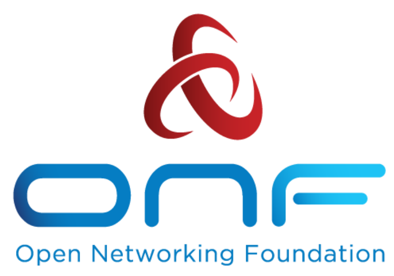


UML to YANG Mapping Guidelines

TR-531 v1.1-info
July 2018

Disclaimer

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Any marks and brands contained herein are the property of their respective owners.

Open Networking Foundation
1000 El Camino Real, Suite 100, Menlo Park, CA 94025
www.opennetworking.org

©2018 Open Networking Foundation. All rights reserved.

Open Networking Foundation, the ONF symbol, and OpenFlow are registered trademarks of the Open Networking Foundation, in the United States and/or in other countries. All other brands, products, or service names are or may be trademarks or service marks of, and are used to identify, products or services of their respective owners.

Important note

This Technical Recommendations has been approved by the OIMT Project TST but has not been approved by the ONF board. This Technical Recommendation has been approved under the ONF publishing guidelines for 'Informational' publications that allow Project technical steering teams (TSTs) to authorize publication of Informational documents. The designation of '-info' at the end of the document ID also reflects that the project team (not the ONF board) approved this TR.

Table of Contents

1 Introduction ... 7

2 References ... 7

3 Abbreviations ... 8

4 Overview ... 8

4.1 Documentation Overview ... 8

5 Mapping Guidelines ... 9

5.1 Introduction ... 9
5.2 Generic Mapping Guidelines ... 10
5.2.1 Naming Conventions Mapping ... 10
5.2.2 Mapping Scope ... 11
5.2.3 YANG Workarounds ... 11
5.3 Mapping of Classes ... 11
5.4 Mapping of Attributes ... 19
5.5 Mapping of Data Types ... 27
5.5.1 Generic Mapping of Primitive Data Types ... 27
5.5.2 Generic Mapping of Complex Data Types ... 28
5.5.3 Mapping of Common Primitive and Complex Data Types ... 30
5.5.4 Mapping of Enumeration Types ... 36
5.5.5 Mapping of Bit Field Data Types (preliminary) ... 40
5.6 Mapping of Relationships ... 43
5.6.1 Mapping of Associations ... 43
5.6.2 Mapping of Dependencies ... 56
5.7 Mapping of Interfaces (grouping of operations) ... 59
5.8 Mapping of Operations ... 59
5.9 Mapping of Operation Parameters ... 64
5.10 Mapping of Notifications ... 67
5.11 Mapping of UML Packages ... 70
5.12 Mapping of Lifecycle ... 70
5.13 Mapping Issues ... 71
5.13.1 YANG 1.0 or YANG 1.1? ... 71
5.13.2 Combination of different Associations? ... 71

6 Mapping Patterns ... 72

6.1 UML Recursion ... 72
6.1.1 Reference Based Approach ... 73
6.2 UML Conditional Pacs ... 75
6.3 {xor} Constraint ... 77
6.4 «Choice» Stereotype (obsolete) ... 80

 © Open Networking Foundation

6.5 Mapping of UML Support and Condition Properties 84
6.6 Proxy Class Association Mapping 86
6.7 Building YANG Tree 86

7 Generic UML Model and specific YANG Configuration Information 88
7.1 YANG Module Header 88
7.2 Lifecycle State Treatment 90

8 Mapping Basics 92
8.1 UML → YANG or XMI → YANG 92
8.2 Open Model Profile YANG Extensions 93

9 Reverse Mapping From YANG to UML 95

10 Requirements for the YANG Module Structure 95

11 Main Changes between Releases 97
11.1 Summary of main changes between version 1.0 and 1.1 97

12 Proposed Addendum 2 98

List of Figures

Figure 4.1: Specification Architecture 9
Figure 5.1: Pre-defined Packages in a UML Module 70
Figure 6.1: Example: Proxy Class Mapping 86
Figure 8.1: Example UML to YANG Mapping 92
Figure 8.2: Example XMI (Papyrus) to YANG Mapping 93

List of Tables

Table 5.1: Naming Conventions Mapping 10
Table 5.2: Class Mapping (Mappings required by currently used UML artifacts) 12
Table 5.3: Class Mapping (Mappings for remaining YANG substatements) 14
Table 5.4: Class Mapping Examples 15
Table 5.5: Attribute Mapping (Mappings required by currently used UML artifacts) 19
Table 5.6: Attribute Mapping (Mappings for remaining YANG substatements) 24
Table 5.7: Attribute Type Mapping Example 25
Table 5.8: Primitive Data Type Mapping 27
Table 5.9: Complex Data Type Mapping 28

 © Open Networking Foundation

Table 5.10: Complex Data Type Mapping Example 30
Table 5.11: Common Primitive and Complex Data Type Mapping 31
Table 5.12: Enumeration Type Mapping (Mappings required by currently used UML artifacts) 37
Table 5.13: Enumeration Type Mapping Example 38
Table 5.14: BITS Encoding Mapping Example 42
Table 5.15: Association Mapping Examples 45
Table 5.16: Association Mapping Summary 56
Table 5.17: Dependency Mapping Examples 56
Table 5.18: UML Interface Mapping 59
Table 5.19: Operation Mapping 60
Table 5.20: Interface/Operation Mapping Example 63
Table 5.21: Operation Exception Mapping Example 64
Table 5.22: Parameter Mapping 65
Table 5.23: Interface/Operation/Parameter Mapping Example 67
Table 5.24: Notification Mapping 68
Table 5.25: Notification Mapping Example 69
Table 5.26: UML Package to YANG Heading Mapping 70
Table 5.27: Lifecycle Mapping 71
Table 5.28: Combination of Associations Mapping Examples 72
Table 6.1: Recursion Mapping Examples 73
Table 6.2: Mapping of Conditional Packages 76
Table 6.3: {xor} Constraint Mapping Examples 77
Table 6.4: «Choice» Stereotype Mapping Examples 81
Table 6.5: Support and Condition Mapping Examples 84
Table 6.6: Composition Associations Mapping to YANG Tree Example 87

 © Open Networking Foundation

Document History

Version	Date	Description of Change
1.0	Sept. 20, 2016	Initial version.
1.1	July 2018	Version 1.1 A summary of main changes between version 1.0 and 1.1 is contained in section 11.1.

 © Open Networking Foundation

1 Introduction

This Technical Recommendation has been developed within IISOMI (**I**nformal **I**nter-**S**DO **O**pen **M**odel **I**nitiative) and is published by ONF.

IISOMI is 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.
The deliverables are developed in an open source community under the “Creative Commons Attribution 4.0 International Public License”.

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 (https://wiki.tools.ietf.org/wg/netmod/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 1.3” (https://www.opennetworking.org/software-defined-standards/models-apis/)
[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)

 © Open Networking Foundation

[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"
[12] Draft OpenConfig YANG best practices: http://www.openconfig.net/docs/style-guide/

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:

 © Open Networking Foundation

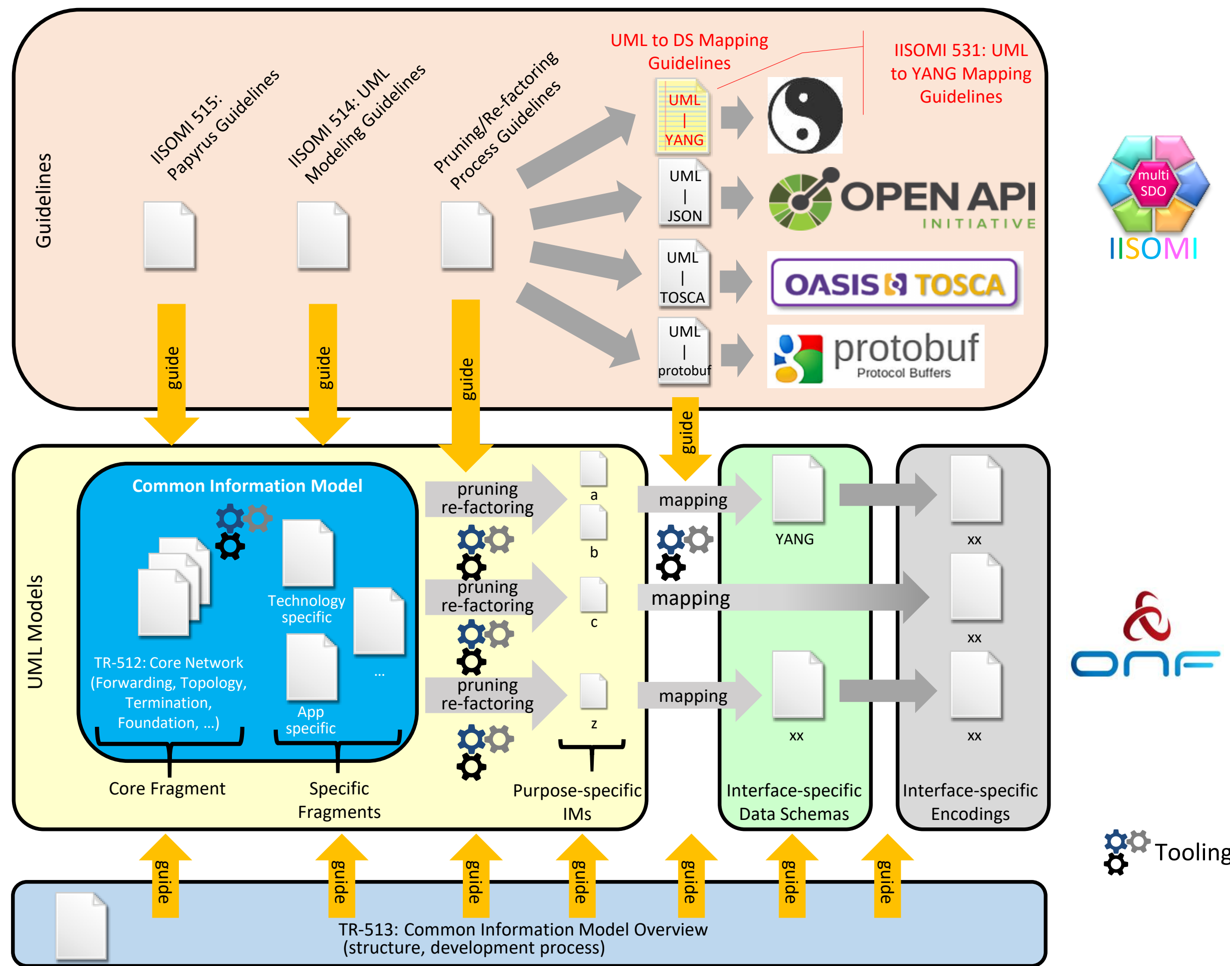


Figure 4.1: Specification Architecture

5 Mapping Guidelines

5.1 Introduction

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 5.13.

 © Open Networking Foundation

5.2 Generic Mapping Guidelines

5.2.1 Naming Conventions Mapping

UML and YANG use different naming conventions. UML mainly use camel case and YANG use only lower-case letters, numbers, and dashes in identifier names.

The grammar of an identifier in YANG is defined as (from [11]):

(ALPHA / "_")
*(ALPHA / DIGIT / "_" / "-" / ".")

Where:

- ALPHA = %x41-5A / %x61-7A
  ; A-Z / a-z
- DIGIT = %x30-39
  ; 0-9

Note: The restriction that identifiers should not start with the characters "xml" was removed from YANG in version 1.1.

Table 5.1: Naming Conventions Mapping

UML Naming	YANG Naming	Comments
lowerCamelCase	lower-camel-case	
_navigableAssociationEnd	navigable-association-end	
UpperCamelCase	upper-camel-case	
LITERAL_NAME_1	enum LITERAL_NAME_1 identity LITERAL_NAME_1	literals stay the same
Abc_Pac	abc-pac	
ABC ABCon OMS_TrailTerminationPoint	a-b-c a-b-con o-m-s-trail-termination-point	abbreviations
qom1024Times 15Min	qom-1024-times 15-min	letter strings and numbers
AbcClass	grouping: abc-class(-g) identity: abc-class	object classes suffix “-g” only if required in the config file
AbcDataType	typedef: abc-data-type grouping: abc-data-type	primitive data types complex data types
AbcEnumeration	typedef: abc-enumeration identity: abc-enumeration	enumerations

 © Open Networking Foundation

5.2.2 Mapping Scope

UML artifacts which are annotated by the «Example» stereotype are ruled out from mapping.

5.2.3 YANG Workarounds

- **Key attributes**
  Attributes marked as "partOfObjectKey" must be mapped as read/write, even when marked as WRITE_NOT_ALLOWED.

5.3 Mapping of Classes

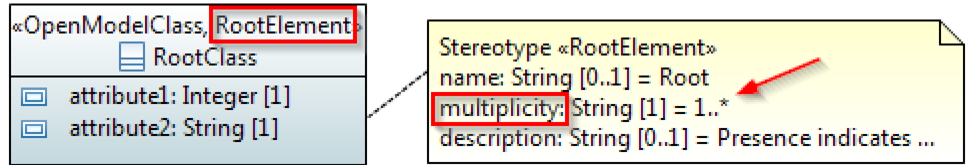
The classes are mapped in three steps.

Step 1: All classes are mapped to "grouping" statements; even if they only have a single attribute identified as partOfObjectKey.

Step 2: One or more additional groupings are defined for every class that has at least one attribute identified as partOfObjectKey. These "*class-reference-grouping*" statements group together the key-attributes of the specific class as well as the key attributes of its ancestors (starting from its immediate parent, all the way to the root-element). Thus it follows that one reference grouping would be defined per schema-tree-path for the class. (see section 5.6.1. for more details)

Step 3: The groupings are then integrated into the YANG schema tree:

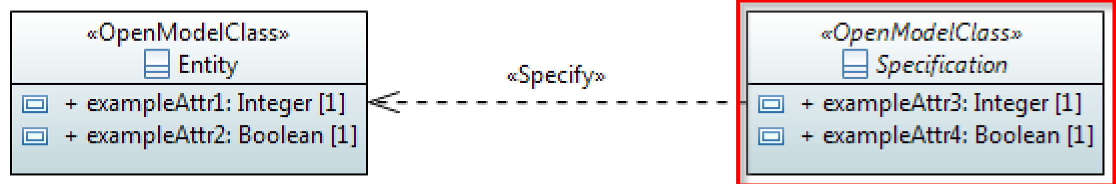
- Object classes identified as «RootElement» are incorporated as top-level Yang root elements using a "list" or "container" statement which in-turn includes the associated *class-grouping* via the "uses" sub-statement. The RootElement::multiplicity stereotype-property is used to determine whether to use a list or container.
  RootElement::multiplicity = 1 → container statement.
  RootElement::multiplicity > 1 → list statement.



  See example mapping in Table 5.4.
- Object class-attributes (UML association-end-role) which are referenced via a «StrictComposite» association are incorporated into the *class-grouping* statement of the owner class using a "list" or "container" statement which in-turn includes the attribute *class-grouping* via the "uses" sub-statement. The association-end-role multiplicity property is used to determine whether to use a list or container. See example mapping in Table 5.15.
- Object class-attributes (UML association-end-role) which are referenced via an «ExtendedComposite» association are incorporated into the *class-grouping* statement of the owner class directly via the "uses" sub-statement. See example mapping in Table 5.15.

 © Open Networking Foundation

- Object class-attributes (UML association-end-role) which are referenced via a pointer association (incl. «LifecycleAggregate» association) are incorporated into the *class-grouping* statement of the owner class using a "list" or "container" statement which in-turn includes the attribute *class-reference-grouping* via the "uses" sub-statement. The association-end-role multiplicity property is used to determine whether to use a list or container. See example mapping in Table 5.15.
- Specification classes (i.e., classes pointing via the «Specify» abstraction relationship to the entity classes) enhance their entity classes using the *augment* statement which in-turn includes the specification *class-grouping* directly via the "uses" sub-statement.



See example in Table 5.17.

Table 5.2: Class Mapping
(Mappings required by currently used UML artifacts)

Class → "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.
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 shall be identified from within the used grouping of the superclass.
isLeaf	??	
object identifier See OpenModelAttribute::partOfObjectKey in Table 5.5.		Attributes of the class can be marked as object identifier.

 © Open Networking Foundation

Class → “grouping” statement		
UML Artifact	YANG Artifact	Comments
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.
OpenInterfaceModel_Profile::objectCreationNotification [YES/NO/NA] (obsolete)	“notification” statement	See section 5.10 Goes beyond the simple “a notification has to be sent”; a tool can construct the signature of the notification by reading the created object.
OpenInterfaceModel_Profile::objectDeletionNotification [YES/NO/NA] (obsolete)	“notification” statement	See section 5.10 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).
OpenInterfaceModel_Profile::«RootElement»	“list“ or “container” statement depending on OpenInterfaceModel_Profile::RootElement::multiplicity	See guidelines above the table.
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
OpenModel_Profile::«Reference»	“reference” substatement	
OpenModel_Profile::«Example»	Ignore Example elements and all composed parts	

 © Open Networking Foundation

Class → “grouping” statement		
UML Artifact	YANG Artifact	Comments
OpenModel_Profile::lifecycleState	“status” substatement or “description” substatement	See section 5.12.
Proxy Class See section 6.6. XOR See section 6.3 OpenModel_Profile::«Choice» (obsolete) See section 6.4	“choice” substatement	
OpenModelClass::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.
OpenModelClass::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.
Root container		Presence statement = “This container shall not be deleted.”

