysAddress~~	~~ietf-yang-types::phys-address~~	Not needed
Protocol Field related Types		
Dscp	ietf-inet-types::dscp	
	ietf-inet-types::ip-version	
IpV6FlowLabel	ietf-inet-types::ipv6-flow-label	
PortNumber	ietf-inet-types::port-number	
String related Types		
DottedQuad	ietf-yang-types::dotted-quad	
«OctetEncoded» String	??	
HexString «HexEncoded» String	ietf-yang-types::hex-string	
«Base64Encoded» String	??	

UML CommonDataTypes → YANG Built-In Types, ietf-yang-types, ietf-inet-types		
UML Artifact	YANG Artifact	Comments
Uuid	ietf-yang-types::uuid	To map to a language specific implementation

??	typedef duration {
	type string {
	pattern "P[0-9]+Y[0-9]+M[0-9]+DT[0-9]+H[0-9]+M [0-9]+(\.[0-9]+)?S";
	}
	}
	}
		e.g. P0Y1347M0D

5.3.4 Mapping of Enumeration Types

In UML, the definition of enumerated data-types allows to constrain the set of accepted values for an attribute. There are two ways to map this in YANG: either using the “enumeration” built-in type or via the “identity” statement.

YANG allows to use the “enumeration” built-in type either directly in the “leaf” or “leaf-list” definition or indirect via a separate “typedef”. Since UML supports only the indirect way via the definition of a separate Enumeration data type, the direct definition of an enumeration within a “leaf” or “leaf-list” is not recommended.

The YANG “enumeration” is a centralized definition totally included inside a single YANG module and eventually imported by the other modules. All the importing modules have access to the full set of defined values. Every variation of an enumeration shall be done in the defining module and it is globally available to all modules using it. It is not possible to have local extensions of the value set, where “local” means limited to a single YANG module, as it would be useful in case e.g. of experimental or proprietary extensions which should not affect or should be kept hidden to the rest of the modules.

The YANG “identity” is a distributed definition that can spread across several YANG modules. A YANG module could contain the definition of the base identity, representing the reference container for the allowed values, together with a first common set of values intended for global usage. Each importing module can then also locally add further values related to that identity. Every importing module can access the global sub-set of values and the additional values defined locally, but it has no access to the other local values defined inside other modules that it not imports. This means that extra identity values defined within one YANG module X are not visible to other YANG modules unless they import the module X. This allows for flexible and decoupled extensions and for accommodating additional experimental or proprietary values without impacts on the other modules, which are not even aware of the additional values.

YANG enumeration is in general more straightforward and shall be preferred when the UML enumeration is or can be considered highly consolidated and unlikely to be extended.

YANG identity shall be used for all the cases where the UML enumeration is not fully consolidated or cannot be consolidated, e.g. because the associated set of value is known to be open (or has to be left open) for future yet not known or not consolidated extensions.

To direct the mapping tool to perform the appropriate choice,

Table 5.11: Enumeration Type Mapping
(Mappings required by currently used UML artifacts)

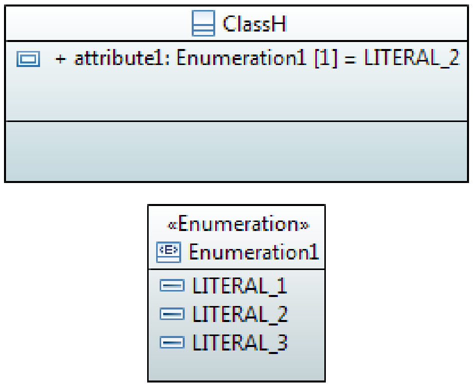
Fixed Enumeration Type → "typedef" with "enum" statement Enhanceable Enumeration Type → "identity"/"base" statements		
UML Artifact	YANG Artifact	Comments
documentation "Applied comments" (carried in XMI as "ownedComment")	"description" substatement	Multiple "applied comments" defined in UML, need to be collapsed into a single "description" substatement.
literal name	enum name	
literal integer	"value" substatement	

isLeaf = true
isLeaf = false
	"enum" substatement
	"identity"/"base" pattern
		UML definition ""

«Reference»	"reference" substatement	
«Example»	Ignore Example elements and all composed parts	
lifecycleState	"status" substatement or "description" substatement	See section 5.10.

The table below shows the two approaches applied to the YANG mapping for a UML enumerated type.

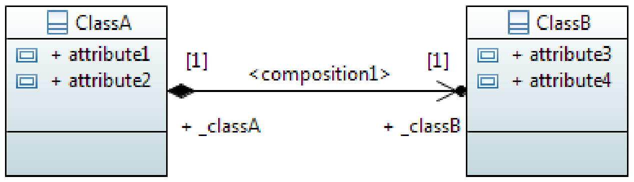
Table 5.12: Enumeration Type Mapping Example

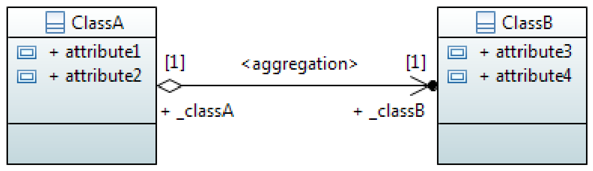
	**Using “enumeration”**
	direct usage (not recommended):
	container ClassH {
	…
	leaf attribute1 {
	type enumeration {
	enum LITERAL_1;
	enum LITERAL_2;
	enum LITERAL_3;
	default LITERAL_2;
	mandatory true;
	}
	}
	}
	typedef Enumeration1 {
	type enumeration {
	enum LITERAL_1;
	enum LITERAL_2;
	enum LITERAL_3;
	}
	}
	indirect usage:
	container ClassH {
	…
	leaf attribute1 {
	type Enumeration1;
	default LITERAL_2;
	}
	}

	Using “identity”/”base”
	// an empty identity value is a “base identity”
	// i.e. it provides the reference name for a set of values
	identity Enumeration1;
	identity LITERAL_1 {
	// the “base” statement qualifies this identity value
	// as belonging to the AdministrativeState set
	base Enumeration1;
	}
	identity LITERAL_2 {
	base Enumeration1;
	}
	identity LITERAL_3 {
	base Enumeration1;
	}
	……
	typedef Enumeration1 {
	type identityref {
	// “identityref” defines the associated set
	base Enumeration1;
	}
	}
	......
	leaf attribute1 {
	type Enumeration1;
	}

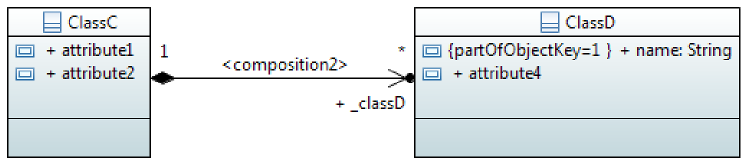
5.4 Mapping of Associations

Table 5.13: Association Mapping Examples

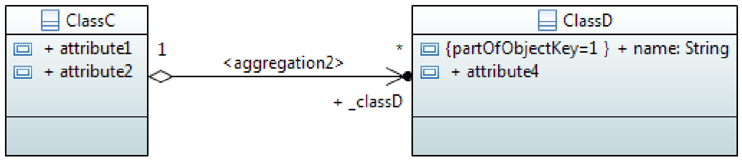
 ClassA
+ attribute1
+ attribute2
[1]
<composition1>
[1]
+ _classA
+ _classB
ClassB
+ attribute3
+ attribute4
	grouping ClassA {
	      …
	      leaf attribute1 {
	            …
	      }
	      leaf attribute2 {
	            …
	      }
	}
	grouping ClassB {
	      …
	      leaf attribute3 {
	            …
	      }
	      leaf attribute4 {
	            …
	      }
	}
	container ClassA {
	      …
	      uses ClassA;
	      uses ClassB;
	}



```
grouping ClassA {
    …
    leaf attribute1 {
        …
    }
    leaf attribute2 {
        …
    }
}
grouping ClassB {
    …
    leaf attribute3 {
        …
    }
    leaf attribute4 {
        …
    }
}
container ClassB {
    …
    uses ClassB;
}
container ClassA {
    …
    uses ClassA;
    leaf _classB {
        type leafref {
            path '/ClassB';
        }
    }
}
```



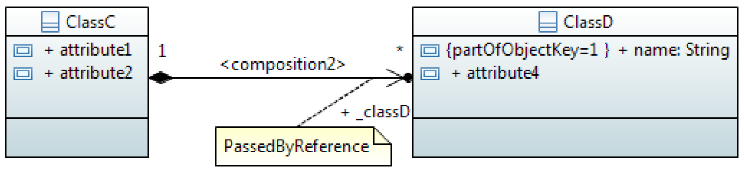
```
grouping ClassC {
    …
    leaf attribute1 {
        …
    }
    leaf attribute2 {
        …
    }
    list classD {
        key "name";
        uses ClassD;
    }
}
grouping ClassD {
    leaf name {
        type string;
        …
    }
    leaf attribute4 {
        …
    }
}
container ClassC {
    …
    uses ClassC;
}
```



```
grouping ClassC {
      …
      leaf attribute1 {
            …
      }
      leaf attribute2 {
            …
      }
}
grouping ClassD {
      leaf name {
            type string;
      }
      leaf attribute4 {
            …
      }
}
list ClassD {
      key "name";
      uses ClassD;
}
```