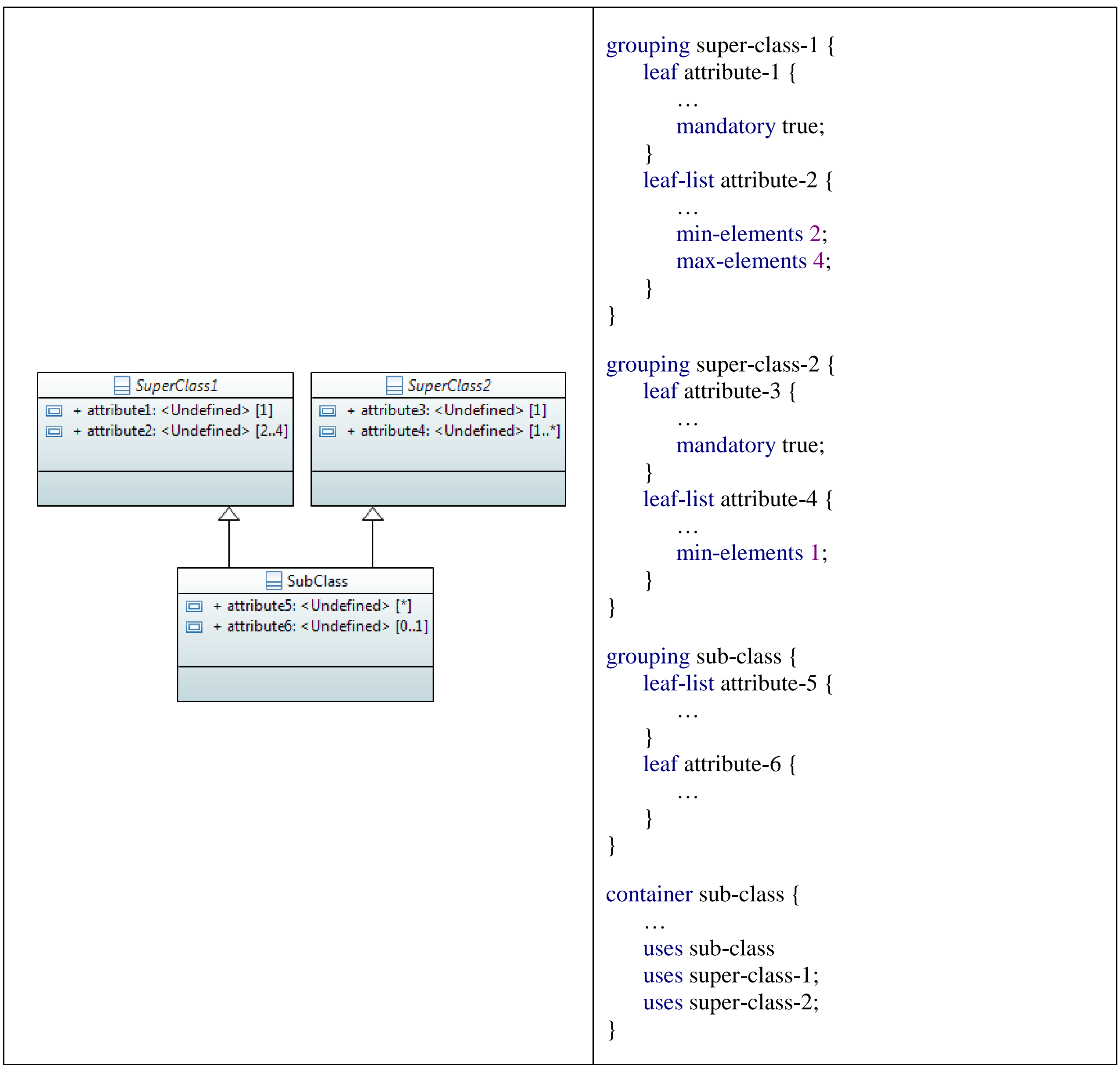
Table 5.3: Class Mapping
(Mappings for remaining YANG substatements)

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

 © Open Networking Foundation

Class → “grouping” statement → “list“ or “container” statement		
UML Artifact	YANG Artifact	Comments
{<constraint>}	“when” substatement	

Table 5.4: Class Mapping Examples




```
grouping super-class-1 {
    leaf attribute-1 {
        …
        mandatory true;
    }
    leaf-list attribute-2 {
        …
        min-elements 2;
        max-elements 4;
    }
}

grouping super-class-2 {
    leaf attribute-3 {
        …
        mandatory true;
    }
    leaf-list attribute-4 {
        …
        min-elements 1;
    }
}

grouping sub-class {
    leaf-list attribute-5 {
        …
    }
    leaf attribute-6 {
        …
    }
}

container sub-class {
    …
    uses sub-class
    uses super-class-1;
    uses super-class-2;
}
```

 © Open Networking Foundation

«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 opt-if-o-ch-rs-ss {
        description "RsSs path configuration
for an
        Interface";
        container if-current-application-code
{
            description "Current Application
code of the
            interface";
        uses opt-if-o-ch-application-code;
        }

        container if-supported-application-
codes {
            config false;
            description "Supported
Application codes of
                the interface";
            uses opt-if-o-ch-application-code-
list;
        }

        uses opt-if-o-ch-power;
        uses opt-if-o-ch-central-frequency;
    }
}

…
```

 © Open Networking Foundation

<table>
<tr>
<td>

«OpenModelClass, RootElement»
RootClass
attribute1: Integer [1]
attribute2: String [1]

Stereotype «RootElement»
name: String [0..1] = Root
multiplicity: String [1] = 1..1
description: String [0..1] = Presence indicates data-store is enabled

</td>
<td>

```
grouping root-class {
    …
    leaf attribute-1 {
        type uint64;
        …
    }
    leaf attribute-2 {
        type string;
        …
    }
}
container root {
    presence "Presence indicates data-store
is enabled";
    uses root-class;
    …
}
```

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

«OpenModelClass, RootElement»
RootClass
attribute1: Integer [1]
attribute2: String [1]

Stereotype «RootElement»
name: String [0..1] = Root
multiplicity: String [1] = 1..*
description: String [0..1] = ~~Presence indicates data-store is enabled~~

</td>
<td>

```
grouping root-class {
    …
    leaf attribute-1 {
        type uint64;
        …
    }
    leaf attribute-2 {
        type string;
        …
    }
}
list root {
    key attribute-1;
    uses root-class;
    …;
}
```

Note: If RootElement::multiplicity is > 1, a description property is ignored.

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

 © Open Networking Foundation

```
grouping top-level-ref {
    leaf top-level-attribute-1 {
        type leafref {
            path 'main-model:top-level/main-model:attribute-1';
        }
    }
}
grouping sub-level-2-ref {
    uses top-level-ref;
    leaf sub-level-2-key {
        type leafref {
            path 'main-model:sub-level-2/main-model:key';
        }
    }
}
grouping sub-level-4-ref {
    uses top-level-ref;
    leaf sub-level-4-key-1 {
        type leafref {
            path 'main-model:sub-level-4/main-model:key-1';
        }
    }
}
```

```
module main-model {
    …
grouping top-level {
    leaf attribute-1 {
        type int64;
    }
    …
}
grouping sub-level-1 {
    …
}
grouping sub-level-2 {
    …
    leaf key {
        type string;
    }
    …
}
grouping sub-level-3 {
    …
}
grouping sub-level-4 {
    …
    leaf key-1 {
        type string;
    }
    …
}
list root {
    key attribute-1;
    uses top-level;
    container sub-level-1 {
        uses sub-level-1;
    }
    list sub-level-2 {
        uses sub-level-2-ref;
        key 'sub-level-2-key';
    }
    container sub-level-3 {
        uses sub-level-3;
    }
    list sub-level-4 {
        uses sub-level-4-ref;
        key 'sub-level-4-key-1';
    }
```

 © Open Networking Foundation

grouping top-level-specification-ref {
  leaf top-level-specification-attribute-2 {
    type leafref {
      path 'specification-model:top-level-specification/specification-model:attribute-2';
    }
  }
}
	…
	}
	
	module specifications-model {
	  …
	  grouping top-level-specification {
	    leaf attribute-2 {
	      type string;
	      …
	    }
	    …
	  }
	  augment "/main-module:top-level" {
	    uses top-level-specification;
	    …
	  }
	}

5.4 Mapping of Attributes

Table 5.5: 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.
type	"type" substatement (built-in or derived type)	

 © Open Networking Foundation

Attribute → "leaf" (for single-valued attribute) or "leaf-list" (for multi-valued attribute) statement		
UML Artifact	YANG Artifact	Comments
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 e.g., positionSequence in OTN, layerProtocolList If the attribute is ordered and • read only (writeAllowed = WRITE_NOT_ALLOWED|CREATE_ONLY) = system • writeable = user
isUnique	No unique sub-statement in leaf-list.	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. See example in Table 5.7 below.
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

 © Open Networking Foundation

Attribute → “leaf” (for single-valued attribute) or “leaf-list” (for multi-valued attribute) statement		
UML Artifact	YANG Artifact	Comments
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.
OpenModelAttribute:: partOfObjectKey >0	list::“key” substatement	It is possible that the (abstract) superclass contains the key attribute for the instantiated subclass. Always read/write.
OpenModelAttribute::uniqueSet	list::“unique” substatement	See also Example in Table 5.7.
OpenModelAttribute::isInvariant	“extension” substatement → ompExt:is-invariant	See extensions YANG module in section 8.2.
OpenModelAttribute::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.
OpenModelAttribute::unsigned	See data type mapping in Table 5.11. Built-In Type::uintX	Only relevant for Integer typed attributes.

 © Open Networking Foundation

Attribute → “leaf” (for single-valued attribute) or “leaf-list” (for multi-valued attribute) statement		
UML Artifact	YANG Artifact	Comments
OpenModelAttribute::counter	ietf-yang-types::counterX ietf-yang-types::gaugeX ietf-yang-types::zero-based-counterX	Only relevant for Integer typed attributes.
OpenInterfaceModelAttribute::unit	“units” substatement	
OpenInterfaceModelAttribute::writeAllowed		
CREATE_ONLY + isInvariant = false		e.g., ODUflex with HAO
CREATE_ONLY + isInvariant = true		e.g., fixed size ODU, identifier provided by the client
UPDATE_ONLY + isInvariant = false		initial value provided by the system
CREATE_AND_UPDATE + isInvariant = false		unrestricted read/write
WRITE_NOT_ALLOWED + isInvariant = false	“config” substatement (false)	config default = true e.g., operationalState
WRITE_NOT_ALLOWED + isInvariant = true		e.g., identifier provided by the system
OpenInterfaceModelAttribute:: attributeValueChangeNotification [YES/NO/NA] (obsolete)	??	
OpenInterfaceModel_Profile:: bitLength	See data type mapping in Table 5.11.	
OpenInterfaceModelAttribute:: encoding - NA - BASE_64 - HEX - OCTET	- - ?? - ?? - ??	

 © Open Networking Foundation

Attribute → “leaf” (for single-valued attribute) or “leaf-list” (for multi-valued attribute) statement		
UML Artifact	YANG Artifact	Comments
OpenInterfaceModelAttribute:: bitsDefinition	Bits Built-In Type	Only if solution 1 is used
OpenModel_Profile:: «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.
OpenModel_Profile::«Reference»	“reference” substatement	
OpenModel_Profile::«Example»	Ignore Example elements and all composed parts	
OpenModel_Profile::lifecycleState	“status” substatement or “description” substatement	See section 5.12.
OpenModelAttribute::support	For conditional support only: “if-feature” substatement “when” substatement if condition can be formalized as XPath expression (i.e., it is conditioned by the value of another attribute)	Support and condition belong together. If the “support” is conditional, then the “condition” explains the conditions under which the class has to be supported.
OpenModelAttribute::condition		

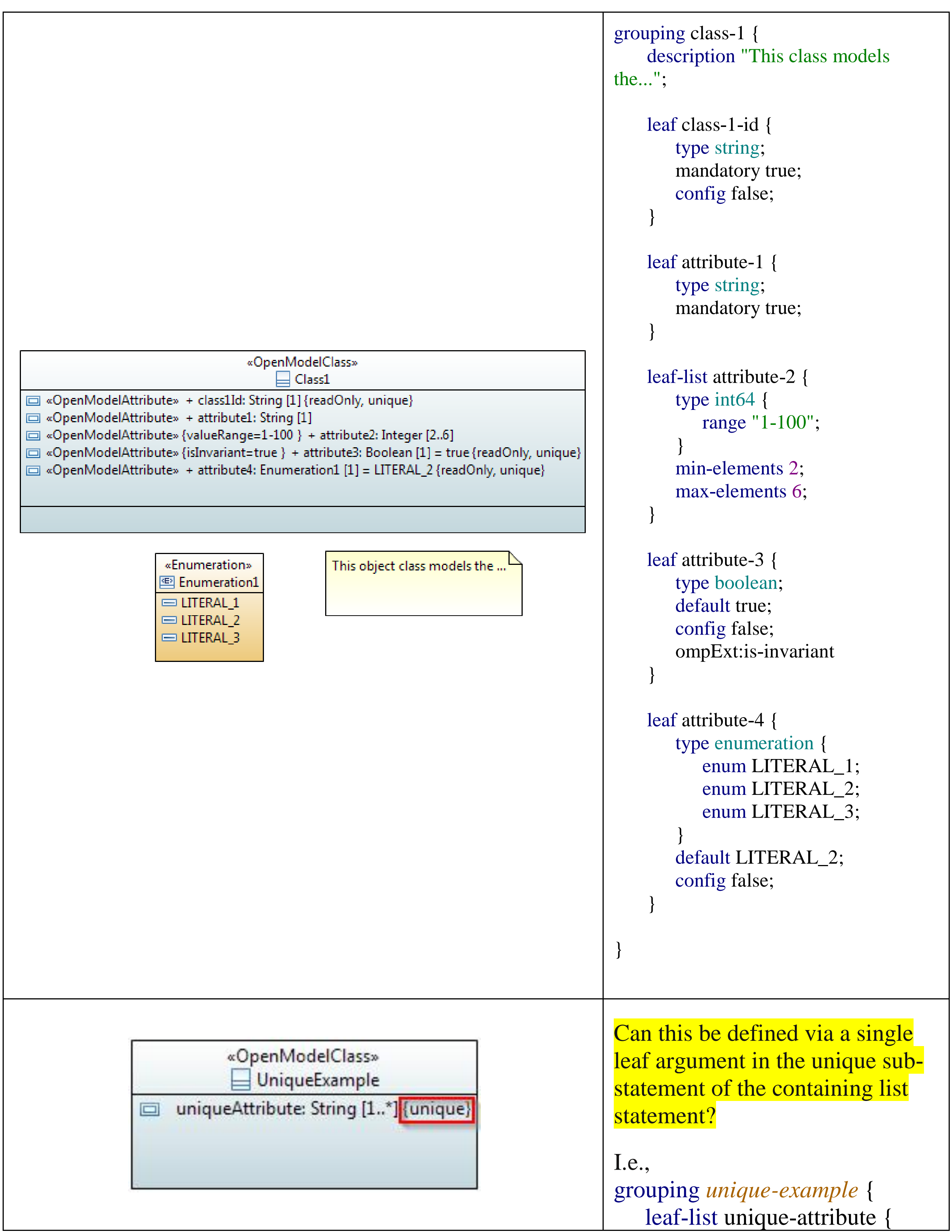
 © Open Networking Foundation

Table 5.6: 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	

 © Open Networking Foundation

Table 5.7: Attribute Type Mapping Example



UML	YANG

	grouping class-1 {
	    description "This class models the...";
	
	    leaf class-1-id {
	        type string;
	        mandatory true;
	        config false;
	    }
	
	    leaf attribute-1 {
	        type string;
	        mandatory true;
	    }
	
	    leaf-list attribute-2 {
	        type int64 {
	            range "1-100";
	        }
	        min-elements 2;
	        max-elements 6;
	    }
	
	    leaf attribute-3 {
	        type boolean;
	        default true;
	        config false;
	        ompExt:is-invariant
	    }
	
	    leaf attribute-4 {
	        type enumeration {
	            enum LITERAL_1;
	            enum LITERAL_2;
	            enum LITERAL_3;
	        }
	        default LITERAL_2;
	        config false;
	    }
	
	}

	Can this be defined via a single leaf argument in the unique sub-statement of the containing list statement?
	
	I.e.,
	grouping *unique-example* {
	    leaf-list unique-attribute {

 © Open Networking Foundation

	type String;
	min-elements 1;
	}
	}
	
	list unique-example {
	key "unique-attribute";
	unique "unique-attribute";
	uses *unique-example*;
	}
	
	------- 	
	
	You could transform:
	
	leaf-list numbers {
	type int16;
	unique true; // NOT ALLOWED IN YANG !!!!
	}
	
	to:
	
	list numbers {
	key number ;
	leaf number { type int16; }
	}
	
	If this is config=true data uniqueness is guaranteed for the leaf-list itself.
	
	If this is config=false the key statement guarantees uniqueness. If you don't want uniqueness just remove the key statement.
	
	Note: this will change the protocol encoding by adding in a new "numbers" wrapper element.
	
	So… a recommendation could be… If you really need a non-configuration data leaf-list that is guaranteed to be 'unique'… you need to create a list with a leaf in it and a "key", instead of using leaf-list.

 © Open Networking Foundation

«OpenModelClass»
<UniqueSetExample>
{partOfObjectKey=0 , uniqueSet=[1]} attribute1: String [1]
{uniqueSet=[1, 2]} attribute2: Integer [1]
{uniqueSet=[2]} attribute3: String [1]

```
grouping unique-set-example {
    leaf attribute-1 {
        type string;
    }
    leaf attribute-2 {
        type int64;
    }
    leaf attribute-3 {
        type string;
    }
}

list unique-set-example {
    key "attribute-1";
    unique "attribute-1 attribute-2";
    unique "attribute-2 attribute-3";
    uses unique-set-example;
}
```

5.5 Mapping of Data Types

Various kinds of data types are defined in UML:

1. Primitive Data Types (not further structured; e.g., Integer, MAC address)
2. Complex Data Types (containing attributes; e.g., Host which combines ipAddress and domainName)
3. Enumerations
4. Bit field Data Types (also known as flags)

Data Types are used as type definition of attributes and parameters.

5.5.1 Generic Mapping of Primitive Data Types

Table 5.8: 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.

 © Open Networking Foundation

Primitive Data Type → “typeDef” statement		
UML Artifact	YANG Artifact	Comments
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	
OpenModel_Profile::«Reference»	“reference” substatement	
OpenModel_Profile::«Example»	Ignore Example elements and all composed parts	
OpenModel_Profile::lifecycleState	“status” substatement or “description” substatement	See section 5.12.

5.5.2 Generic Mapping of Complex Data Types

Table 5.9: Complex Data Type Mapping

Complex Data Type containing only one attribute → “typedef” statement; see Table 5.8 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	

 © Open Networking Foundation

Complex Data Type containing only one attribute → "typedef" statement; see Table 5.8 Complex Data Type containing more than one attribute → "grouping" statement		
UML Artifact	YANG Artifact	Comments
XOR See section 6.3 OpenModel_Profile::«Choice» See section 6.4	"choice" substatement	
OpenModel_Profile::«Reference»	"reference" substatement	
OpenModel_Profile::«Example»	Ignore Example elements and all composed parts	
OpenModel_Profile::«Exception»	??	
OpenModel_Profile::lifecycleState	"status" substatement or "description" substatement	See section 5.11.
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.

 © Open Networking Foundation

Table 5.10: Complex Data Type Mapping Example

UML	YANG

«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]
	grouping class-r
	…
	container attribute-current {
	…
	uses data-type-a;
	}
	list attribute-potential {
	key "attribute-1 attribute-3";
	…
	min-elements 1;
	uses data-type-a;
	}
	}
	------------------------------------- 	
	grouping data-type-a {
	…
	leaf attribute-1 {
	…
	type string;
	mandatory true;
	}
	leaf attribute-2 {
	…
	type int64;
	}
	leaf-list attribute-3 {
	…
	type string;
	mandatory true;
	}
	}

5.5.3 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.

 © Open Networking Foundation

Table 5.11: 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 • 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 and unsigned = default (i.e., false).
«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	

 © Open Networking Foundation

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.

 © Open Networking Foundation

UML CommonDataTypes → YANG Built-In Types, ietf-yang-types, ietf-inet-types		
UML Artifact	YANG Artifact	Comments

«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
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

 © Open Networking Foundation

UML CommonDataTypes → YANG Built-In Types, ietf-yang-types, ietf-inet-types		
UML Artifact	YANG Artifact	Comments
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	
«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	

 © Open Networking Foundation

UML CommonDataTypes → YANG Built-In Types, ietf-yang-types, ietf-inet-types		
UML Artifact	YANG Artifact	Comments
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	
«Enumeration» IpVersion UNKNOWN IP_V4 IP_V6	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	??	
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

 © Open Networking Foundation

UML CommonDataTypes → YANG Built-In Types, ietf-yang-types, ietf-inet-types		
UML Artifact	YANG Artifact	Comments
	} } }	

5.5.4 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 indirectly 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.

Note: Since the literal names are mapped to identities and all identity names have to be unique within a module, all extendable enumerations have to have different literal names. Therefore, the following rules apply (see also example mapping in Table 5.13):

- Use the "UML literal name styled" enumeration name as the base identity name
- Prefix the literal identity names with the "UML literal name styled" enumeration name
- Convert the UML enumeration name according to the YANG naming convention and use this name as the name of the typedef.

 © Open Networking Foundation

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.

The Enumeration property "isLeaf" is used to direct the mapping tool to perform the appropriate mapping. isLeaf = true → maps to typedef with enum statement; isLeaf = false → maps to identity/base.

Table 5.12: Enumeration Type Mapping
(Mappings required by currently used UML artifacts)

Fixed Enumeration Type (isLeaf = true) → "typedef" with "enum" statement Extendable Enumeration Type → "identity"/"base" pattern		
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 identity name	
literal integer	"value" substatement	Example: Properties Model Validation Documentatio interval10s UML / Comments / Profile / Style / Appearance / Rulers And Grid Name: interval10s Label: Visibility: public Specification: 5
«Reference»	"reference" substatement	
«Example»	Ignore Example elements and all composed parts	
lifecycleState	"status" substatement or "description" substatement	See section 5.12.

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

 © Open Networking Foundation

Table 5.13: Enumeration Type Mapping Example

 © Open Networking Foundation

«OpenModelClass»
ClassH
attribute1: Enumeration1 [1] = LITERAL_2

«Enumeration»
Enumeration1
LITERAL_1
LITERAL_2
LITERAL_3

Using "enumeration" (isLeaf = true)

direct usage (not recommended):

```
grouping class-h {
    …
    leaf attribute-1 {
        type enumeration {
            enum LITERAL_1;
            enum LITERAL_2;
            enum LITERAL_3;
        default LITERAL_2;
        mandatory true;
        }
    }
}
```

indirect usage:

```
typedef enumeration-1 {
    type enumeration {
        enum LITERAL_1;
        enum LITERAL_2;
        enum LITERAL_3;
    }
}
container class-h {
    …
    leaf attribute-1 {
        type enumeration-1;
        default LITERAL_2;
    }
}
```

 © Open Networking Foundation

	Using "identity"/"base" (isLeaf = false)
	// an empty identity value is a "base identity"
	// i.e. it provides the reference name for a set of values
	identity ENUMERATION_1;
	identity ENUMERATION_1_LITERAL_1 {
	    // the "base" statement qualifies this identity value
	    // as belonging to the ENUMERATION_1 set
	    base ENUMERATION_1;
	}
	identity ENUMERATION_1_LITERAL_2 {
	    base ENUMERATION_1;
	}
	identity ENUMERATION_1_LITERAL_3 {
	    base ENUMERATION_1;
	}
	……
	typedef enumeration-1 {
	    type identityref {
	        // "identityref" defines the associated set
	        base ENUMERATION_1;
	    }
	}
	......
	leaf attribute-1 {
	    type enumeration-1;
	}

5.5.5 Mapping of Bit Field Data Types (preliminary)

Solution 1:

Attributes/parameters which are typed by the Bits primitive type are mapped to the YANG "bits" built-in type representing a bit set where each bit has an assigned name, a position a support definition and an optional description.

The default setting of each bit is defined in the default value of the Bits typed attribute and are added to the default sub-statement of the corresponding leaf statement as an ordered space-separated list.

Solution 2:

 © Open Networking Foundation

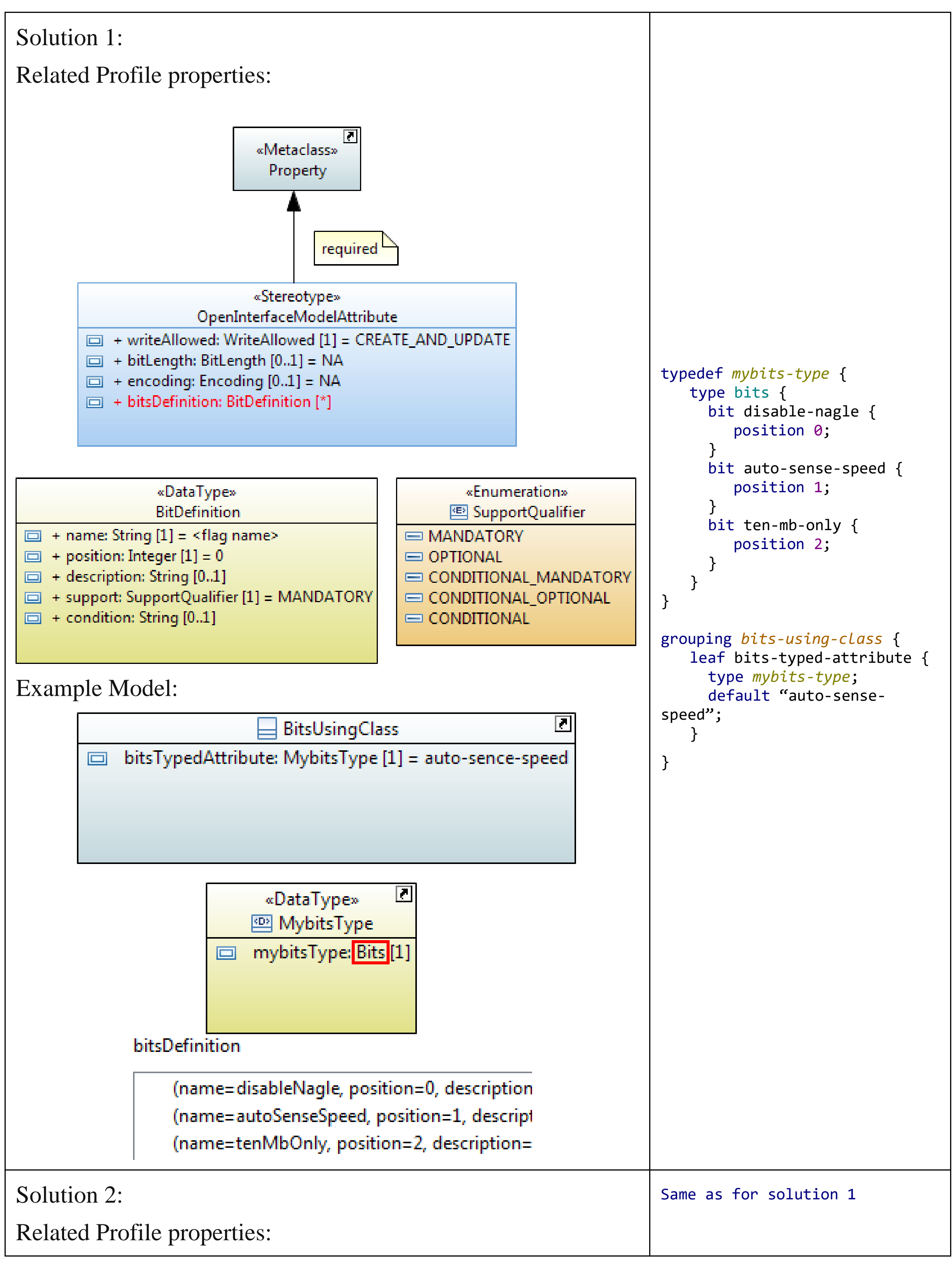
Data types which are annotated by the «Bits» Stereotype are mapped to a YANG typedef using the "bits" built-in type. Each attribute of the data type is mapped to a "bit" sub-statement with the following mapping of the "bit" sub-statements:

- Applied comment → description
- OpenModel_Profile::OpenModelAttribute::support/condition → if-feature
- OpenInterfaceModel_Profile::Bit::position → position
- OpenModel_Profile::Reference::reference → reference
- OpenModel_Profile::<Lifecycle> → status

The default setting of each bit is defined in the default value of the bits typed attribute and is added to the default sub-statement of the corresponding leaf statement as an ordered space-separated list.

 © Open Networking Foundation

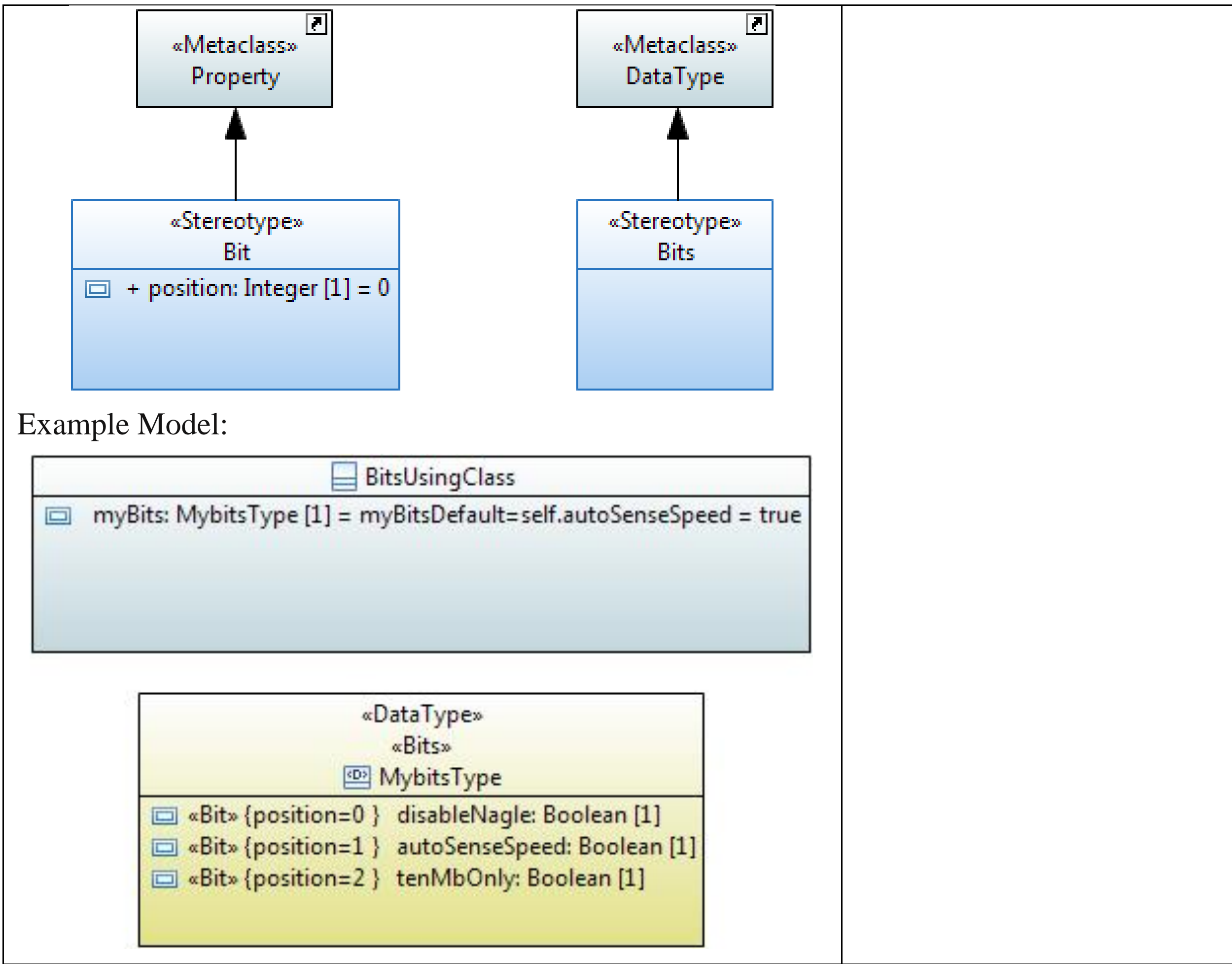
Table 5.14: BITS Encoding Mapping Example



Solution 1:
Related Profile properties:
Example Model:
	typedef mybits-type {
	   type bits {
	     bit disable-nagle {
	        position 0;
	     }
	     bit auto-sense-speed {
	        position 1;
	     }
	     bit ten-mb-only {
	        position 2;
	     }
	   }
	}
	
	grouping bits-using-class {
	   leaf bits-typed-attribute {
	     type mybits-type;
	     default “auto-sense-speed”;
	   }
	}

Solution 2: Related Profile properties:	Same as for solution 1

 © Open Networking Foundation



5.6 Mapping of Relationships

5.6.1 Mapping of Associations

Pointers and shared aggregation associations are per default passed by reference (i.e., contain only the reference (name, identifier, address) to the referred instance(s) when being transferred across the interface). Composition aggregation, «StrictComposite» and «ExtendedComposite» associations are always passed by value (i.e., contain the complete information of the instance(s) when being transferred across the interface).

This lead to the following 4 kinds of association scenarios:

1. Pointers and shared aggregations which are passed by reference
2. Composition aggregation and «StrictComposite» associations which are passed by value
3. «ExtendedComposite» associations which can also be somehow treated as passed by value.
4. «LifecycleAggregate» associations which adds a lifecycle dependency to a class (in addition to an already existing composition aggregation lifecycle dependency).

Corresponding mapping examples are contained in Table 5.15.

 © Open Networking Foundation

References in YANG can be expressed by leafref. Multiple leafrefs for references to object instances are defined by the mapping tool in a single grouping. Those reference groupings are contained in the "grouping statements for object references" section of the YANG module. The mapping tool needs to define a reference grouping per naming path/tree of the class for all classes that have at least one attribute identified as partOfObjectKey.
The name of the reference grouping is composed by <class name> + "-ref" in case of a single reference and + "-ref-1", "-ref-2", etc. in case of multiple references.
The name of the leafref typed leaf is composed by <class name> + "-" + <key attribute name>.
All navigable pointer and shared aggregation associations need an additional "require-instance" = false sub-statement.

The «LifecycleAggregate» association cannot define the operational behavior which can be seen from containing or contained class point of view. Four deletion policies can be distinguished (see «LifecycleAggregate» mapping example in Table 5.15):

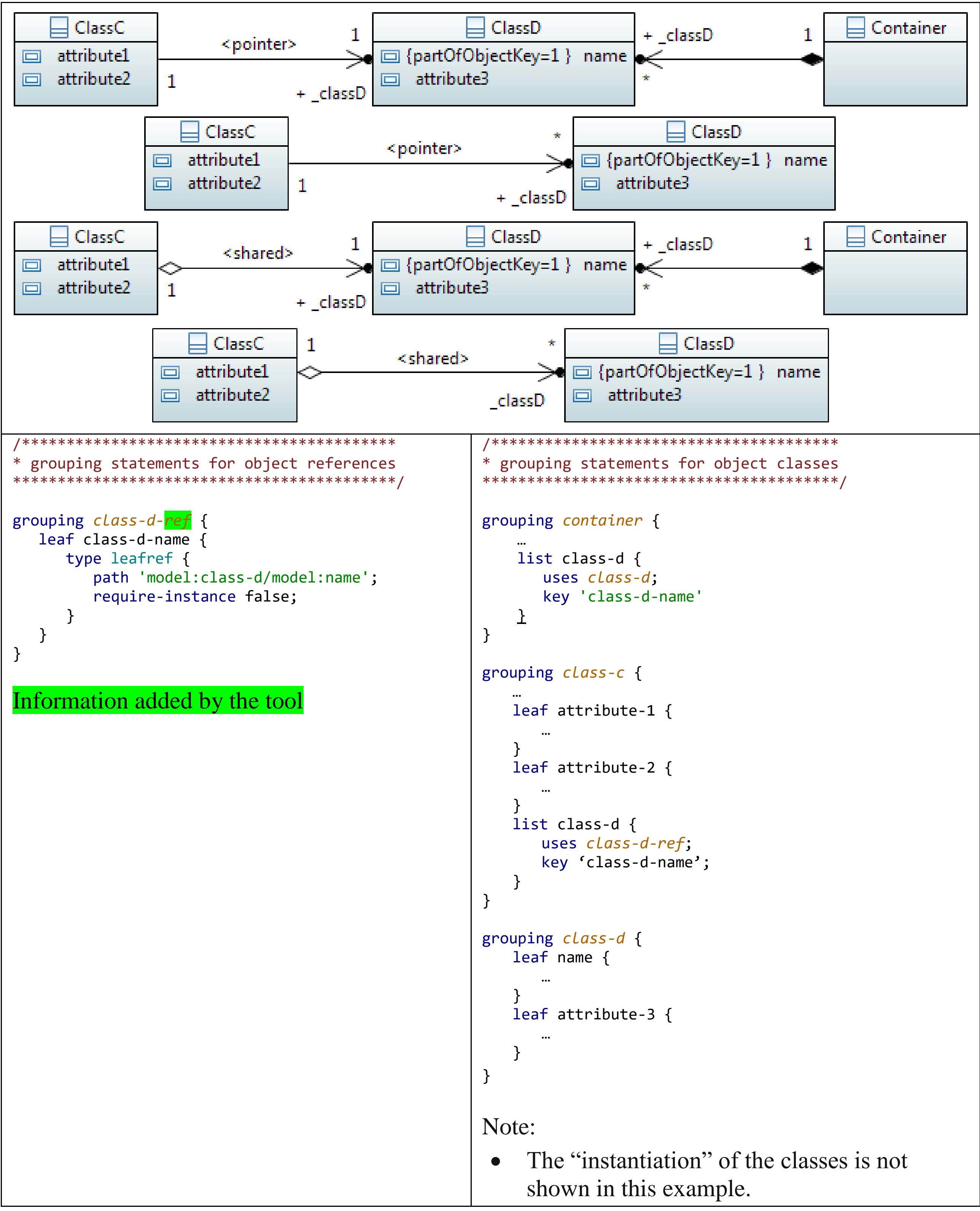
1. Deletion of containing OwningClass instance deletes all contained instances
2. Deletion of containing GroupingClass instance deletes aggregated instances
   → This behavior cannot be expressed in YANG
3. Containing OwningClass instance must not be deleted as long as contained instances exist
   → This behavior cannot be expressed in YANG
4. Containing GroupingClass instance must not be deleted as long as contained instances exist
   → This behavior can be expressed in YANG using leafref with "require-instance" sub-statement = true (which is the default value)

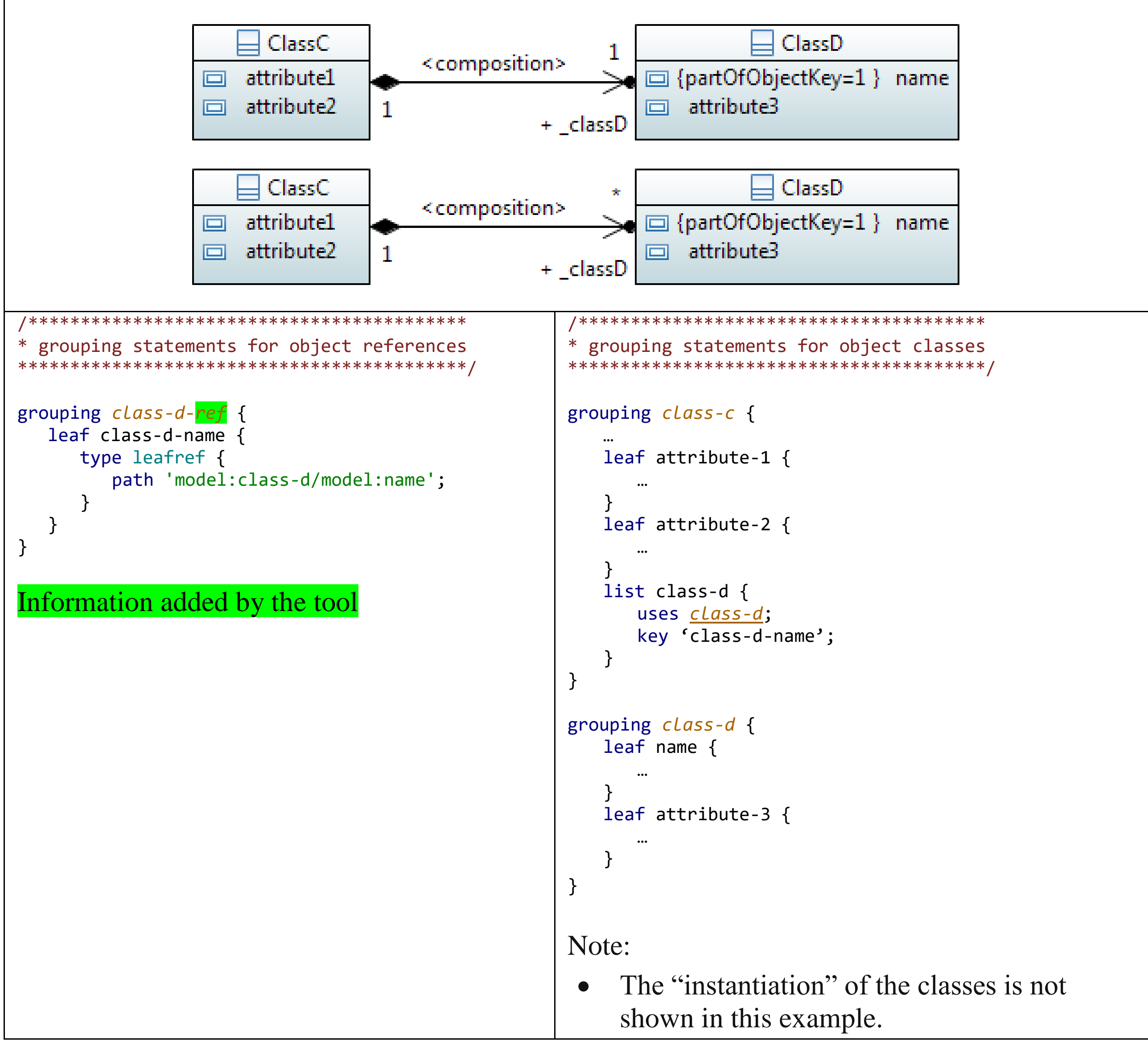
See example mapping in Table 5.15 below.

 © Open Networking Foundation

Table 5.15: Association Mapping Examples

 © Open Networking Foundation


ClassC
attribute1
attribute2
<pointer>
1
1
+ _classD
ClassD
{partOfObjectKey=1 } name
attribute3
+ _classD
1
*
Container
<shared>
_classD
/***
* grouping statements for object references
***/
grouping class-d-ref {
leaf class-d-name {
type leafref {
path 'model:class-d/model:name';
require-instance false;
}
}
}
Information added by the tool
/**
* grouping statements for object classes
**/
grouping container {
…
list class-d {
uses class-d;
key 'class-d-name'
}
}
grouping class-c {
…
leaf attribute-1 {
…
}
leaf attribute-2 {
…
}
list class-d {
uses class-d-ref;
key 'class-d-name';
}
}
grouping class-d {
leaf name {
…
}
leaf attribute-3 {
…
}
}
Note:
The "instantiation" of the classes is not shown in this example.