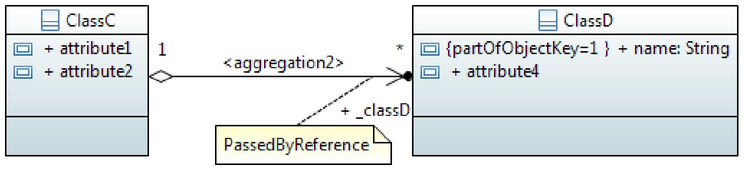
How to relate ClassD to ClassC?

```
container ClassC {
      …
      uses ClassC;
      leaf classD {
            type leafref {
                  path '/ClassD';
            }
      }
}
```



```
grouping ClassC {
        …
        leaf attribute1 {
                …
        }
        leaf attribute2 {
                …
        }
}
grouping ClassD {
        leaf name {
                type string;
        }
        leaf attribute4 {
                …
        }
}
list ClassD {
        key "name";
        uses ClassD;
}
container ClassC {
        …
        uses ClassC;
        leaf classD {
                type leafref {
                        path "/ClassD/name";
                }
        }
}
```

Note: Lifecycle dependency is not considered in the YANG mapping!



```
grouping ClassC {
    …
    leaf attribute1 {
        …
    }
    leaf attribute2 {
        …
    }
    leaf-list _classD {
    type leafref {
        path "/ClassD/name";
    }
}
grouping ClassD {
    leaf name {
        type string;
    }
    leaf attribute4 {
        …
    }
}
list ClassD {
    key "name";
    uses ClassD;
}
container ClassC {
    …
    uses ClassC;
    leaf classD {
        type leafref {
            path "/ClassD/name";
        }
    }
}
```

The following table summarizes the association mappings.

Table 5.14: Association Mapping Summary

		UML		
		containment	association	inheritance
YANG	nesting	√		
	grouping			√ abstract superclasses
	augment			√ concrete superclasses
	leafref		√	

5.5 Mapping of Interfaces (grouping of operations)

Table 5.15: UML Interface Mapping

UML Interface → Container?		
UML Artifact	YANG Artifact	Comments
documentation “Applied comments” (carried in XMI as “ownedComment”)	“description” substatement	Multiple “applied comments” defined in UML, need to be collapsed into a single “description” substatement.
abstract	"grouping" statement	
«Reference»	“reference” substatement	
«Example»	Ignore Example elements and all composed parts	
lifecycleState	“status” substatement or “description” substatement	See section 5.10.
support	“if-feature” substatement	Support and condition belong together. If the “support” is conditional, then the “condition” explains the conditions under which the class has to be supported.
condition		

5.6 Mapping of Operations

Table 5.16: Operation Mapping

Operation → “action” and “rpc” statements (RFC 6020: The difference between an action and an rpc is that an action is tied to a node in the data tree, whereas an rpc is associated at the module level.)		
UML Artifact	YANG Artifact	Comments
documentation “Applied comments” (carried in XMI as “ownedComment”)	“description” substatement	Multiple “applied comments” defined in UML, need to be collapsed into a single “description” substatement.

Operation → “action” and “rpc” statements

(RFC 6020: The difference between an action and an rpc is that an action is tied to a node in the data tree, whereas an rpc is associated at the module level.)

UML Artifact	YANG Artifact	Comments
pre-condition	“extension” substatement→ ompExt: preCondition	RFC 6020: During the NETCONF <edit-config> processing errors are already send for: - Delete requests for non-existent data. - Create requests for existent data. - Insert requests with "before" or "after" parameters that do not exist. - Modification requests for nodes tagged with "when", and the "when" condition evaluates to "false". See extensions YANG module in section 8.2.
post-condition	“extension” substatement→ ompExt: postCondition	See extensions YANG module in section 8.2.
input parameter	“input” substatement	
output parameter	“output” substatement	
operation exceptions Internal Error Unable to Comply Comm Loss Invalid Input Not Implemented Duplicate Entity Not Found Object In Use Capacity Exceeded Not In Valid State Access Denied	“extension” substatement→ ompExt:operationExceptions	See extensions YANG module in section 8.2.
isOperationIdempotent	“extension” substatement→ ompExt:isOperationIdempotent	See extensions YANG module in section 8.2.

error-tag	error-app-tag
operation-failed	too-many-elements too-few-elements must-violation
data-missing	instance-required missing-choice
bad-attribute	missing-instance

Operation → "action" and "rpc" statements (RFC 6020: The difference between an action and an rpc is that an action is tied to a node in the data tree, whereas an rpc is associated at the module level.)		
UML Artifact	YANG Artifact	Comments
isAtomic	"extension" substatement→ ompExt:isAtomic	See extensions YANG module in section 8.2
«Reference»	"reference" substatement	
«Example»	Ignore Example elements and all composed parts	
lifecycleState	"status" substatement or "description" substatement	See section 5.10.
support	"if-feature" substatement	Support and condition belong together. If the "support" is conditional, then the "condition" explains the conditions under which the class has to be supported.
condition		

Table 5.17: Interface/Operation Mapping Example

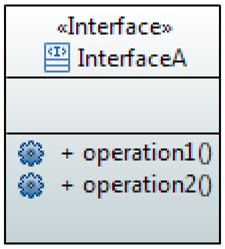
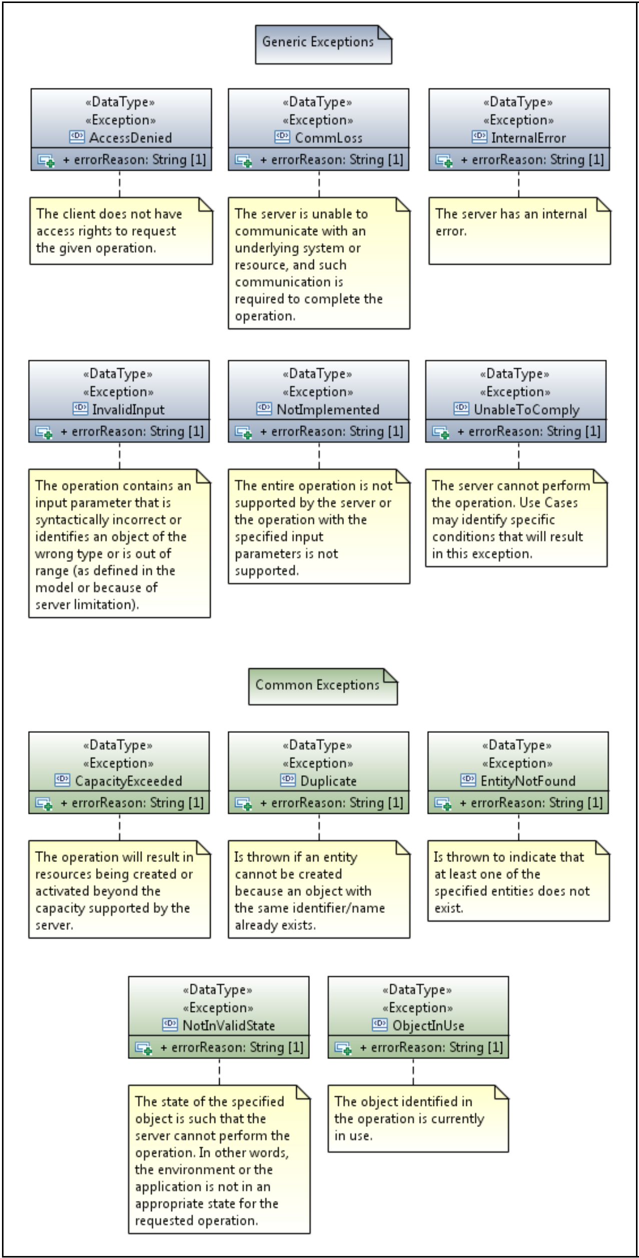
 «Interface»
InterfaceA
+ operation1()
+ operation2()
	container InterfaceA {
	    …
	    action operation1 {
	        …
	    }
	    action operation2 {
	        …
	    }
	}

Table 5.18: Operation Exception Mapping Example

 Generic Exceptions «DataType» «Exception» AccessDenied + errorReason: String [1] The client does not have access rights to request the given operation. «DataType» «Exception» CommLoss + errorReason: String [1] The server is unable to communicate with an underlying system or resource, and such communication is required to complete the operation. «DataType» «Exception» InternalError + errorReason: String [1] The server has an internal error. «DataType» «Exception» InvalidInput + errorReason: String [1] The operation contains an input parameter that is syntactically incorrect or identifies an object of the wrong type or is out of range (as defined in the model or because of server limitation). «DataType» «Exception» NotImplemented + errorReason: String [1] The entire operation is not supported by the server or the operation with the specified input parameters is not supported. «DataType» «Exception» UnableToComply + errorReason: String [1] The server cannot perform the operation. Use Cases may identify specific conditions that will result in this exception. Common Exceptions «DataType» «Exception» CapacityExceeded + errorReason: String [1] The operation will result in resources being created or activated beyond the capacity supported by the server. «DataType» «Exception» Duplicate + errorReason: String [1] Is thrown if an entity cannot be created because an object with the same identifier/name already exists. «DataType» «Exception» EntityNotFound + errorReason: String [1] Is thrown to indicate that at least one of the specified entities does not exist. «DataType» «Exception» NotInValidState + errorReason: String [1] The state of the specified object is such that the server cannot perform the operation. In other words, the environment or the application is not in an appropriate state for the requested operation. «DataType» «Exception» ObjectInUse + errorReason: String [1] The object identified in the operation is currently in use.	??

5.7 Mapping of Operation Parameters

Table 5.19: Parameter Mapping

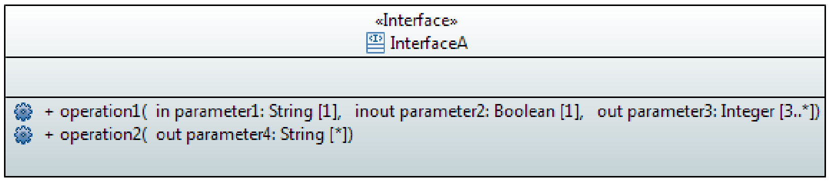
Operation Parameters → "input" substatement or "output" substatement		
UML Artifact	YANG Artifact	Comments

Operation Parameters → "input" substatement or "output" substatement		
UML Artifact	YANG Artifact	Comments
documentation "Applied comments" (carried in XMI as "ownedComment")	"description" substatement	Multiple "applied comments" defined in UML, need to be collapsed into a single "description" substatement.
direction	"input" / "output" substatement	
type	see mapping of attribute types (grouping, leaf, leaf-list, container, list, typedef, uses)	
isOrdered		
multiplicity		
defaultValue		
valueRange		

passedByReference	if passedByReference = true →
	type leafref {
	path "/<object>/<object identifier>"
	if passedByReference = false →
	either "list" statement (key property, multiple instances) or "container" statement (single instance)
		Relevant only to parameters that have a class defined as their type.

«Reference»	"reference" substatement of the individual parameters (container, leaf, leaf-list, list, uses)	
«Example»	Ignore Example elements and all composed parts	
lifecycleState	"status" substatement or "description" substatement of the individual parameters (container, leaf, leaf-list, list, uses)	See section 5.10.
support	"if-feature" substatement of the individual parameters (container, leaf, leaf-list, list, uses)	Support and condition belong together. If the "support" is conditional, then the "condition" explains the conditions under which the class has to be supported.
condition		
XOR: See section 6.3	"choice" substatement	
error notification?	"must" substatement	
complex parameter	"uses" substatement	

Table 5.20: Interface/Operation/Parameter Mapping Example

 «Interface»
InterfaceA
+ operation1(in parameter1: String [1], inout parameter2: Boolean [1], out parameter3: Integer [3..*])
+ operation2(out parameter4: String [*])
	container InterfaceA {
	    …
	    action operation1 {
	        …
	        input {
	            leaf parameter1 {
	                type string;
	                mandatory true;
	            }
	            leaf parameter2 {
	                type boolean;
	                mandatory true;
	            }
	        }
	        output {
	            leaf parameter2 {
	                type boolean;
	                mandatory true;
	            }
	            leaf-list parameter3 {
	                type int16;
	                min-elements 3;
	            }
	        }
	    }
	    action operation2 {
	        …
	        output {
	            leaf-list parameter4 {
	                type string;
	            }
	        }
	    }
	}

5.8 Mapping of Notifications

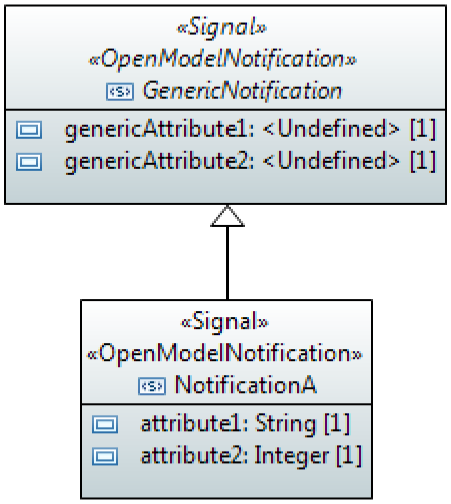
Like the class mapping, the signals are also mapped in two steps. In the first step, all signals are mapped to “grouping” statements. In the second step the groupings of all non-abstract signals are “instantiated” in “notification” statements.

Table 5.21: Notification Mapping

Signal → “grouping” statement → “notification“ statement		
UML Artifact	YANG Artifact	Comments
documentation “Applied comments” (carried in XMI as “ownedComment”)	“description” substatement	Multiple “applied comments” defined in UML, need to be collapsed into a single “description” substatement.
«Reference»	“reference” substatement	

Signal → “grouping” statement → “notification“ statement		
UML Artifact	YANG Artifact	Comments
«Example»	Ignore Example elements and all composed parts	
lifecycleState	“status” substatement or “description” substatement	See section 5.10.
support	“if-feature” substatement	Support and condition belong together. If the “support” is conditional, then the “condition” explains the conditions under which the class has to be supported.
condition		
Proxy Class: See section 6.6. XOR: See section 6.3.	“choice” substatement	
error notification?	“must” substatement	
attributes	see mapping of attribute types (grouping, leaf, leaf-list, container, list, typedef, uses)	
complex attribute	“uses” substatement	

Table 5.22: Notification Mapping Example



```
grouping GenericNotification {
      …
      leaf genericAttribute1 {
            …
            mandatory true;
      }
      leaf-list genericAttribute2 {
            …
            mandatory true;
      }
}
grouping NotificationA {
      …
      leaf attribute1 {
            type string
            …
      }
      leaf attribute2 {
            type integer
            …
      }
}
notification NotificationA {
      …
      uses GenericNotification;
      uses NotificationA;
}
```

Table from onf2015.276:

Parameter name	**ITU-T M.3702**	**3GPP TS32.302**
objectClass	M	M
objectInstance	M	M
notificationId aka notificationIdentifier	M	M
eventTime	M	M
systemDN	M	M
notificationType	M	M

5.9 Mapping of UML Packages

The mapping tool shall generate a YANG module per UML model.

According to the UML Modeling Guidelines [7], each UML model is basically structured into the following packages:

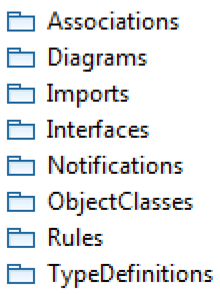

Associations
Diagrams
Imports
Interfaces
Notifications
ObjectClasses
Rules
TypeDefinitions

Figure 5.1: Pre-defined Packages in a UML Module

The grouping that is provided through these packages shall persist in the YANG module using headings defined as comments.

Table 5.23: Interface/Operation/Parameter Mapping Example

TypeDefinitions	/** * package TypeDefinitions **/ …
ObjectClasses	/** * package ObjectClasses **/ …
Interfaces	/** * package Interfaces **/ …
Notifications	/** * package Notifications **/ …

5.10 Mapping of Lifecycle

Table 5.24: Lifecycle Mapping

UML Lifecycle		
UML Artifact	YANG Artifact	Comments
<Lifecycle Stereotypes>	“status“ substatement or “description” substatement	«UML» → “YANG” «Deprecated» → "deprecated" «Experimental» (default) → description «Faulty» → description «LikelyToChange» → description «Mature» → "current" (default) «Obsolete» → "obsolete" «Preliminary» → description Allow having a switch per state in the mapping tool to map it or not; default is Mature only. See also section 7.2.

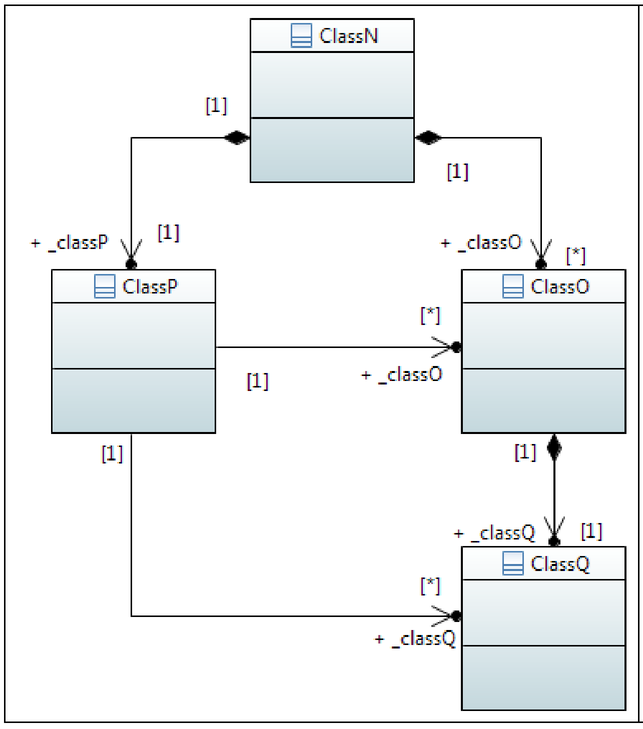
5.11 Mapping Issues

5.11.1 YANG 1.0 or YANG 1.1?

YANG 1.0 is approved and defined in RFC 6020.
YANG 1.1 is not approved and its definition is ongoing in draft-ietf-netmod-rfc6020bis (currently version 14). The enhancements are listed in section “Summary of Changes from RFC 6020”.