```
/*******************************************
* grouping statements for object references
*******************************************/

grouping class-d-ref {
   leaf class-d-name {
      type leafref {
         path 'model:class-d/model:name';
      }
   }
}
```

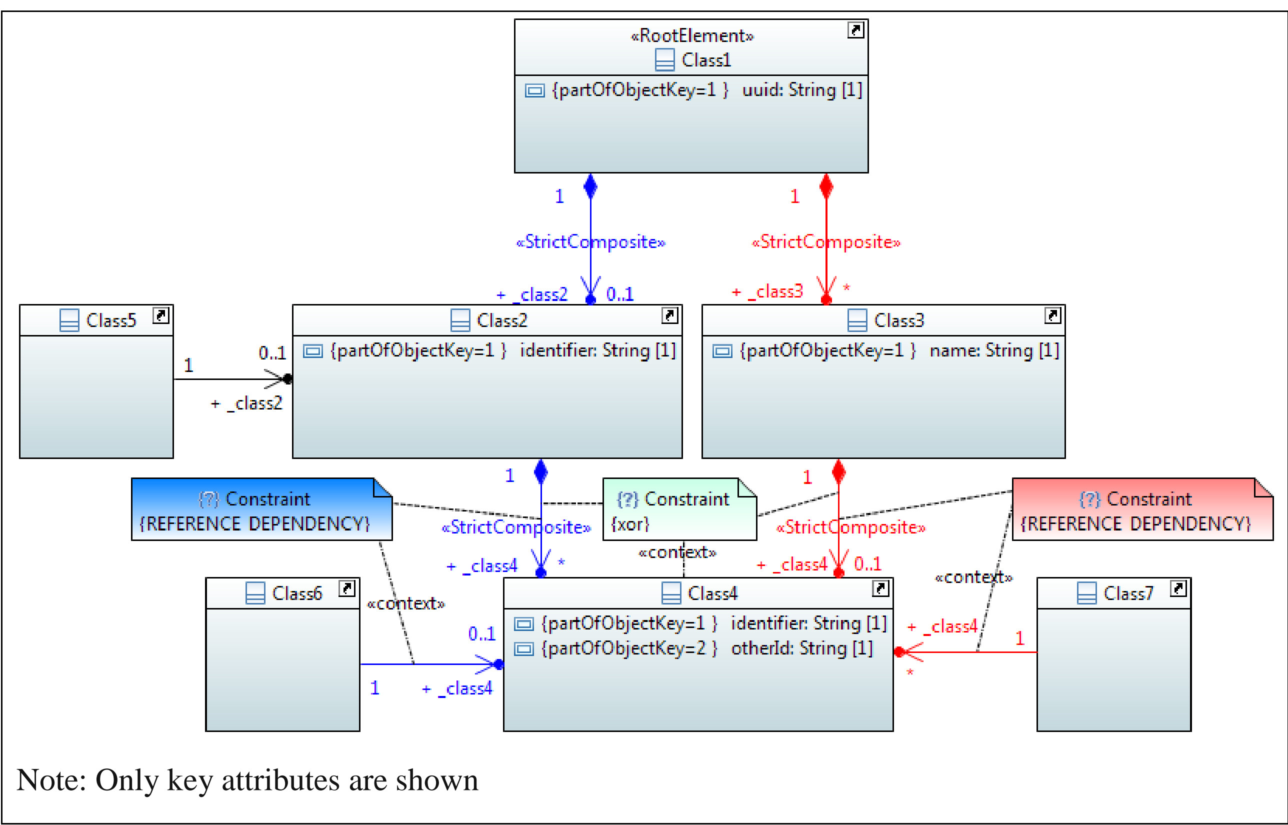
Information added by the tool

```
/****************************************
* grouping statements for object classes
****************************************/

grouping class-c {
    …
    leaf attribute-1 {
       …
    }
    leaf attribute-2 {
       …
    }
    list class-d {
       uses class-d;
       key ‘class-d-name’;
    }
}

grouping class-d {
    leaf name {
       …
    }
    leaf attribute-3 {
       …
    }
}
```

Note:

- The “instantiation” of the classes is not shown in this example.

 © Open Networking Foundation



 © Open Networking Foundation

<table>
<tr>
<td>

```
/*******************************************
* grouping statements for object references
********************************************/

grouping class-1-ref {
   leaf class-1-uuid {
     type leafref {
        path 'model:class-1/model:uuid';
     }
   }
}

grouping class-2-ref {
   uses class-1-ref;
   leaf class-2-identifier {
     type leafref {
        path 'model:class-2/model:uuid';
     }
   }
}

grouping class-3-ref {
   uses class-1-ref;
   leaf class-3-name {
     type leafref {
        path 'model:class-3/model:name';
     }
   }
}

grouping class-4-ref-1 {
   uses class-2-ref;
   leaf class-4-identifier {
     type leafref {
        path 'model:class-4/model:uuid';
     }
   }
   leaf class-4-otherId {
     type leafref {
        path 'model:class-4/model:otherId';
     }
   }
}

grouping class-4-ref-2 {
   uses class-3-ref;
   leaf class-4-identifier {
     type leafref {
        path 'model:class-4/model:uuid';
     }
   }
   leaf class-4-otherId {
     type leafref {
        path 'model:class-4/model:otherId';
     }
   }
}
```

Information added by the tool

</td>
<td>

```
/***************************************
* grouping statements for object classes
****************************************/

grouping class-5 {
   container class-2 {
     uses class-2-ref;
   }
}

grouping class-6 {
   container class-4 {
     uses class-4-ref-1;
   }
}

grouping class-7 {
   list class-4 {
     uses class-4-ref-2;
     key 'class-1-uuid class-3-name class-4-uuid class-
4-other-id';
   }
}
```

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

 © Open Networking Foundation

ClassC 1 «StrictComposite» 1 ClassD
attribute1 {partOfObjectKey=1 } name
attribute2 _classD attribute3

ClassC 1 «StrictComposite» * ClassD
attribute1 {partOfObjectKey=1 } name
attribute2 _classD attribute3

```
grouping class-c {
    …
    leaf attribute-1 {
        …
    }
    leaf attribute-2 {
        …
    }
//start choice
//multiplicity = 1
    container class-d {
        uses class-d;
    }
//multiplicity = *
    list class-d {
        key "name";
        uses class-d;
    }
//end choice
}

grouping class-d {
    leaf name {
        …
    }
    leaf attribute-3 {
        …
    }
}

container class-c {
    …
    uses class-c;
}
```

Lifecycle requirement from UML is enforced inYANG.

 © Open Networking Foundation

<table>
<tr>
<td>

</td>
<td>

```
grouping class-a {
    …
    leaf attribute-1 {
        …
    }
    leaf attribute-2 {
        …
    }
    uses class-b;
      }
}

grouping class-b {
    leaf attribute 3 {
        …
    }
    leaf attribute-4 {
        …
    }
}

container class-a {
    …
    uses class-a;
}
```

Lifecycle requirement from UML is enforced inYANG.
</td>
</tr>
</table>

 © Open Networking Foundation

```
/*****************************************
******
* grouping statements for object
classes
******************************************
*****/

grouping owning-class {
  list shared-part-class {
     key 'name';
     uses shared-part-class;
     description "none";
  }
  list grouping-class {
     key 'name';
     uses grouping-class;
     description "none";
  }
  description "none";
}

grouping grouping-class {
  leaf-list shared-part-class {
     type leafref {
         path '/model:owning-
class/model:shared-part-
class/model:name';
     }
     description "none";
  }
  leaf name {
     type string;
     description "none";
  }
  description "none";
}

grouping shared-part-class {
  leaf name {
     type string;
     description "none";
  }
  leaf grouping-class { /* in case of
exclusive mode */
     type leafref {
```

 © Open Networking Foundation

	path '/model:owning-class/model:grouping-class/model:name';
	require-instance true; /* not necessary since true is default */
	}
	description "none";
	}
	leaf-list grouping-class { /* in case of shared mode */
	type leafref {
	path '/model:owning-class/model:grouping-class/model:name';
	require-instance true; /* not necessary since true is default */
	}
	description "none";
	}
	description "none";
	}

 © Open Networking Foundation

```
/*******************************************
* grouping statements for object references
*******************************************/

grouping owning-class-ref {
   leaf owning-class-name {
      type leafref {
         path 'model:owning-class/model:name';
      }
   }
}

grouping another-grouping-class-ref {
   uses owning-class-ref;
   leaf another-grouping-class-name {
      type leafref {
         path 'model:another-grouping-
class/model:name';
      }
   }
}

grouping grouping-class-ref {
   uses owning-class-ref;
   leaf grouping-class-name {
      type leafref {
         path 'model:grouping-class/model:name';
      }
   }
}

grouping shared-part-class-ref {
   uses owning-class-ref;
   leaf shared-part-class-name {
      type leafref {
         path 'model:shared-part-class/model:name';
      }
   }
}
```

 © Open Networking Foundation