5.11.2 Combination of different Associations?

Table 5.25: Combination of Associations Mapping Examples



```
container ClassN {
      …
      container ClassP {
            …
            leaf-list classO {
                  type leafref {
                              path "/ClassO/name";
                  }
            }
            leaf classQ {
                  type leafref {
                              path "/ClassQ/name";
                  }
            }
      }
      list ClassO {
            …
            container ClassQ {
                  …
            {
      }
}
```

6 Mapping Patterns

6.1 UML Recursion

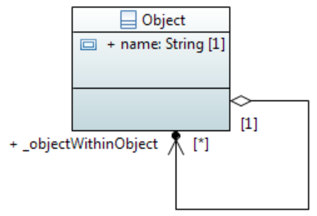
As YANG defines hierarchical data store, any instances that need to store recursive containment will require translation. A mapping between object-oriented store and a hierarchical store is possible; however, there is more than one option: e.g.,

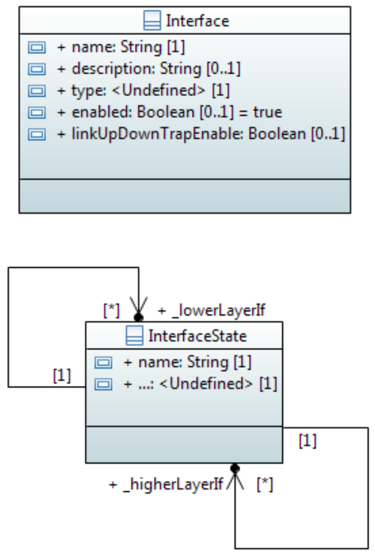
- Reference based approach - have a flat list of objects, where the objects are linked into a hierarchy using references. An example of a two-way navigable approach is in RFC 7223.
- Assume some specific number of "recursions"; i.e., specify some default number of recursion levels, and define a configurable parameter to allow changing the number of levels.

Text to be inserted discussing the pros and cons of these options, and rational for selecting the referenced based approach.

6.1.1 Reference Based Approach

Table 6.1: Recursion Mapping Examples

 Object
+ name: String [1]
[1]
+ _objectWithinObject
[*]
	list object {
	    key name;
	    leaf name {
	        type string;
	    }
	    leaf-list object-within-object {
	        type leafref {
	            path “/object/name";
	        }
	    }
	}

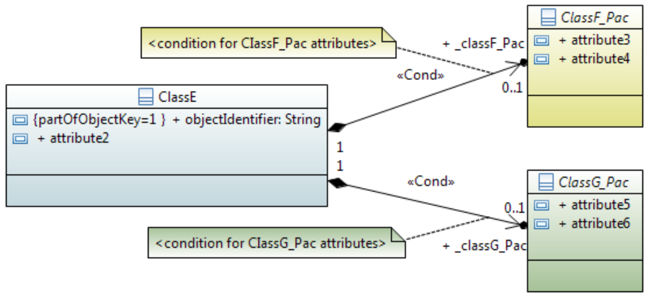
 Interface
+ name: String [1]
+ description: String [0..1]
+ type: <Undefined> [1]
+ enabled: Boolean [0..1] = true
+ linkUpDownTrapEnable: Boolean [0..1]
[*]
+ _lowerLayerIf
InterfaceState
[1]
+ name: String [1]
+ ...: <Undefined> [1]
[1]
+ _higherLayerIf
[*]
	Example from IETF RFC 7223 (https://datatracker.ietf.org/doc/rfc7223/)
	`+--rw interfaces`
	`|  +--rw interface* [name]`
	`|     +--rw name                         string`
	`|     +--rw description?                 string`
	`|     +--rw type                         identityref`
	`|     +--rw enabled?                     Boolean`
	`|     +--rw link-up-down-trap-enable?   enumeration`
	`+--ro interfaces-state`
	`   +--ro interface* [name]`
	`      +--ro name               string`
	`      +-- ...`
	`      +--ro higher-layer-if*   interface-state-ref`
	`      +--ro lower-layer-if*    interface-state-ref`
	`      +-- ...`
	`where`
	`  typedef interface-state-ref {`
	`    type leafref {`
	`      path "/if:interfaces-state/if:interface/if:name";`
	`    }`
	`    description`
	`      "This type is used by data models that need to`
	`       reference the operationally present interfaces.";`
	`  }`
	`  leaf-list higher-layer-if {`
	`    type interface-state-ref;`
	`    description`
	`      "A list of references to interfaces layered on top`
	`       of this interface.";`
	`    reference`
	`      "RFC 2863: The Interfaces Group MIB -`
	`       fStackTable";`
	`  }`
	`  leaf-list lower-layer-if {`
	`    type interface-state-ref;`
	`    description`
	`      "A list of references to interfaces layered`
	`       underneath this interface.";`
	`    reference`
	`      "RFC 2863: The Interfaces Group MIB -`
	`       ifStackTable";`
	`  }`

6.2 UML Conditional Pacs

UML conditional Pacs are abstract classes used to group attributes which are associated to the containing class under certain conditions. The abstract “attribute containers” are mapped to container statements. The condition is mapped to the “presence” property of the container statement.

Note: An example of this usage is given in the “Data nodes for the operational state of IP on interfaces." within ietf-ip.yang (RFC 7277).

Table 6.2: Mapping of Conditional Packages

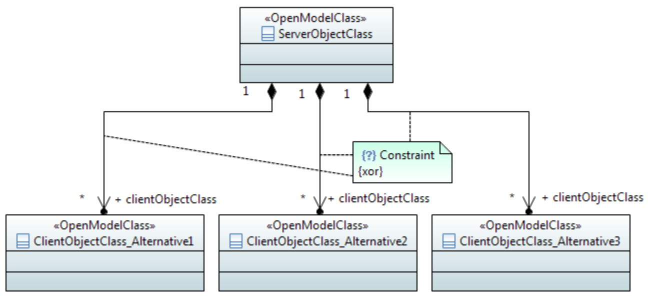


```
grouping ClassE {
    …
    leaf objectIdentifier {
        type string;
    }
    leaf attribute2 {
        …
    }
grouping ClassF_Pac {
    …
    leaf attribute3 {
        …
    }
    leaf attribute4 {
        …
    }
}
grouping ClassG_Pac {
    …
    leaf attribute5 {
        …
    }
    leaf attribute6 {
        …
    }
}
list ClassE {
    key "objectIdentifier";
    uses ClassE;
    …
    container ClassF_Pac {
        presence " <condition for ClassF_Pac attributes>";
        uses ClassF_Pac;
        …
    }
    container ClassG_Pac {
        presence " <condition for ClassG_Pac attributes>";
        uses ClassG_Pac;
        …
    }
}
```

6.3 XOR Relationship

The associations related by the “xor” constraint are mapped to the “choice” property of the container/list statement.

Table 6.3: XOR Relationship Mapping Example

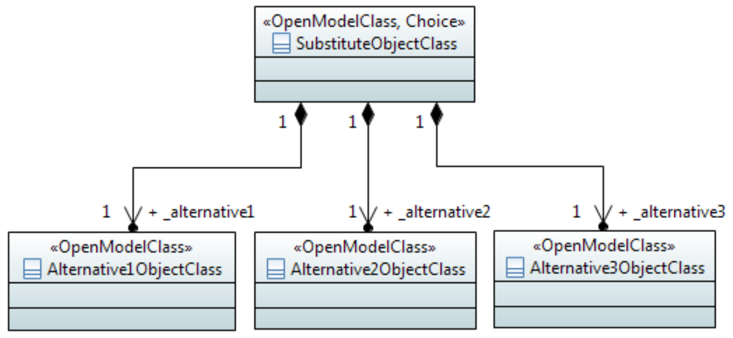


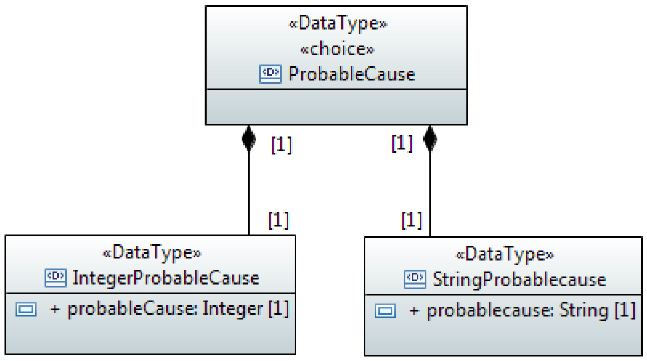
```
grouping ServerObjectClass {
     …
     }
grouping ClientObjectCLass_Alternative1 {
     …
}
grouping ClientObjectCLass_Alternative2 {
     …
}
grouping ClientObjectCLass_Alternative3 {
     …
}
list ClientObjectCLass_Alternative1 {
     key …;
     uses ClientObjectCLass_Alternative1;
     …
}
list ClientObjectCLass_Alternative2 {
     key …;
     uses ClientObjectCLass_Alternative2;
     …
}
list ClientObjectCLass_Alternative3 {
     key …;
     uses ClientObjectCLass_Alternative3;
     …
}
list ServerObjectClass {
     key …;
     …
     choice _clientObjectClass {
          case ClientObjectCLass_Alternative1 {
               leaf ClientObjectCLass_Alternative1 {
                    type leafref {
                         path '/ ClientObjectCLass_Alternative1';
                    }
               }
          }
          case ClientObjectCLass_Alternative2 {
               leaf ClientObjectCLass_Alternative2 {
                    type leafref {
                         path '/ ClientObjectCLass_Alternative2';
                    }
               }
          }
          case ClientObjectCLass_Alternative3 {
               leaf ClientObjectCLass_Alternative3 {
                    type leafref {
                         path '/ ClientObjectCLass_Alternative3';
                    }
               }
          }
     }
}
```

6.4 «Choice» Stereotype

The «Choice» stereotype can be associated in UML to a class or a data type. The class or a data type which is annotated with the «Choice» stereotype represents one of a set of classes/data types. This pattern is mapped to the “choice” property of the container/list/grouping statement.