```
/****************************************
* grouping statements for object classes
****************************************/

grouping owning-class {
   list shared-part-class {
      key 'owning-class-name shared-part-class-name';
/*    uses shared-part-class-ref; -necessary?*/
      uses shared-part-class;
   }
   list grouping-class {
      key 'owning-class-name grouping-class-name';
/*    uses grouping-class-ref; -necessary?*/
      uses grouping-class;
   }
   list another-grouping-class {
      key 'owning-class-name another-grouping-class-name';
/*    uses another-grouping-class-ref; -necessary?*/
      uses another-grouping-class;
   }
}

grouping grouping-class {
   list shared-part-class {
      uses shared-part-class-ref;
      key 'owning-class-name shared-part-class-name';
   }
/* leaf-list shared-part-class {
      type leafref {
      path '/model:owning-class/model:shared-part-class/model:name';
      }
   } previous definition*/
   leaf name {
      type string;
   }
}

grouping another-grouping-class {
   list shared-part-class {
      uses shared-part-class-ref;
      key 'owning-class-name shared-part-class-name';
   }
/* leaf-list shared-part-class {
      type leafref {
         path '/model:owning-class/model:shared-part-class/model:name';
      }
   } previous definition*/
   list another-grouping-class {
      uses another-grouping-class-ref;
      key 'owning-class-name another-grouping-class-name another-grouping-class-name';
   }
/* leaf-list another-grouping-class {
      type leafref {
         path '/model:owning-class/model:another-grouping-class/
                lifecycle-aggregate-model:name';
      }
   } previous definition*/
   leaf name {
      type string;
   }
}

grouping shared-part-class {
   leaf name {
      type string;
   }
}
```

 © Open Networking Foundation

The following table summarizes the association mappings.

Table 5.16: Association Mapping Summary

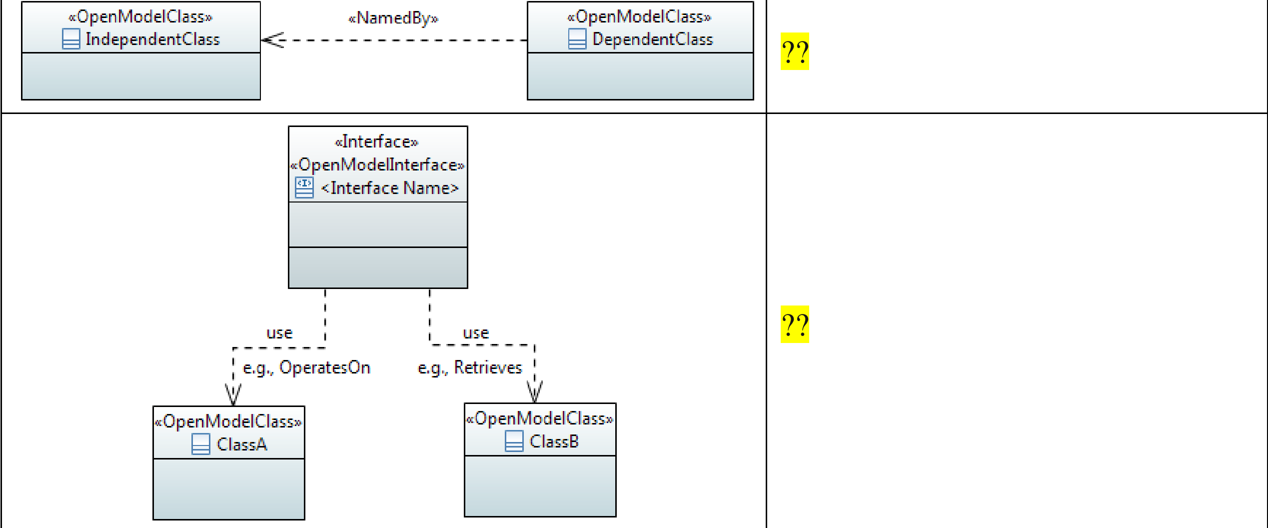
		UML		
		containment	association	inheritance
YANG	nesting	√		
	grouping			√ abstract superclasses
	augment			√ concrete superclasses
	leafref		√	

5.6.2 Mapping of Dependencies

Three different kinds of dependency scenarios need to be mapped:

1. Dependency relationship annotated by the «NamedBy» stereotype
2. Usage dependency relationship between an Interface and the object class the Interface is working on (along with the relationship name)
3. Abstraction dependency relationship annotated by the «Specify» stereotype

Table 5.17: Dependency Mapping Examples



 © Open Networking Foundation

«OpenModelClass»
Entity
+ exampleAttr1: Integer [1]
+ exampleAttr2: Boolean [1]
Entity Model

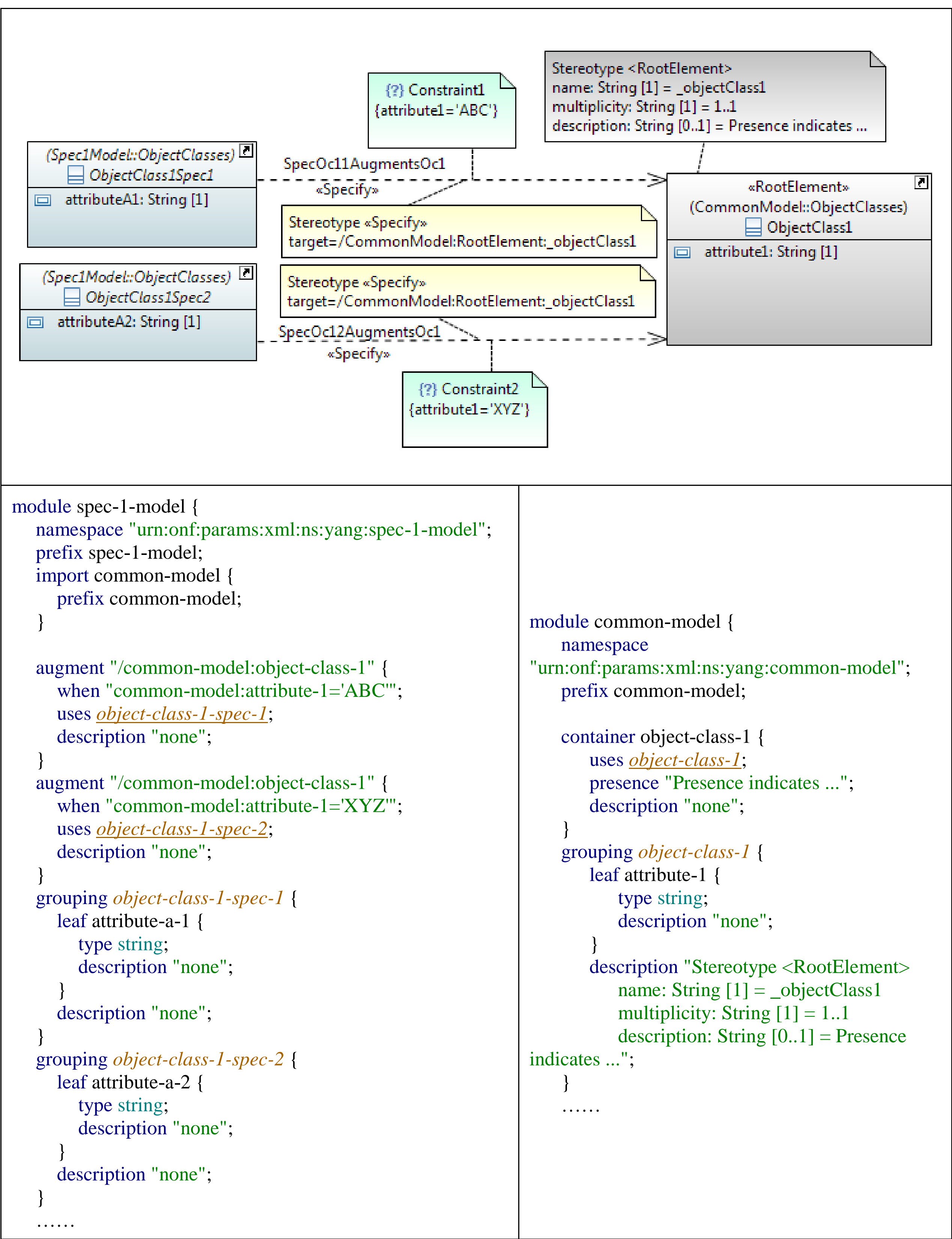
«Specify»

Stereotype «Specify»
target=/EntityModule:_entity

«OpenModelClass»
Specification
+ exampleAttr3: Integer [1]
+ exampleAttr4: Boolean [1]
Spec Model

```
module entity-model {
    …
    grouping entity {
        leaf example-attr-1 {
            type int64;
            …
        }
        leaf example-attr-2 {
            type boolean;
            …
        }
        …
    }
    list entity {
        key 'example-attr-1';
        uses entity;
        …
    }
}

module spec-model {
    …
    grouping specification {
        leaf example-attr-3 {
            type int64;
            …
        }
        leaf example-attr-4 {
            type boolean;
            …
        }
        …
    }

    augment "/entity-module:entity" {
        uses specification;
        …
    }
}
```

 © Open Networking Foundation




```
module spec-1-model {
   namespace "urn:onf:params:xml:ns:yang:spec-1-model";
   prefix spec-1-model;
   import common-model {
      prefix common-model;
   }

   augment "/common-model:object-class-1" {
      when "common-model:attribute-1='ABC'";
      uses object-class-1-spec-1;
      description "none";
   }
   augment "/common-model:object-class-1" {
      when "common-model:attribute-1='XYZ'";
      uses object-class-1-spec-2;
      description "none";
   }
   grouping object-class-1-spec-1 {
      leaf attribute-a-1 {
         type string;
         description "none";
      }
      description "none";
   }
   grouping object-class-1-spec-2 {
      leaf attribute-a-2 {
         type string;
         description "none";
      }
      description "none";
   }
   ……
```

```
module common-model {
    namespace
"urn:onf:params:xml:ns:yang:common-model";
    prefix common-model;

    container object-class-1 {
        uses object-class-1;
        presence "Presence indicates ...";
        description "none";
    }
    grouping object-class-1 {
        leaf attribute-1 {
            type string;
            description "none";
        }
        description "Stereotype <RootElement>
            name: String [1] = _objectClass1
            multiplicity: String [1] = 1..1
            description: String [0..1] = Presence
indicates ...";
    }
    ……
```

5.7 Mapping of Interfaces (grouping of operations)

Table 5.18: UML Interface Mapping

UML Interface → Submodule		
UML Artifact	YANG Artifact	Comments
documentation "Applied comments" (carried in XMI as "ownedComment")	"description" statement	Multiple "applied comments" defined in UML, need to be collapsed into a single "description" substatement.
abstract	"grouping" statement	
OpenModel_Profile::«Reference»	"reference" statement	
OpenModel_Profile::«Example»	Ignore Example elements and all composed parts	
OpenModel_Profile::lifecycleState	"status" substatement or "description" substatement	See section 5.12.
OpenModelInterface::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.
OpenModelInterface::condition		

5.8 Mapping of Operations

 © Open Networking Foundation

Table 5.19: 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.
pre-condition	“extension” substatement→ ompExt: pre-condition	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: post-condition	See extensions YANG module in section 8.2.
input parameter	“input” substatement	

 © Open Networking Foundation

<table>
<tr><th colspan="3">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.)</th></tr>
<tr><th>UML Artifact</th><th>YANG Artifact</th><th>Comments</th></tr>
<tr><td>output parameter</td><td>"output" substatement</td><td></td></tr>
<tr><td>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</td><td>"extension" substatement→ ompExt:operation-exceptions
<table><tr><td>error-tag</td><td>error-app-tag</td></tr><tr><td>operation-failed</td><td>too-many-elements
too-few-elements
must-violation</td></tr><tr><td>data-missing</td><td>instance-required
missing-choice</td></tr><tr><td>bad-attribute</td><td>missing-instance</td></tr></table></td><td>See extensions YANG module in section 8.2.

RE Operations Exceptions.msg</td></tr>
<tr><td>OpenModelOperation:: isOperationIdempotent (obsolete)</td><td>"extension" substatement→ ompExt:is-operation-idempotent</td><td>See extensions YANG module in section 8.2.</td></tr>
<tr><td>OpenModelOperation:: isAtomic (obsolete)</td><td>"extension" substatement→ ompExt:is-atomic</td><td>See extensions YANG module in section 8.2</td></tr>
</table>

 © Open Networking Foundation

<table>
<tr><td colspan="3">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.)</td></tr>
<tr><th>UML Artifact</th><th>YANG Artifact</th><th>Comments</th></tr>
<tr><td>OpenModel_Profile::«Reference»</td><td>“reference” substatement</td><td></td></tr>
<tr><td>OpenModel_Profile::«Example»</td><td>Ignore Example elements and all composed parts</td><td></td></tr>
<tr><td>OpenModel_Profile::lifecycleState</td><td>“status” substatement
or
“description” substatement</td><td>See section 5.12.</td></tr>
<tr><td>OpenModelOperation::support</td><td rowspan="2">“if-feature” substatement</td><td rowspan="2">Support and condition belong together. If the “support” is conditional, then the “condition” explains the conditions under which the class has to be supported.</td></tr>
<tr><td>OpenModelOperation::condition</td></tr>
</table>

 © Open Networking Foundation

Table 5.20: Interface/Operation Mapping Example

OpenModelUmlYangMappingGuidelines.di

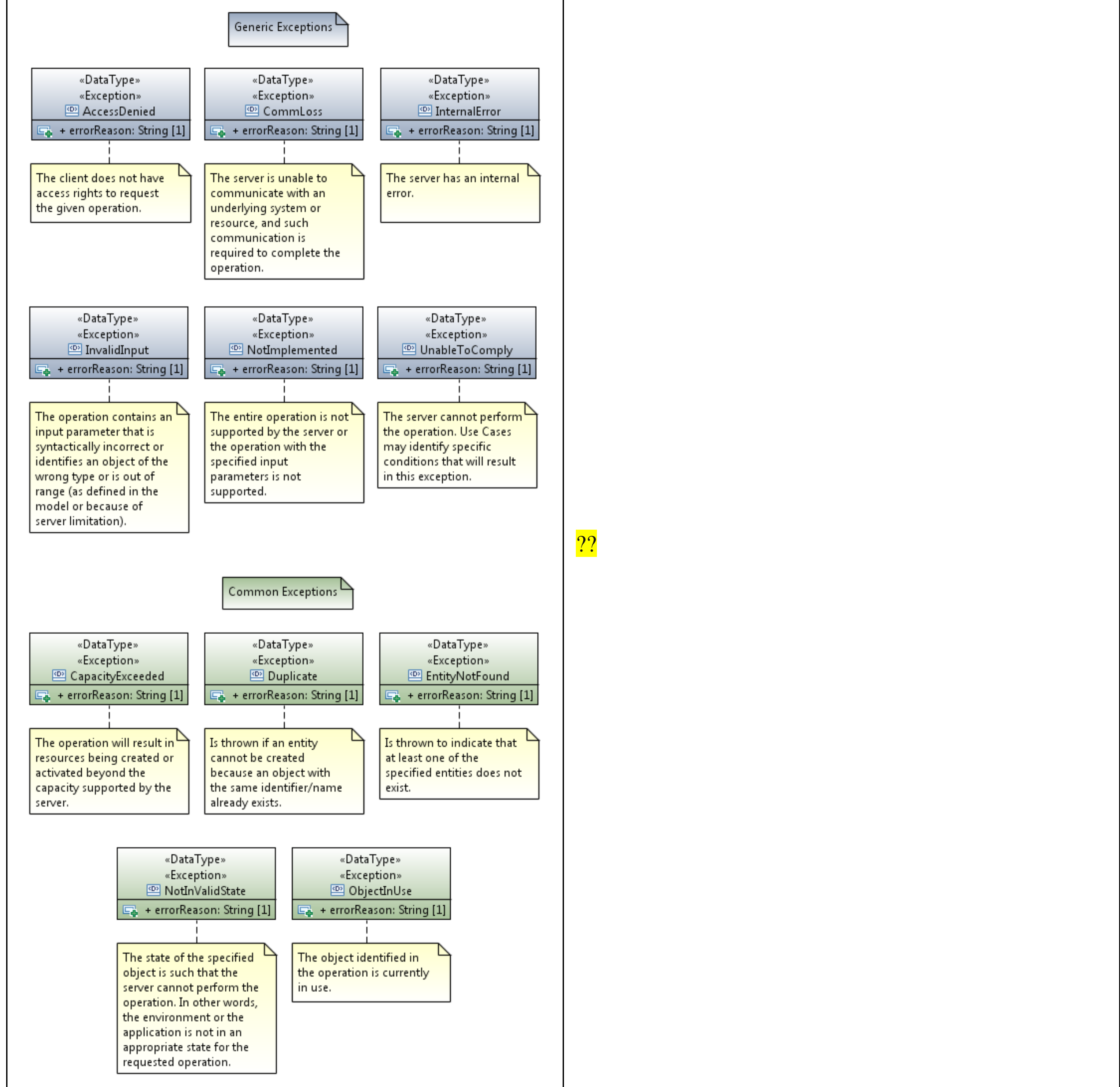
«Interface»
«OpenModelInterface»
InterfaceA

+ operation1()
+ operation2()

```
module open-model-uml-yang-mapping-guidelines {
      <yang-version statement>
      <namespace statement>
      prefix "mapg";
      <import statements>
      include “interface-a” {
            …;
      }
      <organization statement>
      <contact statement>
      <description statement>
      <reference statement>
      …
}

submodule interface-a {
      <yang-version statement>
      belongs-to " open-model-uml-yang-mapping-guidelines" {
            prefix "mapg";
      }
      <import statements>
      <organization statement>
      <contact statement>
      <description statement>
      <reference statement>
      <revision statements>
      …
      rpc operation-1 {
            …
      }
      rpc operation-2 {
            …
      }
}
```

 © Open Networking Foundation

Table 5.21: Operation Exception Mapping Example



??

5.9 Mapping of Operation Parameters

 © Open Networking Foundation

Table 5.22: Parameter Mapping

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		
OpenModelParameter::valueRange		
OpenInterfaceModel_Profile:: 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.
OpenModel_Profile::«Reference»	“reference” substatement of the individual parameters (container, leaf, leaf-list, list, uses)	
OpenModel_Profile::«Example»	Ignore Example elements and all composed parts	

 © Open Networking Foundation

Operation Parameters → "input" substatement or "output" substatement		
UML Artifact	YANG Artifact	Comments
OpenModel_Profile::lifecycleState	"status" substatement or "description" substatement of the individual parameters (container, leaf, leaf-list, list, uses)	See section 5.12.
OpenModelParameter::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.
OpenModelParameter::condition		
XOR: See section 6.3	"choice" substatement	
error notification?	"must" substatement	
complex parameter	"uses" substatement	

 © Open Networking Foundation

Table 5.23: Interface/Operation/Parameter Mapping Example

UML	YANG

«Interface»
InterfaceA
+ operation1(in parameter1: String [1], inout parameter2: Boolean [1], out parameter3: Integer [3..*])
+ operation2(out parameter4: String [*])
	submodule interface-a {
	…
	rpc operation-1 {
	…
	input {
	leaf parameter-1 {
	type string;
	mandatory true;
	}
	leaf parameter-2 {
	type boolean;
	mandatory true;
	}
	}
	output {
	leaf parameter-2 {
	type boolean;
	mandatory true;
	}
	leaf-list parameter-3 {
	type int64;
	min-elements 3;
	}
	}
	}
	rpc operation2 {
	…
	output {
	leaf-list parameter-4 {
	type string;
	}
	}
	}
	}

5.10 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.

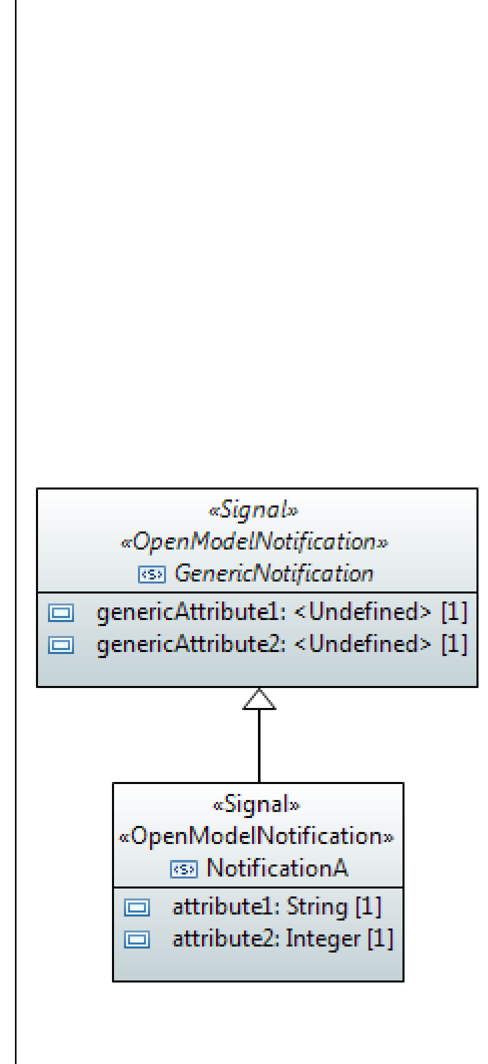
 © Open Networking Foundation

Table 5.24: 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.
OpenModel_Profile::«Reference»	“reference” substatement	
OpenModel_Profile::«Example»	Ignore Example elements and all composed parts	
OpenModel_Profile::lifecycleState	“status” substatement or “description” substatement	See section 5.12.
OpenModelNotification:: triggerConditionList	Not mapped	
OpenModelNotification::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.
OpenModelNotification::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	

 © Open Networking Foundation

Table 5.25: Notification Mapping Example




```
grouping generic-notification {
    …
    leaf generic-attribute-1 {
        …
        mandatory true;
    }
    leaf-list generic-attribute-2 {
        …
        mandatory true;
    }
}

grouping notification-a {
    …
    leaf attribute-1 {
        type string;
        …
    }
    leaf attribute-2 {
        type integer;
        …
    }
}

notification notification-a {
    …
    uses generic-notification;
    uses notification-a;
}

notification notification-a {
    …
    uses generic-notification;
    uses notification-a;
}
```

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

 © Open Networking Foundation

5.11 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 5.26: UML Package to YANG Heading Mapping

UML Package	YANG Heading
Associations	grouping statements for object references augment statements
TypeDefinitions	identity statements typedef statements grouping statements for complex data types
ObjectClasses	grouping statements for object-classes
Interfaces	rpc statements
Notifications	notification statements

5.12 Mapping of Lifecycle

 © Open Networking Foundation

Table 5.27: 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.

5.13 Mapping Issues

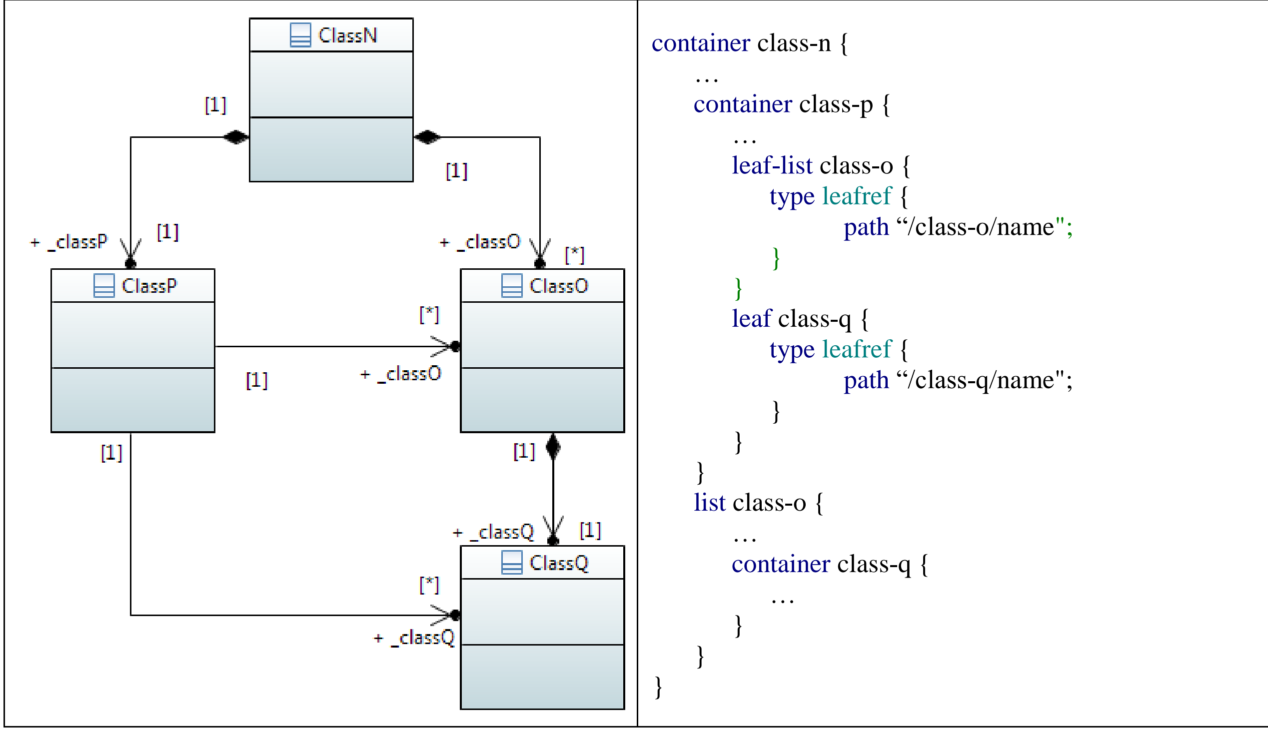
5.13.1 YANG 1.0 or YANG 1.1?

YANG 1.0 is approved and defined in RFC 6020 [1].
YANG 1.1 is approved and defined in RFC 7950 [11]. The enhancements are listed in section “Summary of Changes from RFC 6020”.

5.13.2 Combination of different Associations?

 © Open Networking Foundation

Table 5.28: Combination of Associations Mapping Examples




```
container class-n {
    …
    container class-p {
        …
        leaf-list class-o {
            type leafref {
                    path "/class-o/name";
            }
        }
        leaf class-q {
            type leafref {
                    path "/class-q/name";
            }
        }
    }
    list class-o {
        …
        container class-q {
            …
        }
    }
}
```

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.,

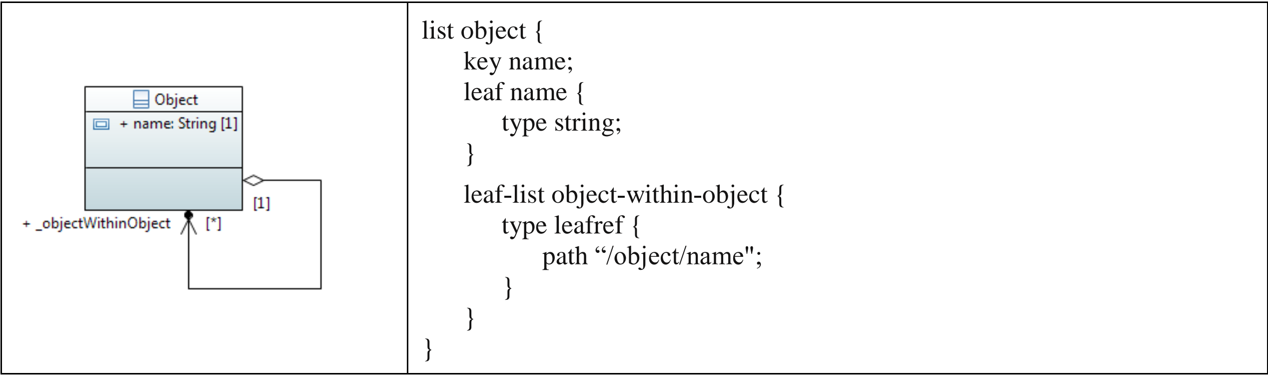
- Reference based approach - have a flat list of objects, where the objects are linked into a hierarchy using references. An example of a two-way navigable approach is in RFC 7223.
- Assume some specific number of "recursions"; i.e., specify some default number of recursion levels, and define a configurable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

 © Open Networking Foundation

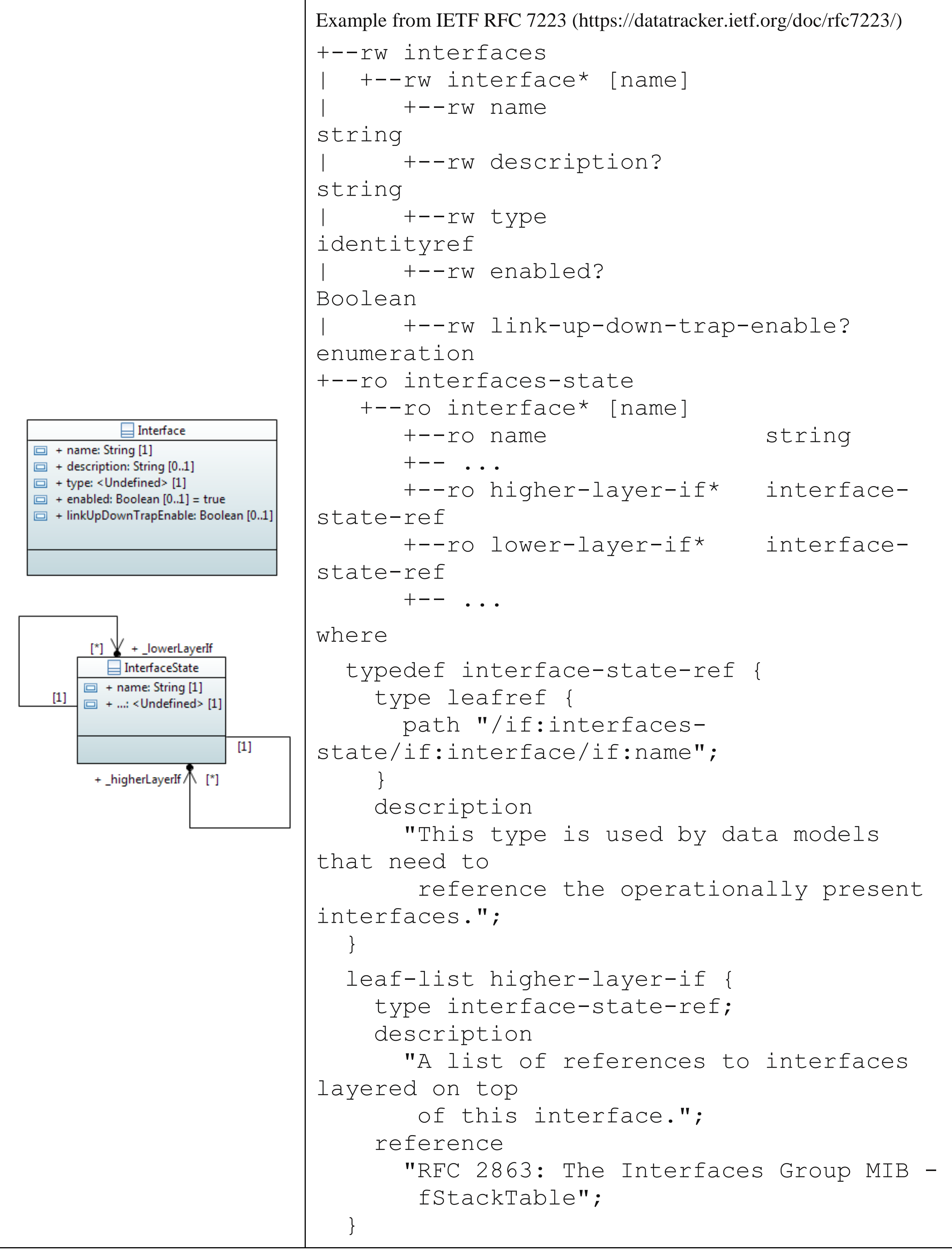
6.1.1 Reference Based Approach

Table 6.1: Recursion Mapping Examples




```
list object {
    key name;
    leaf name {
        type string;
    }
    leaf-list object-within-object {
        type leafref {
            path "/object/name";
        }
    }
}
```

 © Open Networking Foundation



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";
  }
```

 © Open Networking Foundation

	``
	  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).

 © Open Networking Foundation

Table 6.2: Mapping of Conditional Packages

ClassF_Pac
+ attribute3
+ attribute4
<condition for ClassF_Pac attributes>
+ _classF_Pac
«Cond»
0..1
ClassE
{partOfObjectKey=1 } + objectIdentifier: String
+ attribute2
1
1
«Cond»
0..1
ClassG_Pac
+ attribute5
+ attribute6
<condition for ClassG_Pac attributes>
+ _classG_Pac

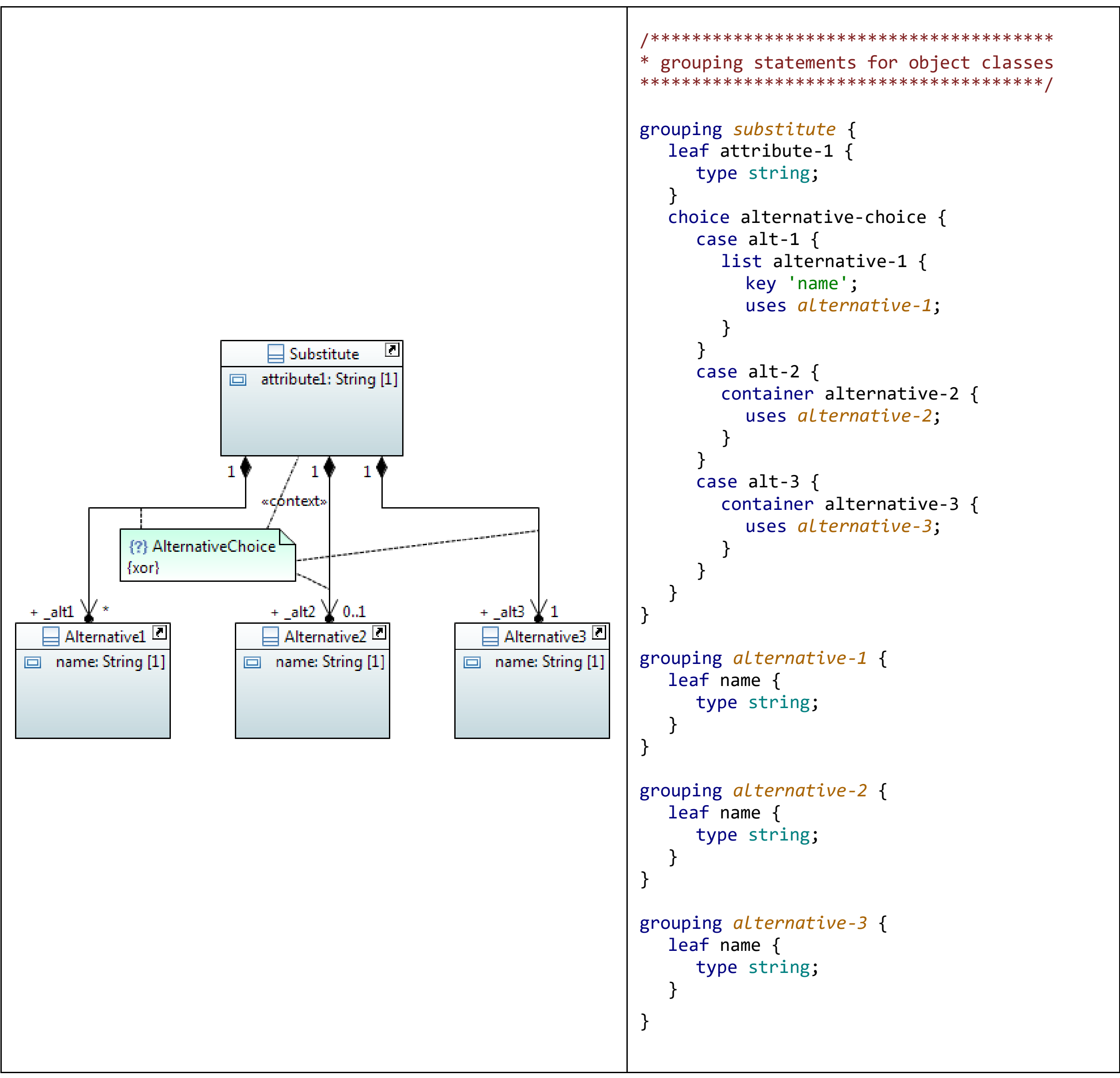
```
grouping class-e {
    …
    leaf object-identifier {
        type string;
    }
    leaf attribute-2 {
        …
    }
grouping class-f-pac {
    …
    leaf attribute-3 {
        …
    }
    leaf attribute-4 {
        …
    }
}
grouping class-g-pac {
    …
    leaf attribute-5 {
        …
    }
    leaf attribute-6 {
        …
    }
}
list class-e {
    key "object-identifier";
    uses class-e;
    …
    container class-f-pac {
        presence " <condition for ClassF_Pac
attributes>";
        uses class-f-pac;
        …
    }
    container class-g-pac {
        presence " <condition for ClassG_Pac
attributes>";
        uses class-g-pac;
        …
    }
}
```