Table 6.4: «Choice» Stereotype Mapping Examples

 «OpenModelClass, Choice»
SubstituteObjectClass
1
1
1
1
+ _alternative1
1
+ _alternative2
1
+ _alternative3
«OpenModelClass»
Alternative1ObjectClass
«OpenModelClass»
Alternative2ObjectClass
«OpenModelClass»
Alternative3ObjectClass
	```
	grouping SubstituteObjectClass {
	     …
	     }
	grouping Alternative1ObjectClass {
	     …
	}
	grouping Alternative2ObjectClass {
	     …
	}
	grouping Alternative3ObjectClass {
	     …
	}
	container Alternative1ObjectClass {
	     uses Alternative1ObjectClass;
	     …
	}
	container Alternative2ObjectClass {
	     uses Alternative2ObjectClass;
	     …
	}
	container Alternative3ObjectClass {
	     uses Alternative3ObjectClass;
	     …
	}
	list SubstituteObjectClass {
	     key …;
	     …
	     choice __alternative {
	          case Alternative1ObjectClass {
	               leaf Alternative1ObjectClass {
	                    type leafref {
	                         path '/Alternative1ObjectClass';
	                    }
	               }
	          }
	          case Alternative2ObjectClass {
	               leaf Alternative2ObjectClass {
	                    type leafref {
	                         path '/Alternative2ObjectClass';
	                    }
	               }
	          }
	          case Alternative3ObjectClass {
	               leaf Alternative3ObjectClass {
	                    type leafref {
	                         path '/Alternative3ObjectClass';
	                    }
	               }
	          }
	     }
	}
	```

 «DataType»
«choice»
ProbableCause
[1]
[1]
[1]
[1]
«DataType»
IntegerProbableCause
+ probableCause: Integer [1]
«DataType»
StringProbablecause
+ probablecause: String [1]
	grouping IntegerProbableCause {
	    …
	    leaf probableCause {
	        type int64;
	        …
	    }
	}
	grouping StringProbableCause {
	    …
	    leaf probableCause {
	        type string;
	        …
	    }
	}
	grouping ProbableCause {
	    …
	    choice probableCause {
	        case IntegerProbableCause {
	            container IntegerProbableCause {
	                uses IntegerProbableCause;
	                …
	            }
	        }
	        case StringProbableCause {
	            container StringProbableCause {
	                uses StringProbableCause;
	                …
	            }
	        }
	    }
	}

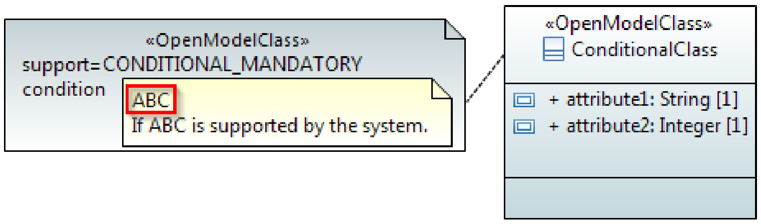
6.5 Mapping of UML Support and Condition Properties

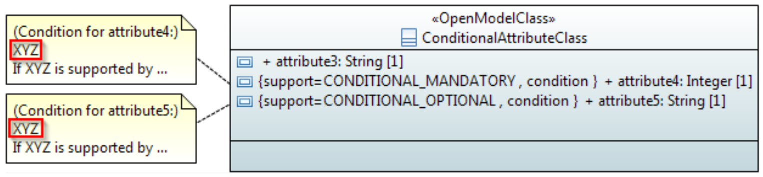
The UML Modeling Guidelines [7] define support and condition properties for the UML artifacts class, attribute, signal, interface, operation and parameter. The support property can be defined as one of M – Mandatory, O – Optional, C – Conditional, CM – Conditional-Mandatory, CO – Conditional-Optional. It qualifies the support of the artifact at the management interface. The condition property contains the condition for the condition-related support qualifiers (C, CM, CO).

M – Mandatory maps to the “mandatory” substatement. O – Optional need not be mapped since the default value of the “mandatory” substament is “false”; i.e., optional.

For the conditional UML support qualifiers, the first line of the condition text is mapped to a “feature” statement. The mapping tool needs to scan – in a first step – all conditions and create “feature” statements for each **different** first line of all conditions. The second and further lines of the condition text are mapped to the “description” substatement of the “feature” statement. In a second step, the tool adds an “if-feature” substatement with a reference to the corresponding “feature” to all mapped UML conditional artefacts.

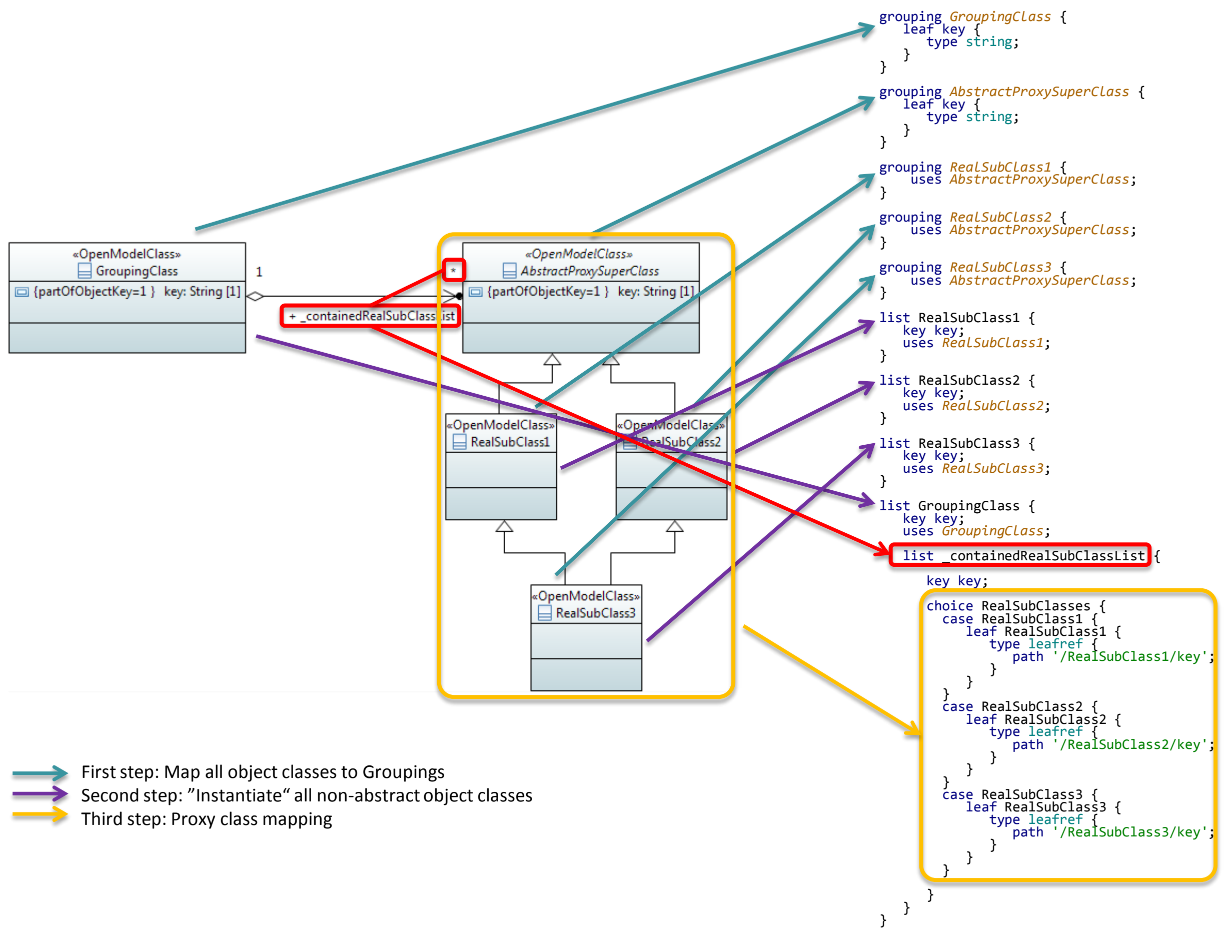
Table 6.5: Support and Condition Mapping Examples

 «OpenModelClass»
support=CONDITIONAL_MANDATORY
condition
ABC
If ABC is supported by the system.
«OpenModelClass»
ConditionalClass
+ attribute1: String [1]
+ attribute2: Integer [1]
	feature ABC {
	description
	"If ABC is supported by the system.";
	}
	container ConditionalClass {
	…
	if-feature “ABC”;
	leaf attribute1 {
	type string;
	…
	}
	leaf attribute2 {
	type int64;
	…
	}
	}

 (Condition for attribute4:)
XYZ
If XYZ is supported by ...
(Condition for attribute5:)
XYZ
If XYZ is supported by ...
«OpenModelClass»
ConditionalAttributeClass
+ attribute3: String [1]
{support=CONDITIONAL_MANDATORY , condition } + attribute4: Integer [1]
{support=CONDITIONAL_OPTIONAL , condition } + attribute5: String [1]
	feature XYZ {
	description
	"If XYZ is supported by …";
	}
	container ConditionalAttributeClass {
	…
	leaf attribute3 {
	type string;
	…
	}
	leaf attribute4 {
	type int64;
	if-feature “XYZ”;
	…
	}
	leaf attribute5 {
	type string;
	if-feature “not XYZ”;
	…
	}
	}

6.6 Proxy Class Association Mapping

UML allows an association to an abstract proxy class. This abstract proxy class is acting as a placeholder for all related (via inheritance or composition) classes. The mapping tool has to map this single association into relationships to all classes which are related to the proxy class.


«OpenModelClass»
GroupingClass
{partOfObjectKey=1 } key: String [1]
1
*
+ _containedRealSubClassList
«OpenModelClass»
AbstractProxySuperClass
{partOfObjectKey=1 } key: String [1]
«OpenModelClass»
RealSubClass1
«OpenModelClass»
RealSubClass2
«OpenModelClass»
RealSubClass3
First step: Map all object classes to Groupings
Second step: "Instantiate" all non-abstract object classes
Third step: Proxy class mapping

```
grouping GroupingClass {
   leaf key {
      type string;
   }
}
grouping AbstractProxySuperClass {
   leaf key {
      type string;
   }
}
grouping RealSubClass1 {
    uses AbstractProxySuperClass;
}
grouping RealSubClass2 {
    uses AbstractProxySuperClass;
}
grouping RealSubClass3 {
    uses AbstractProxySuperClass;
}
list RealSubClass1 {
   key key;
   uses RealSubClass1;
}
list RealSubClass2 {
   key key;
   uses RealSubClass2;
}
list RealSubClass3 {
   key key;
   uses RealSubClass3;
}
list GroupingClass {
   key key;
   uses GroupingClass;
   list _containedRealSubClassList {
      key key;
      choice RealSubClasses {
        case RealSubClass1 {
           leaf RealSubClass1 {
              type leafref {
                 path '/RealSubClass1/key';
              }
           }
        }
        case RealSubClass2 {
           leaf RealSubClass2 {
              type leafref {
                 path '/RealSubClass2/key';
              }
           }
        }
        case RealSubClass3 {
           leaf RealSubClass3 {
              type leafref {
                 path '/RealSubClass3/key';
              }
           }
        }
      }
   }
}
```

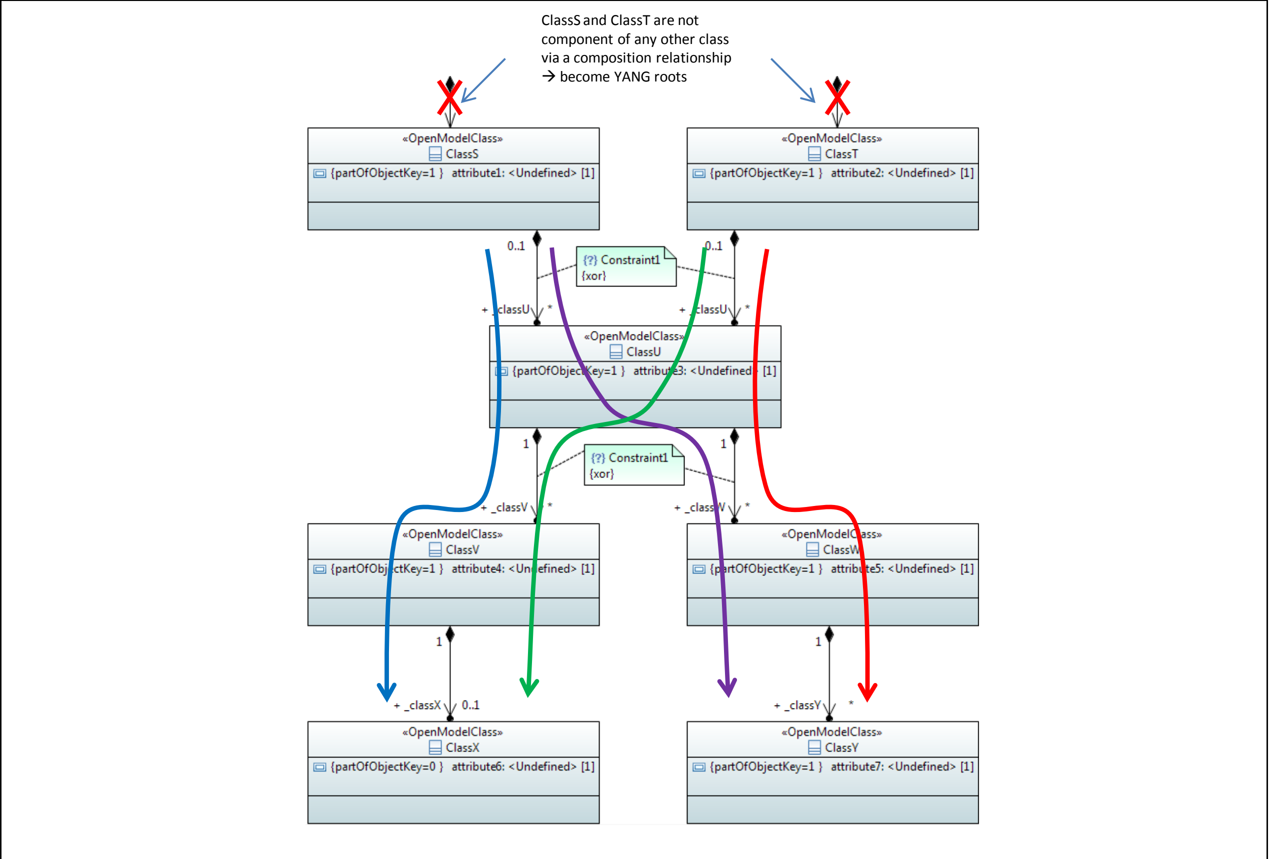
Figure 6.1: Example: Proxy Class Mapping

6.7 Building YANG Tree

The YANG data schema is tree structured. The tool analyses the UML composition associations and creates the YANG tree(s).

UML classes which are not component of any other class (via a composition relationship) are mapped to YANG tree roots. The YANG trees are created below the roots following the "lines" of composition associations in UML.

Table 6.6: Composition Associations Mapping to YANG Tree Example


ClassS and ClassT are not
component of any other class
via a composition relationship
→ become YANG roots
«OpenModelClass» ClassS
{partOfObjectKey=1 } attribute1: <Undefined> [1]
«OpenModelClass» ClassT
{partOfObjectKey=1 } attribute2: <Undefined> [1]
0..1
0..1
{?} Constraint1 {xor}
+ _classU *
+ _classU *
«OpenModelClass» ClassU
{partOfObjectKey=1 } attribute3: <Undefined> [1]
1
1
{?} Constraint1 {xor}
+ _classV *
+ _classW *
«OpenModelClass» ClassV
{partOfObjectKey=1 } attribute4: <Undefined> [1]
«OpenModelClass» ClassW
{partOfObjectKey=1 } attribute5: <Undefined> [1]
1
1
+ _classX 0..1
+ _classY *
«OpenModelClass» ClassX
{partOfObjectKey=0 } attribute6: <Undefined> [1]
«OpenModelClass» ClassY
{partOfObjectKey=1 } attribute7: <Undefined> [1]

```
/****************************
* Groupings
****************************/
grouping ClassS {
        leaf attribute1 {
                … }
}

grouping ClassT {
        leaf attribute2 {
                … }
}

grouping ClassU {
        leaf attribute3 {
                … }
}

grouping ClassV {
        leaf attribute4 {
                … }
}

grouping ClassW {
        leaf attribute5 {
                … }
}

grouping ClassX {
        leaf attribute6 {
                … }
}

grouping ClassY {
        leaf attribute7 {
                … }
}
```

```
/****************************
* Main Containers
****************************/
list ClassS {
        key attribute1;
        uses ClassS;
        …
        list ClassU {
                key attribute3;
                uses ClassU;
                …
                list ClassV {
                        key attribute4;
                        uses ClassV;
                        uses ClassX
                        …
                }
        }
}
list ClassS {
        key attribute1;
        uses ClassS;
        …
        list ClassU {
                key attribute3;
                uses ClassU;
                …
                list ClassW {
                        key attribute5;
                        uses ClassW;
                        …
                        list ClassY {
                                key attribute7;
                                uses ClassY;
                                …
                        }
                }
        }
}
```

```
list ClassT {
        key attribute2;
        uses ClassT;
        …
        list ClassU {
                key attribute3;
                uses ClassU;
                …
                list ClassV {
                        key attribute4;
                        uses ClassV;
                        uses ClassX
                        …
                }
        }
}
list ClassT {
        key attribute2;
        uses ClassT;
        …
        list ClassU {
                key attribute3;
                uses ClassU;
                …
                list ClassW {
                        key attribute5;
                        uses ClassW;
                        …
                        list ClassY {
                                key attribute7;
                                uses ClassY;
                                …
                        }
                }
        }
}
```

7 Tool – User Interactions

Some features of the mapping tool need additional instructions from the user which are gathered by the tool in interactions with the user.