 © Open Networking Foundation

6.3 {xor} Constraint

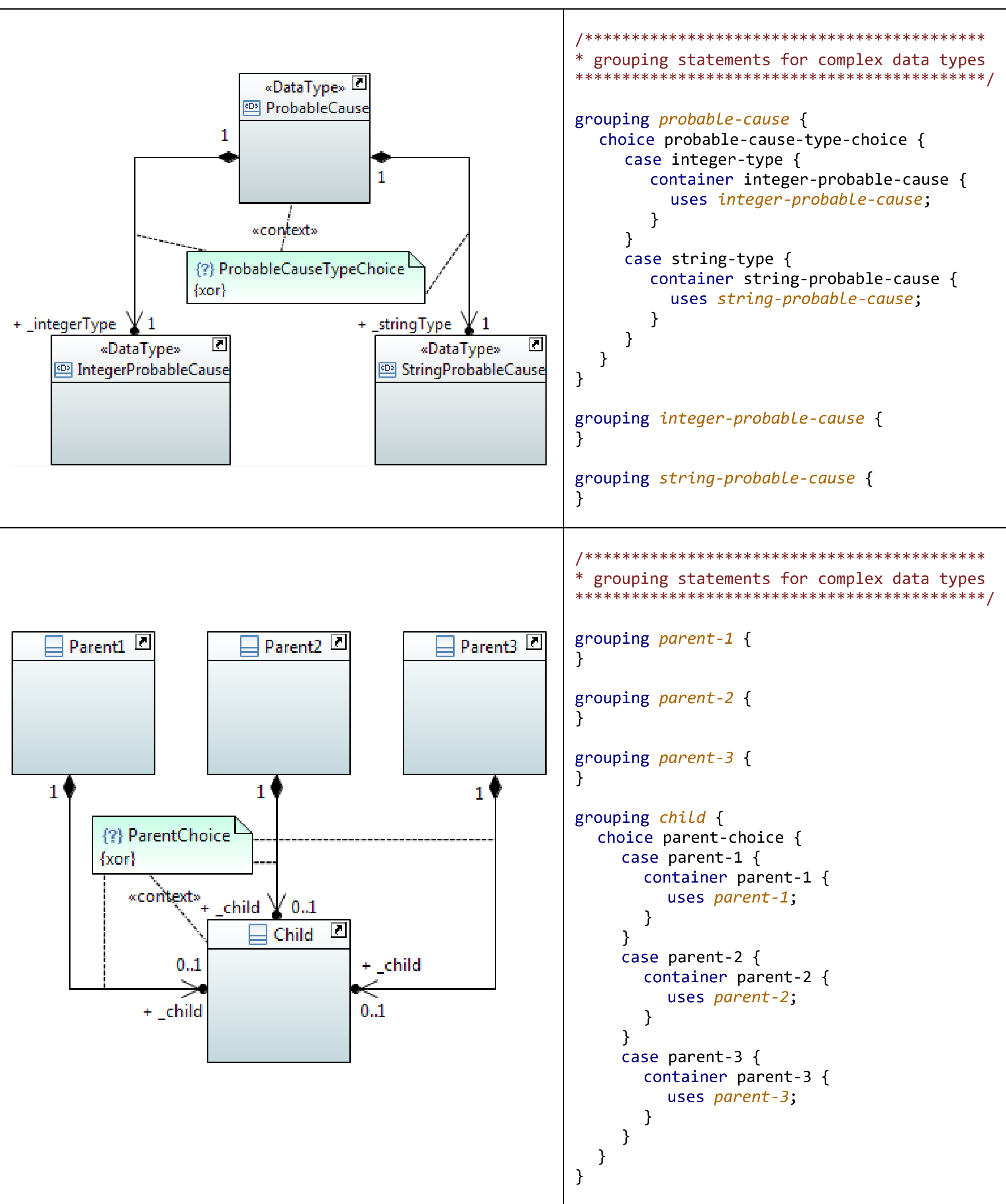
Associations related by the {xor} constraint are mapped to the "choice" statement.

Table 6.3: {xor} Constraint Mapping Examples




```
/****************************************
* grouping statements for object classes
****************************************/

grouping substitute {
   leaf attribute-1 {
      type string;
   }
   choice alternative-choice {
      case alt-1 {
         list alternative-1 {
            key 'name';
            uses alternative-1;
         }
      }
      case alt-2 {
         container alternative-2 {
            uses alternative-2;
         }
      }
      case alt-3 {
         container alternative-3 {
            uses alternative-3;
         }
      }
   }
}

grouping alternative-1 {
   leaf name {
      type string;
   }
}

grouping alternative-2 {
   leaf name {
      type string;
   }
}

grouping alternative-3 {
   leaf name {
      type string;
   }
}
```

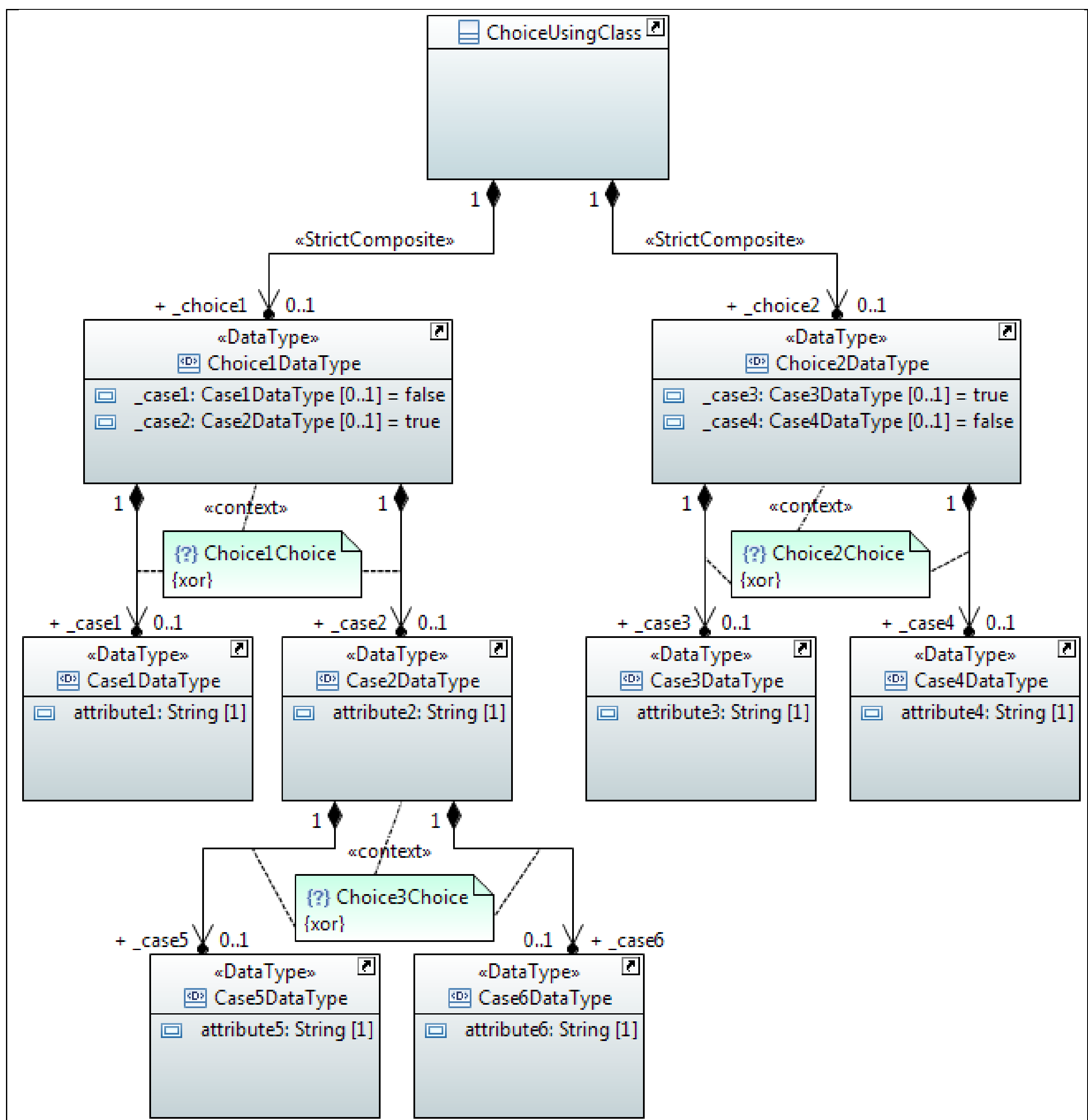
 © Open Networking Foundation




```
/*********************************************
* grouping statements for complex data types
*********************************************/

grouping probable-cause {
   choice probable-cause-type-choice {
      case integer-type {
         container integer-probable-cause {
            uses integer-probable-cause;
         }
      }
      case string-type {
         container string-probable-cause {
            uses string-probable-cause;
         }
      }
   }
}

grouping integer-probable-cause {
}

grouping string-probable-cause {
}
```

```
/*********************************************
* grouping statements for complex data types
*********************************************/

grouping parent-1 {
}

grouping parent-2 {
}

grouping parent-3 {
}

grouping child {
   choice parent-choice {
      case parent-1 {
         container parent-1 {
            uses parent-1;
         }
      }
      case parent-2 {
         container parent-2 {
            uses parent-2;
         }
      }
      case parent-3 {
         container parent-3 {
            uses parent-3;
         }
      }
   }
}
```

 © Open Networking Foundation



 © Open Networking Foundation

```
/*******************************************
* grouping statements for complex data types
*******************************************/

grouping choice-1-data-type {
  choice choice-1-choice {
    default case-2;
    case case-1 {
      uses case-1-data-type;
    }
    case case-2 {
      uses case-2-data-type;
    }
  }
}

grouping choice-2-data-type {
  choice choice-2-choice {
    default case-3;
    case case-3 {
      uses case-3-data-type;
    }
    case case-4 {
      uses case-4-data-type;
    }
  }
}

grouping case-1-data-type {
  leaf attribute-1 {
    type string;
  }
}

grouping case-2-data-type {
  leaf attribute-2 {
    type string;
  }
  choice choice-3-choice {
    case case-5 {
      uses case-5-data-type;
    }
    case case-6 {
      uses case-6-data-type;
    }
  }
}
```

```
grouping case-3-data-type {
  leaf attribute-3 {
    type string;
  }
}

grouping case-4-data-type {
  leaf attribute-4 {
    type string;
  }
}

grouping case-5-data-type {
  leaf attribute-5 {
    type string;
  }
}

grouping case-6-data-type {
  leaf attribute-6 {
    type string;
  }
}

/**************************************
* grouping statements for object classes
**************************************/

grouping choice-using-class {
  container choice-1 {
    uses choice-1-data-type;
  }
  container choice-2 {
    uses choice-2-data-type;
  }
}
```

6.4 «Choice» Stereotype (obsolete)

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.

 © Open Networking Foundation

Table 6.4: «Choice» Stereotype Mapping Examples

 © Open Networking Foundation

«OpenModelClass, Choice»
SubstituteObjectClass
1 1 1
1 + _alternative1
1 + _alternative2
1 + _alternative3
«OpenModelClass»
Alternative1ObjectClass
«OpenModelClass»
Alternative2ObjectClass
«OpenModelClass»
Alternative3ObjectClass

```
grouping substitute-object-class {
    …
    }
grouping alternative-1-object-class {
    …
}
grouping alternative-2-object-class {
    …
}
grouping alternative-3-object-class {
    …
}
container alternative-1-object-class {
    uses alternative-1-object-class;
    …
}
container alternative-2-object-class {
    uses alternative-2-object-class;
    …
}
container alternative-3-object-class {
    uses alternative-3-object-class;
    …
}
list substitute-object-class {
    key …;
    …
    choice _alternative {
        case alternative-1-object-class {
            leaf alternative-1-object-class
{
                type leafref {
                    path '/alternative-1-
object-class';
                }
            }
        }
        case alternative-2-object-class {
            leaf alternative-2-object-class
{
                type leafref {
                    path '/alternative-2-
object-class';
                }
```

 © Open Networking Foundation

<table>
<tr>
<td></td>
<td>}
}
case alternative-3-object-class {
leaf alternative-3-object-class
{
type leafref {
path '/alternative-3-
object-class;
}
}
}
}
}</td>
</tr>
<tr>
<td>«DataType»
«choice»
ProbableCause
[1] [1]
[1] [1]
«DataType»
IntegerProbableCause
+ probableCause: Integer [1]
«DataType»
StringProbablecause
+ probablecause: String [1]</td>
<td>grouping integer-probable-cause {
…
leaf probable-cause {
type int64;
…
}
}
grouping string-probable-cause {
…
leaf probable-cause {
type string;
…
}
}
grouping probable-cause {
…
choice probable-cause {
case integer-probable-cause {
container integer-probable-
cause {
uses integer-probable-
cause;
…
}
}
case string-probable-cause {
container string-probable-
cause {
uses string-probable-cause;
…
}
}</td>
</tr>
</table>

 © Open Networking Foundation

```
    }
}
```

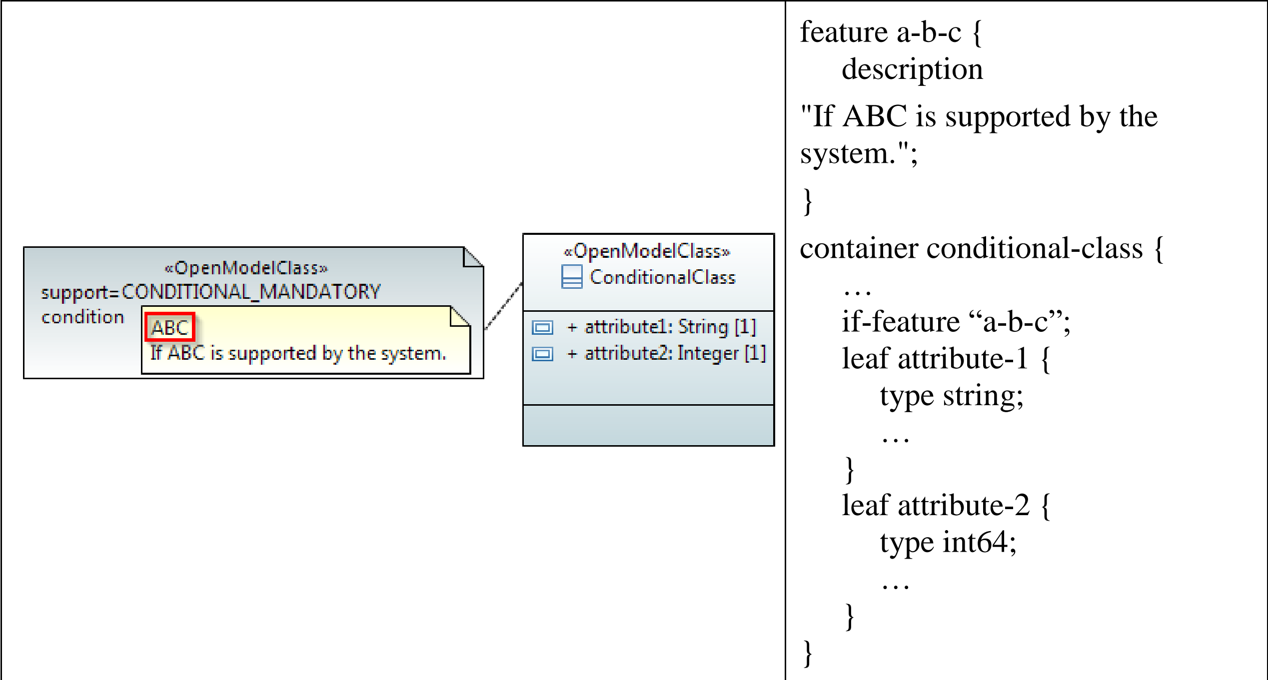
6.5 Mapping of UML Support and Condition Properties

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. I.e., all condition strings which have the same first line must also have the same condition text. It is possible to add the same condition text to more than one artifact. 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




```
feature a-b-c {
    description
"If ABC is supported by the system.";
}
container conditional-class {
    …
    if-feature “a-b-c”;
    leaf attribute-1 {
        type string;
        …
    }
    leaf attribute-2 {
        type int64;
        …
    }
}
```

 © Open Networking Foundation

<table>
<tr>
<td>(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]</td>
<td>feature x-y-z {
    description
"If XYZ is supported by …";
}
container conditional-attribute-class {
    …
    leaf attribute-3 {
        type string;
        …
    }
    leaf attribute-4 {
        type int64;
        if-feature “x-y-z”;
        …
    }
    leaf attribute-5 {
        type string;
        if-feature “not x-y-z”;
        …
    }
}</td>
</tr>
</table>

 © Open Networking Foundation

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.