7.1 YANG Module Header

RFC 6087bis [2] Appendix C defines a YANG Module Template which require information that is not contained in the UML model; see also section 10. The tool needs to ask the user for the following information and insert it into the YANG header:

1. module name
   The module name is the same as the UML model name; i.e., need not be asked by the tool.
   E.g., "UmlYangSimpleTestModel.uml" is mapped to "UmlYangSimpleTestModel.yang".
2. file name
   The file name consists of the module name and the date: "<module name>@2016-<mm>-<dd>.yang". The tool shall use the date of the mapping; i.e., need not be asked by the tool.
3. namespace
   The namespace must be a globally unique URI.
   E.g., urn:OpenSourceSDN:eagle:<module name>.
4. prefix
   The mapping tool provides the inferred module name and the user enters a corresponding prefix. It is recommended that the user should provide a prefix name with no more than 8-10 characters.
   E.g., The tool presents "UmlYangSimpleTestModel" as the module name and the user enters "stm" for the prefix.
5. organization
   Entered by the user.
   E.g., "Open Source SDN".
6. WG Web contact
   Entered by the user.
   E.g., "https://community.opensourcesdn.org/wg/EAGLE/dashboard".
7. WG List contact
   Entered by the user.
   E.g., "mailto:eagle@community.opensourcesdn.org".
8. WG Chair contact
   Entered by the user.
   I.e., "mailto:your-WG-chair@example.com".
9. Editor contact
   Entered by the user.
   I.e., "mailto:your-email@example.com".
10. description
    Shall be inferred from the "Applied comments" field of the UML model.
11. revision date
    Entered by the user in the form: <yyyy>-<mm>-<dd>.

12. revision description
    Entered by the user.

7.2 Lifecycle State Treatment

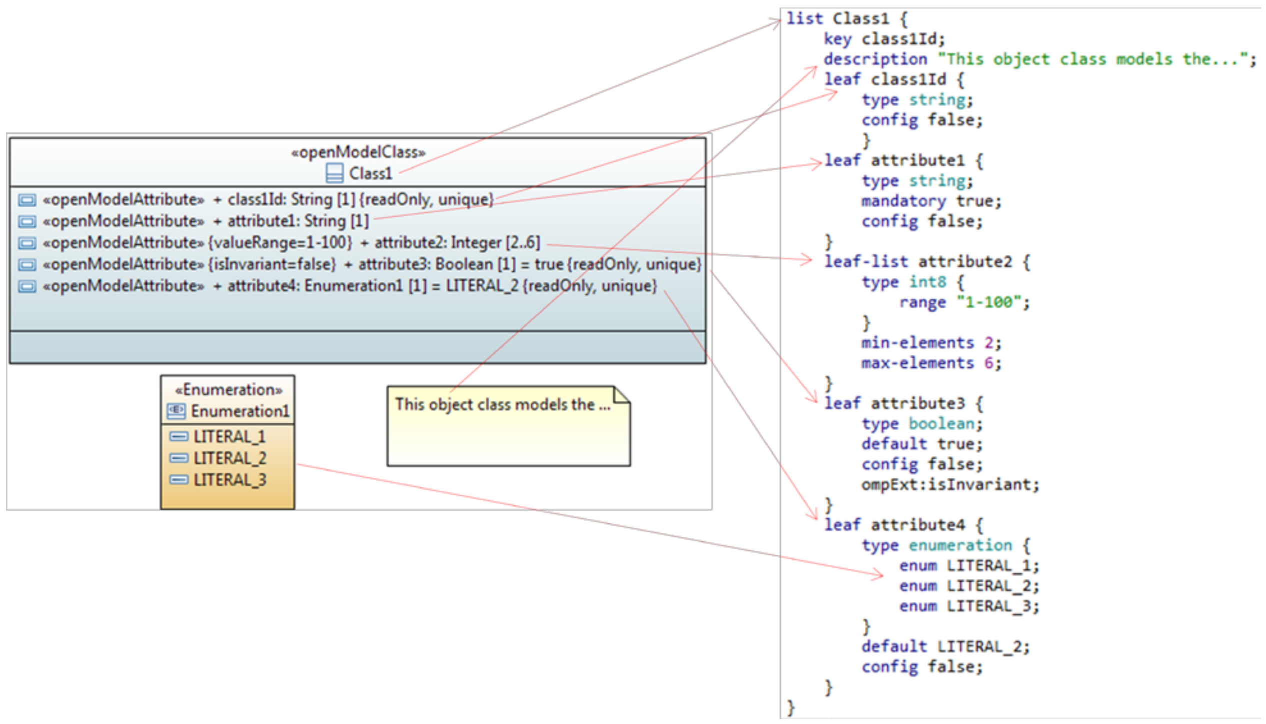
UML elements are annotated by at least one of the following lifecycle states (see also section 5.10):

- «Deprecated»
- «Experimental»
- «Faulty»
- «LikelyToChange»
- «Mature»
- «Obsolete»
- «Preliminary».

The tool shall allow the user to select – based on the lifecycle states – which UML elements are mapped; default is Mature only.

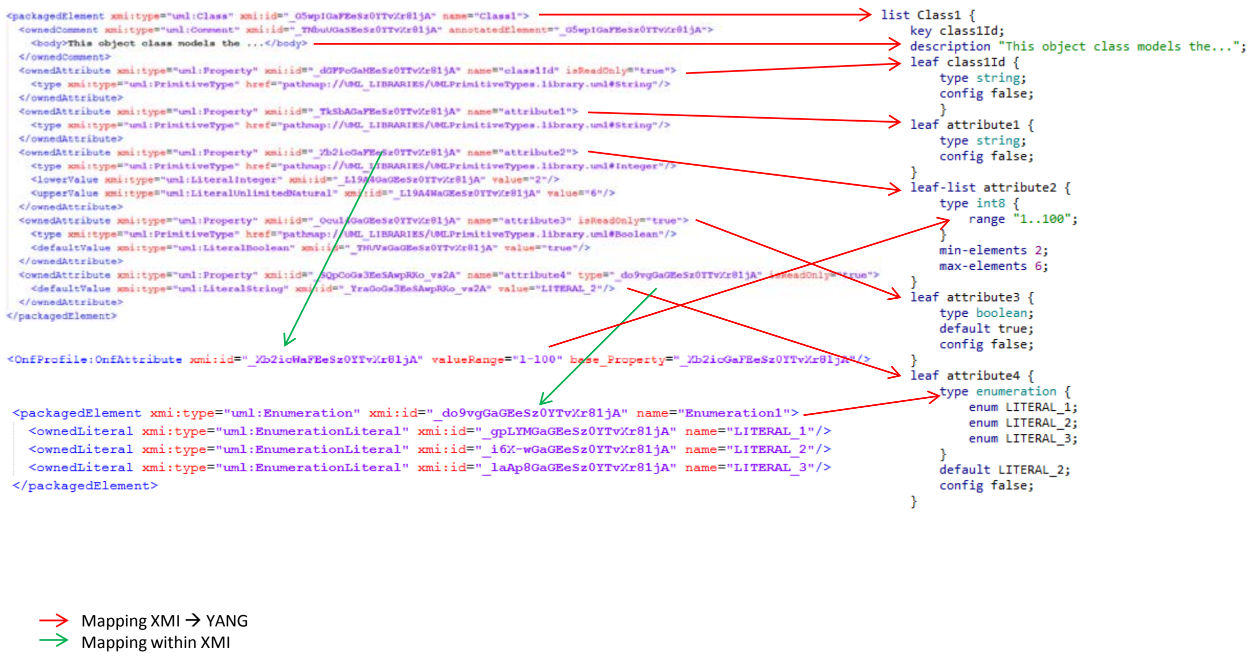
8 Mapping Basics

8.1 UML → YANG or XMI → YANG


«openModelClass»
Class1
«openModelAttribute» + class1Id: String [1] {readOnly, unique}
«openModelAttribute» + attribute1: String [1]
«openModelAttribute» {valueRange=1-100} + attribute2: Integer [2..6]
«openModelAttribute» {isInvariant=false} + attribute3: Boolean [1] = true {readOnly, unique}
«openModelAttribute» + attribute4: Enumeration1 [1] = LITERAL_2 {readOnly, unique}
«Enumeration»
Enumeration1
LITERAL_1
LITERAL_2
LITERAL_3
This object class models the ...

```
list Class1 {
    key class1Id;
    description "This object class models the...";
    leaf class1Id {
        type string;
        config false;
        }
    leaf attribute1 {
        type string;
        mandatory true;
        config false;
    }
    leaf-list attribute2 {
        type int8 {
            range "1-100";
        }
        min-elements 2;
        max-elements 6;
    }
    leaf attribute3 {
        type boolean;
        default true;
        config false;
        ompExt:isInvariant;
    }
    leaf attribute4 {
        type enumeration {
            enum LITERAL_1;
            enum LITERAL_2;
            enum LITERAL_3;
        }
        default LITERAL_2;
        config false;
    }
}
```

Figure 8.1: Example UML to YANG Mapping



```
<packagedElement xmi:type="uml:Class" xmi:id="_G5wpIGaFEeSz0YTvXr81jA" name="Class1">
  <ownedComment xmi:type="uml:Comment" xmi:id="_TNbuUGaSEeSz0YTvXr81jA" annotatedElement="_G5wpIGaFEeSz0YTvXr81jA">
    <body>This object class models the ...</body>
  </ownedComment>
  <ownedAttribute xmi:type="uml:Property" xmi:id="_dGFPoGaHEeSz0YTvXr81jA" name="class1Id" isReadOnly="true">
    <type xmi:type="uml:PrimitiveType" href="pathmap://UML_LIBRARIES/UMLPrimitiveTypes.library.uml#String"/>
  </ownedAttribute>
  <ownedAttribute xmi:type="uml:Property" xmi:id="_TkSbAGaFEeSz0YTvXr81jA" name="attribute1">
    <type xmi:type="uml:PrimitiveType" href="pathmap://UML_LIBRARIES/UMLPrimitiveTypes.library.uml#String"/>
  </ownedAttribute>
  <ownedAttribute xmi:type="uml:Property" xmi:id="_Xb2icGaFEeSz0YTvXr81jA" name="attribute2">
    <type xmi:type="uml:PrimitiveType" href="pathmap://UML_LIBRARIES/UMLPrimitiveTypes.library.uml#Integer"/>
    <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_L19A4GaGEeSz0YTvXr81jA" value="2"/>
    <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_L19A4WaGEeSz0YTvXr81jA" value="6"/>
  </ownedAttribute>
  <ownedAttribute xmi:type="uml:Property" xmi:id="_Ocul4GaGEeSz0YTvXr81jA" name="attribute3" isReadOnly="true">
    <type xmi:type="uml:PrimitiveType" href="pathmap://UML_LIBRARIES/UMLPrimitiveTypes.library.uml#Boolean"/>
    <defaultValue xmi:type="uml:LiteralBoolean" xmi:id="_THUVsGaGEeSz0YTvXr81jA" value="true"/>
  </ownedAttribute>
  <ownedAttribute xmi:type="uml:Property" xmi:id="_SQpCoGs3EeSAwpRKo_vs2A" name="attribute4" type="_do9vgGaGEeSz0YTvXr81jA" isReadOnly="true">
    <defaultValue xmi:type="uml:LiteralString" xmi:id="_YraGoGs3EeSAwpRKo_vs2A" value="LITERAL_2"/>
  </ownedAttribute>
</packagedElement>

<OnfProfile:OnfAttribute xmi:id="_Xb2icWaFEeSz0YTvXr81jA" valueRange="1-100" base_Property="_Xb2icGaFEeSz0YTvXr81jA"/>

 <packagedElement xmi:type="uml:Enumeration" xmi:id="_do9vgGaGEeSz0YTvXr81jA" name="Enumeration1">
   <ownedLiteral xmi:type="uml:EnumerationLiteral" xmi:id="_gpLYMGaGEeSz0YTvXr81jA" name="LITERAL_1"/>
   <ownedLiteral xmi:type="uml:EnumerationLiteral" xmi:id="_i6X-wGaGEeSz0YTvXr81jA" name="LITERAL_2"/>
   <ownedLiteral xmi:type="uml:EnumerationLiteral" xmi:id="_laAp8GaGEeSz0YTvXr81jA" name="LITERAL_3"/>
 </packagedElement>
```

```
list Class1 {
    key class1Id;
    description "This object class models the...";
    leaf class1Id {
        type string;
        config false;
        }
    leaf attribute1 {
        type string;
        config false;
    }
    leaf-list attribute2 {
        type int8 {
            range "1..100";
        }
        min-elements 2;
        max-elements 6;
    }
    leaf attribute3 {
        type boolean;
        default true;
        config false;
    }
    leaf attribute4 {
        type enumeration {
            enum LITERAL_1;
            enum LITERAL_2;
            enum LITERAL_3;
        }
        default LITERAL_2;
        config false;
    }
```

Figure 8.2: Example XMI (Papyrus) to YANG Mapping

8.2 Open Model Profile YANG Extensions

The additional UML artifact properties defined in the Open Model Profile are mapped as YANG extension statements.

<CODE BEGINS> file "onf-OpenModelProfileExtensions@2015-07-28.yang"

```
// Contents of "OpenModelProfileExtensions"
module OpenModelProfileExtensions {
    namespace "urn:ONF:OpenModelProfileExtensions";
    prefix "ompExt";

    organization
        "ONF (Open Networking Foundation) IMP Working Group";

    description
        "This module defines the Open Model Profile extensions for
         usage in other YANG modules.";

    revision 2015-07-28 {
        description "Initial revision";
    }

    //  extension statements

    extension isInvariant {
        description
          "Used with attribute definitions to indicate that the value
           of the attribute cannot be changed after it has been created.";
    }

    extension preCondition {
        description
          "Used with operation definitions to indicate the conditions
           that have to be true before the operation can be started
           (i.e., if not true, the operation will not be started at all
           and a general “precondition not met” error will be returned,
           i.e., exception is raised).";
        argument "condition-list";
    }

    extension postCondition {
        description
          "Used with operation definitions to indicate the state of
           the system after the operation has been executed (if
           successful, or if not successful, or if partially successful).
           Note that partially successful post-condition(s) can only
           be defined in case of non-atomic operations.
           Note that when an exception is raised, it should not be
           assumed that the post-condition(s) are satisfied.";
        argument "condition-list";
    }

    extension operationExceptions {
        description
```

```
          "Used with operation definitions to indicate the allowed
           exceptions for the operation.
           The model uses predefined exceptions which are split in
           2 types:
            - generic exceptions which are associated to all operations
              by default
            - common exceptions which needs to be explicitly associated
              to the operation.

            Note: These exceptions are only relevant for a protocol
            neutral information model. Further exceptions may be
            necessary for a protocol specific information model.
            Generic exceptions:
            • Internal Error: The server has an internal error.
            • Unable to Comply: The server cannot perform the operation.
              Use Cases may identify specific conditions that will result
              in this exception.
            • Comm Loss: The server is unable to communicate with an
              underlying system or resource, and such communication is
              required to complete the operation.
            • Invalid Input: The operation contains an input parameter
              that is syntactically incorrect or identifies an object
              of the wrong type or is out of range (as defined in the
              model or because of server limitation).
            • Not Implemented: The entire operation is not supported
              by the server or the operation with the specified input
              parameters is not supported.
            • Access Denied: The client does not have access rights
              to request the given operation.
            Common exceptions:
            • Entity Not Found: Is thrown to indicate that at least
              one of the specified entities does not exist.
            • Object In Use: The object identified in the operation
              is currently in use.
            • Capacity Exceeded: The operation will result in resources
              being created or activated beyond the capacity supported
              by the server.
            • Not In Valid State: The state of the specified object is
              such that the server cannot perform the operation. In
              other words, the environment or the application is not in
              an appropriate state for the requested operation.
            • Duplicate: Is thrown if an entity cannot be created because
              an object with the same identifier/name already exists.";
        argument "exception-list";
    }

    extension isOperationIdempotent {
        description
          "Used with operation definitions to indicate that the operation
           is idempotent.";
    }

//    extension isAtomic {
//        description
//          "Used with operation definitions to indicate that the operation
//           is atomic; i.e., has to be successful/unsuccessful as a whole.";
//    }
```

```
}
```

<CODE ENDS>

9 Reverse Mapping From YANG to UML

Given the many YANG drafts that have been created, in some cases it might be helpful to revert the mapping (i.e., from YANG to UML; re-engineer) so that comparison/analysis can be made.

Note: Since UML to YANG is not a 1:1 mapping, a tool supported reverse mapping of YANG to UML maybe different from origin UML.

10 Proposed Addendum 1: Requirements for the YANG Module header

This definition is following the YANG Module Template in Appendix C of RFC 6087bis [2].

<CODE BEGINS> file "<module name>@2016-<mm>-<dd>.yang"

```
module <module name> {
    namespace "urn:OpenSourceSDN:<module name>";
    prefix "<prefix>";

    import ietf-yang-types {
        prefix yang;
    }
    import ietf-inet-types {
        prefix inet;
    }
    import OpenModelProfileExtensions {
        prefix ompExt;
    }

    organization
        "Open Source SDN";

    contact
     "WG Web: <https://community.opensourcesdn.org/wg/EAGLE/dashboard>
      WG List: <mailto: <eagle@community.opensourcesdn.org>
      WG Chair: your-WG-chair
        <mailto:your-WG-chair@example.com>
      Editor: your-name
        <mailto:your-email@example.com>";

    description
        "This module defines …
         ….";

    revision <yyyy>-<mm>-<dd> {
        description "<revision description>";
    }

  /******************************************************************
  * package TypeDefinitions
  ******************************************************************/

…

  /******************************************************************
  * package ObjectClasses
  ******************************************************************/

…

  /******************************************************************
  * package Interfaces
  ******************************************************************/

…

  /******************************************************************
  * package Notifications
  ******************************************************************/

…
}
```

<CODE ENDS>

11 Proposed Addendum 2

Stereotype:Do not generate DS?

Using spanning tree algorithm?

Depth first search (DFS) & Breadth First Search (BFS)?