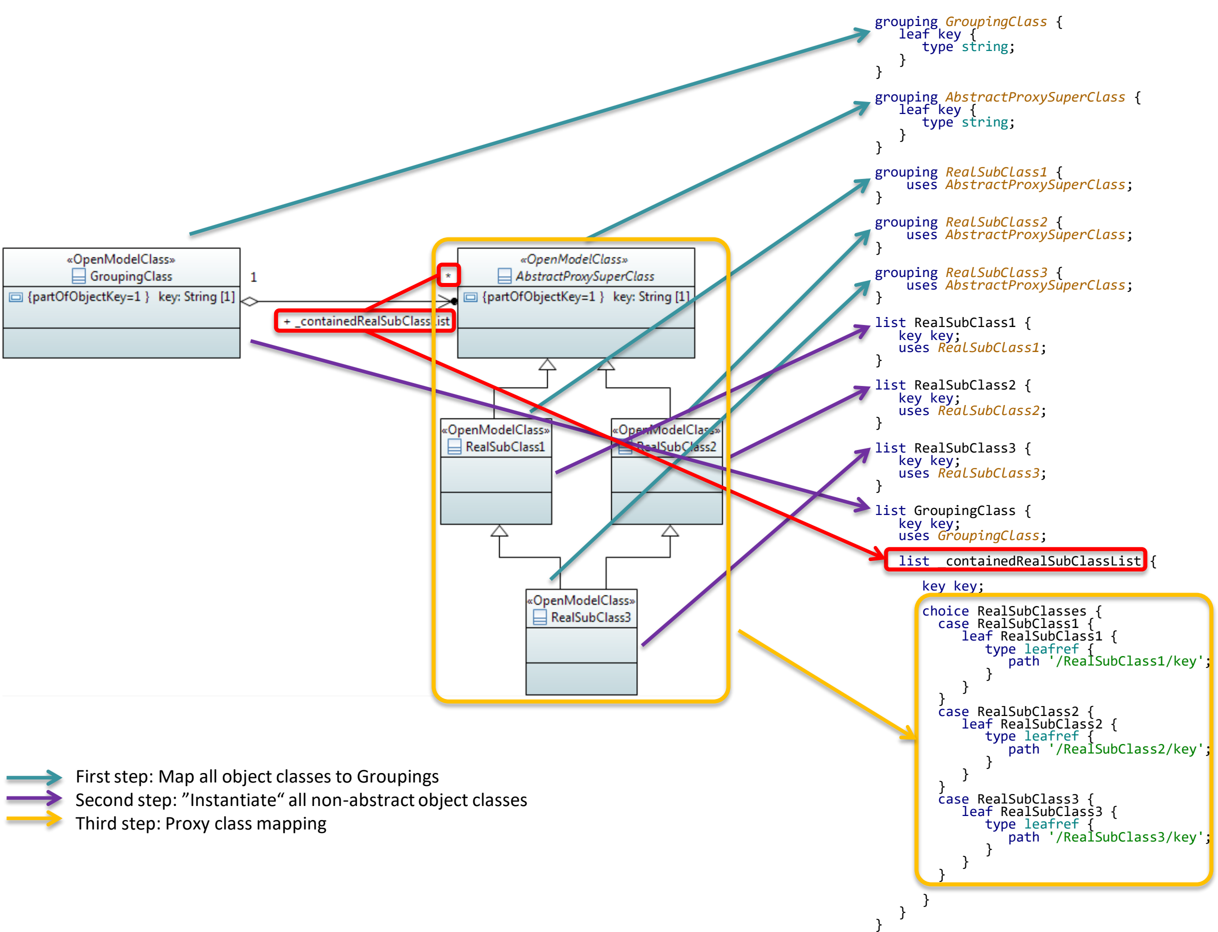


Figure 6.1: Example: Proxy Class Mapping

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.

 © Open Networking Foundation

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]

{?} Constraint1 {xor}

«OpenModelClass» ClassU
{partOfObjectKey=1} attribute3: <Undefined> [1]

«OpenModelClass» ClassV
{partOfObjectKey=1} attribute4: <Undefined> [1]

«OpenModelClass» ClassW
{partOfObjectKey=1} attribute5: <Undefined> [1]

«OpenModelClass» ClassX
{partOfObjectKey=0} attribute6: <Undefined> [1]

«OpenModelClass» ClassY
{partOfObjectKey=1} attribute7: <Undefined> [1]

```
/****************************
* Groupings
****************************/
grouping class-s {
    leaf attribute1 {
        … }
}

grouping class-t {
    leaf attribute2 {
        … }
}

grouping class-u {
    leaf attribute3 {
        … }
}

grouping class-v {
    leaf attribute4 {
        … }
}

grouping class-w {
    leaf attribute5 {
        … }
}

grouping class-x {
    leaf attribute6 {
        … }
}

grouping class-y {
    leaf attribute7 {
        … }
}
```

```
/****************************
* Main Containers
****************************/
list class-s {
    key attribute1;
    uses class-s;
    …
    list class-u {
        key attribute3;
        uses class-u;
        …
        list class-v {
            key attribute4;
            uses class-v;
            uses class-x
            …
        }
    }
}
list class-s {
    key attribute1;
    uses class-s;
    …
    list class-u {
        key attribute3;
        uses class-u;
        …
        list class-w {
            key attribute5;
            uses class-w;
            …
            list class-y {
                key attribute7;
                uses class-y;
                …
            }
        }
    }
}
```

```
list class-t {
    key attribute2;
    uses class-t;
    …
    list class-u {
        key attribute3;
        uses class-u;
        …
        list class-v {
            key attribute4;
            uses class-v;
            uses class-x
            …
        }
    }
}
list class-t {
    key attribute2;
    uses class-t;
    …
    list class-u {
        key attribute3;
        uses class-u;
        …
        list class-w {
            key attribute5;
            uses class-w;
            …
            list class-y {
                key attribute7;
                uses class-y;
                …
            }
        }
    }
}
```


© Open Networking Foundation

7 Generic UML Model and specific YANG Configuration Information

The UML model provides already a lot of generic information which is also mapped to YANG. If necessary, the user can overwrite some of the generic UML model information before it is mapped to YANG and can provide a few mapping instructions to the tool in a config file.

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:

Input		Output
UML	Config file	YANG Module Header Information
UML model name		module name // same as the UML model name; transformed into the YANG naming scheme; e.g., “UmlYangSimpleTestModel.uml” is mapped to “uml-yang-simple-test-model.yang”
OpenModelStatement::revision:Revision::date <yyyy>-<mm>-<dd> See revision::date below	"revision"::"date":"<yyyy>-<mm>-<dd>" See revision::date below	file name // consists of the module name and the date: "<module-name>@<yyyy>-<mm>-<dd>.yang";
OpenModelStatement::namespace urn:<sdo>:<project>	"namespace":"urn:<sdo>:<project>:yang:", e.g., urn:onf:otcc:{tapi|wt|wr|…}:yang:<module name> e.g., urn:itu:t:rec:g.8052.1:yang:{ITU-T-YangModuleIdentifier}	namespace "urn:<sdo>:<project>:yang:<module name>", // string with a unique namespace URN value
	"prefix":{ "<UML model name1>":"<prefix1>”, "<UML model name2>":"<prefix2>”, "<UML model name3>":"<prefix3>”, "<UML model name4>":"<prefix4>”, … },	prefix // try to pick a unique prefix; should not have more than 8-10 characters

 © Open Networking Foundation

Input		Output
UML	Config file	YANG Module Header Information
OpenModelStatement::organization	"organization":"<SDO and project/wg name>",(human friendly written) e.g., ONF OTCC (Open Transport Configuration & Control) Project",	organization // identify also the working group if applicable
OpenModelStatement::contact:Contact [1] - projectWeb:String [1] = <https://.../project-name/> - projectEmail:String [1] = <mailto:project-name@...> - editorName:String [0..1] = <project editor> - editorEmail:String [0..1] = <mailto:project-editor@example.com>	"contact":"\n Project Web: <https://.../project-name/>\n Project Email: <mailto:project-name@...>\n Editor: <project editor>\n <mailto:project-editor@example.com>",	contact " Project Web: <https://.../project-name/> Project Email: <mailto:project-name@...> Editor: <project editor> <mailto:project-editor@example.com>"
OpenModelStatement::description [0..1] "This model defines < brief description of the model; 1 line>" OpenModelStatement::copyright [1] OpenModelStatement::licence [1]		description "This module defines <brief description of the module; 1 line> <copyright notice> <license statement>" //Blue text hard coded in the mapping tool
	"reference":"<clearly identify the source (e.g., UML model)>"	reference "<clearly identify the source (e.g., UML model)>"

© Open Networking Foundation

<table>
<tr><th colspan="2">Input</th><th>Output</th></tr>
<tr><th>UML</th><th>Config file</th><th>YANG Module Header Information</th></tr>
<tr><td>OpenModelStatement::revision:Revision [1..*]
- date [1] <yyyy>-<mm>-<dd>
- version [1] <project/project version>
- description [0..1] {<additional specific description>}
- changelog [0..1] <link to a github UML change log>
- additionalChanges [0..1] {<additional manual changes>}
- reference [0..1] <list of referenced documents></td><td>"revision": [{
"date":"<yyyy>-<mm>-<dd>",

"description":"<project/project version>\n {<additional specific description>}\n <link to the github yang change log> {<additional manual changes>}",

"reference":"< list of referenced documents>"
}]</td><td>revision "<revisiondate; <yyyy>-<mm>-<dd>>" {

revision::description
description "<project/project version>
This YANG module is automatically generated by the xmi2yang Mapping Tool, version <version of the tool>.
{<additional specific description>.}
Changes in this revision: <link to the github yang change log> {<additional manual changes>}";

revision::reference "<list of referenced documents>";
}
// what changed in this revision
//Blue text hard coded in the mapping tool</td></tr>
<tr><td>Applied profiles:
OpenModelProfile, v<version>
OpenInterfaceModelProfile, v<version>
ProfileLifecycleProfile, v<version></td><td></td><td></td></tr>
<tr><td></td><td>"withSuffix":true|false,</td><td>Add suffix "-g" to the groupings: No|Yes (default: No)</td></tr>
</table>

7.2 Lifecycle State Treatment

UML elements are annotated by at least one of the following lifecycle states (see also section 5.12):

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

 © Open Networking Foundation

- «Preliminary».

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

 © Open Networking Foundation

8 Mapping Basics

8.1 UML → YANG or XMI → YANG

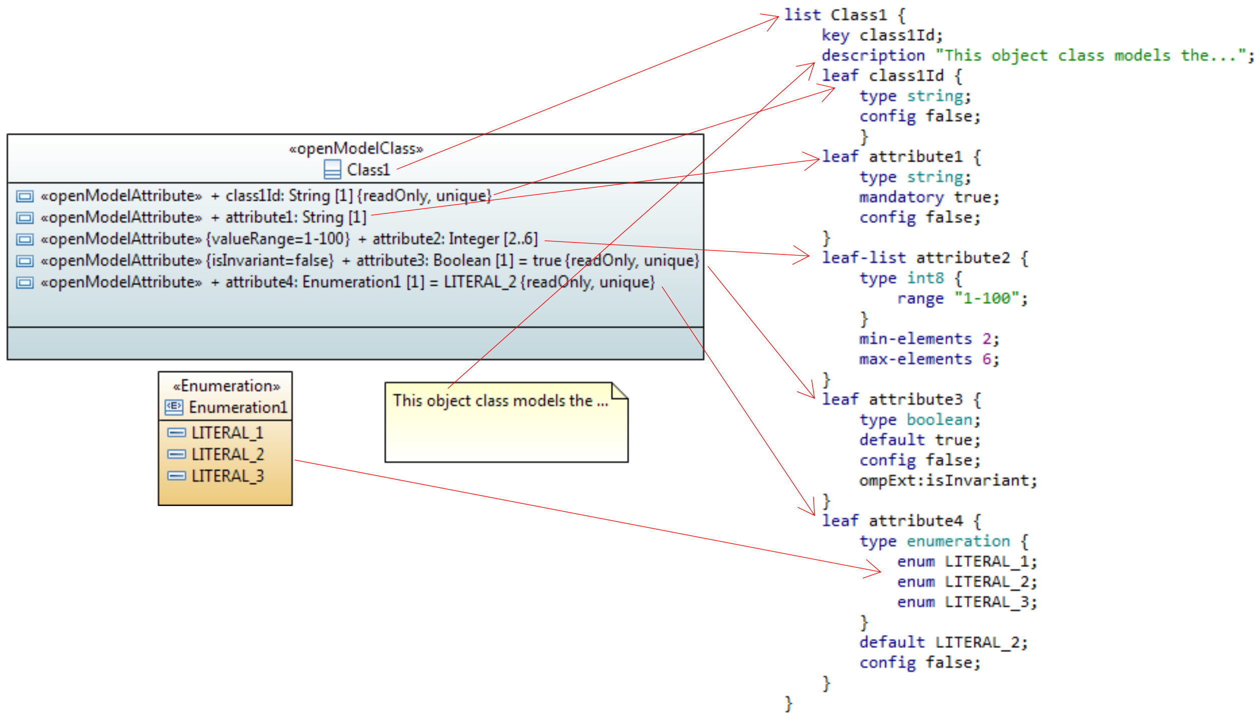


Figure 8.1: Example UML to YANG Mapping

 © Open Networking Foundation

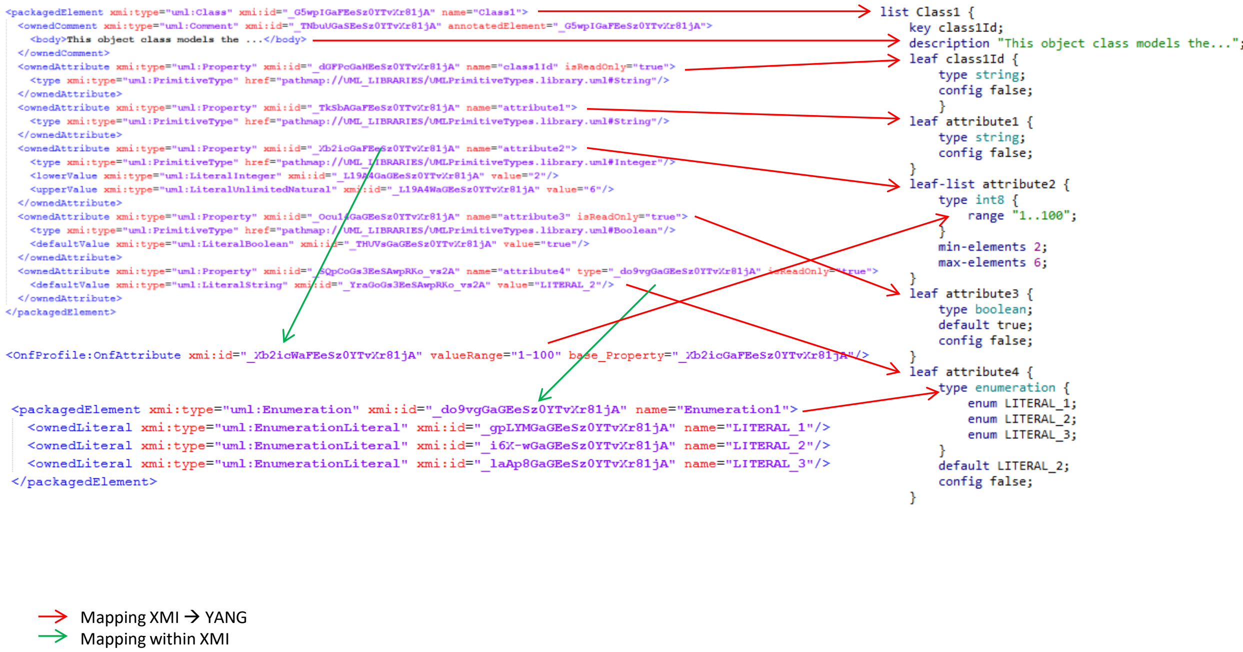


Figure 8.2: Example XMI (Papyrus) to YANG Mapping

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 "iisomi-OpenModelProfileExtensions@2017-04-12.yang"

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

    organization
        "IISOMI (Informal Inter-SDO Open Model Initiative)";

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

    revision 2017-04-12 {
        description "ONF replaced by IISOMI";
        reference "IISOMI 514 UML Modeling Guidelines";
    }

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

 © Open Networking Foundation

```
//  extension statements

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

extension pre-condition {
    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 post-condition {
    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 operation-exceptions {
    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).
```

 © Open Networking Foundation

```
             • 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 is-operation-idempotent {
         description
           "Used with operation definitions to indicate that the operation
            is idempotent.";
     }

}
```

<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/augmentation 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 Requirements for the YANG Module Structure

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

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

```
module <module name> {
// The module header is constructed according to section 7.1
   }
```

 © Open Networking Foundation

```
/**************************************************************
 * augment statements
 **************************************************************/

…

/**************************************************************
 * extension statements
 **************************************************************/

…

/**************************************************************
 * feature statements
 **************************************************************/

…

/**************************************************************
 * identity statements
 **************************************************************/

…

/**************************************************************
 * typedef statements
 **************************************************************/

…

/**************************************************************
 * grouping statements for complex data types
 **************************************************************/

…

/**************************************************************
 * grouping statements for object references
 **************************************************************/

…

/**************************************************************
 * grouping statements for object-classes
 **************************************************************/

…

/**************************************************************
 * data definition statements
 **************************************************************/

…

/**************************************************************
 * rpc statements
```

 © Open Networking Foundation

```
 *****************************************************************/
…
 /****************************************************************
 * notification statements
 *****************************************************************/
…
}
<CODE ENDS>
```

11 Main Changes between Releases

11.1 Summary of main changes between version 1.0 and 1.1

- Adapted to ETSI drafting rules.
- Usage of isLeaf attribute property clarified in section 5.5.4.
- Usage of isLeaf class property added in section 5.3.
- New section 5.6.2 on Mapping of Dependencies added.
- «Specify» abstraction mapping example added in Table 5.4 and Table 5.17.
- Naming Conventions Mapping in Table 5.1 enhanced and naming updated throughout the document.
- Section 5.2 on Generic Mapping Guidelines added.
- Mapping example for RootElement added to Table 5.4.
- New property attribute writeAllowed added to Table 5.5.
- Pointer association mapping examples added in Table 5.15.
- New section 5.2.3 on YANG Workarounds added.
- Literal name style mapping updated in Table 5.1.
- Conditional Specify/Augment mapping example added in Table 5.17.
- Mapping of bit encoding for Enumerations added in Table 5.13.
- «LifecycleAggregate» Relationship Mapping Examples added in Table 5.15.
- Mapping of references added in section 5.6.1 and Table 5.15.
- Mapping of Choice stereotype obsolete.
- Mapping OpenInterfaceModelClass stereotype obsolete.
- Mapping OpenModelOperation::isOperationIdempotent property obsolete.
- Mapping OpenModelOperation::isAtomic property obsolete.
- Mapping OpenInterfaceModelAttribute::attributeValueChangeNotification property obsolete.
- Composite association mapped to list using grouping instead of reference in Table 5.15. “require-instance” substatement = false added to all navigable pointer and shared aggregation associations.

 © Open Networking Foundation

- {xor}-choice mapping updated in section 6.3.
- Bits mapping updated in section 5.5.5.

12 Proposed Addendum 2

Stereotype:Do not generate DS?

Using spanning tree algorithm?

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

 © Open Networking Foundation