

Mapping Guidelines  
UML 🡪 YANG

Draft Version 0.4

Sept. 18, 2015

­ONF Document Type: Technical Recommendation

ONF Document Name: ONF\_Mapping-Guidelines\_UML-YANG\_v0.4

**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

2275 E. Bayshore Road, Suite 103, Palo Alto, CA 94303

[www.opennetworking.org](http://www.opennetworking.org)

©2014 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.

1 Introduction 6

2 References 6

3 Overview 6

3.1 Documentation Overview 6

4 Mapping Guidelines 7

4.1 Mapping of Object Classes 8

4.2 Mapping of Attributes 11

4.3 Mapping of Types 13

4.3.1 Mapping of Primitive Types 14

4.3.2 Mapping of Enumeration Types 14

4.3.3 Mapping of Basic Data Types 16

4.3.4 Mapping of Complex Data Types 18

4.4 Mapping of Associations 18

4.5 Mapping of Interfaces (grouping of operations) 21

4.6 Mapping of Operations 21

4.7 Mapping of Operation Parameters 22

4.8 Mapping of Notifications 24

4.9 Mapping of Lifecycle 25

4.10 Other Mappings 26

4.11 Mapping Issues 26

4.11.1 Using types defined in YANG? 26

4.11.2 YANG 1.0 or YANG 1.1? 27

4.11.3 Mapping of UML Packages? 27

4.11.4 Combination of different Associations? 28

5 Mapping Patterns 29

5.1 UML Recursion 29

5.1.1 Reference Based Approach 29

5.2 UML Conditional Pacs 31

5.3 XOR Relationship 31

5.4 Mapping of UML Support and Condition 31

6 Mapping Basics 32

6.1 UML 🡪 YANG or XMI 🡪 YANG 32

6.2 Open Model Profile YANG Extensions 34

7 Reverse Mapping From YANG to UML 36

8 Proposed Addendum 1: Requirements for the YANG Module header 36

9 Proposed Addendum 2 37

List of Figures

Figure 3.1: ONF Specification Architecture 7

Figure 4.1: Example: Re-engineered ietf-inet-types.yang and ietf-yang-types.yang 26

Figure 6.1: Example UML to YANG Mapping 33

Figure 6.2: Example XMI (Papyrus) to YANG Mapping 33

List of Tables

[Table 4.1: Object Class Mapping (Mappings required by currently used UML artifacts) 9](#_Toc430330754)

[Table 4.2: Object Class Mapping (Mappings for remaining YANG substatements) 9](#_Toc430330755)

[Table 4.3: Object Class Mapping Example 11](#_Toc430330756)

[Table 4.4: Attribute Mapping (Mappings required by currently used UML artifacts) 12](#_Toc430330757)

[Table 4.5: Attribute Mapping (Mappings for remaining YANG substatements) 13](#_Toc430330758)

[Table 4.6: Attribute Type Mapping Example 14](#_Toc430330759)

[Table 4.7: Type Mapping 14](#_Toc430330760)

[Table 4.8: Primitive Type Mapping 15](#_Toc430330761)

[Table 4.9: Enumeration Type Mapping (Mappings required by currently used UML artifacts) 15](#_Toc430330762)

[Table 4.10: Enumeration Type Mapping (Mappings for remaining YANG substatements) 16](#_Toc430330763)

[Table 4.11: Enumeration Type Mapping Example 17](#_Toc430330764)

[Table 4.12: Basic Data Type Mapping (Mappings required by currently used UML artifacts) 17](#_Toc430330765)

[Table 4.13: Basic Data Type Mapping (Mappings for remaining YANG substatements) 18](#_Toc430330766)

[Table 4.14: Basic Data Type Mapping Example 18](#_Toc430330767)

[Table 4.15: Complex Data Type Mapping 19](#_Toc430330768)

[Table 4.16: Complex Data Type Mapping Example 19](#_Toc430330769)

[Table 4.17: Association Mapping 19](#_Toc430330770)

[Table 4.18: Association Mapping Examples 20](#_Toc430330771)

[Table 4.19: Association Mapping Summary 21](#_Toc430330772)

[Table 4.20: UML Interface Mapping 22](#_Toc430330773)

[Table 4.21: Operation Mapping 22](#_Toc430330774)

[Table 4.22: Interface/Operation Mapping Example 23](#_Toc430330775)

[Table 4.23: Parameter Mapping 23](#_Toc430330776)

[Table 4.24: Interface/Operation/Parameter Mapping Example 25](#_Toc430330777)

[Table 4.25: Notification Mapping 25](#_Toc430330778)

[Table 4.26: Notification Mapping Example 26](#_Toc430330779)

[Table 4.27: Lifecycle Mapping 26](#_Toc430330780)

[Table 4.28: Other Mappings 27](#_Toc430330781)

[Table 4.29: IP Address Mapping Examples 28](#_Toc430330782)

[Table 4.30: Combination of Associations Mapping Examples 29](#_Toc430330783)

[Table 5.1: Recursion Mapping Examples 31](#_Toc430330784)

[Table 5.2: Mapping of Conditional Packages 32](#_Toc430330785)

[Table 5.2: Support and Condition Mapping Example 33](#_Toc430330786)

Document History

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Description of Change** |
| 0.0 | April 24, 2015 | Initial version. |
| 0.1 | May 29, 2015 | All defined YANG substatements added to the mapping tables. Mapping tables for individual types are added to section 4.3. Association mapping examples added to section 4.4. New section added for “UML Conditional Pacs” (0). New placeholder section added for “XOR Relationships” (5.3). |
| 0.2 | July 24, 2015 | All change marks from version 0.1 accepted. Contain changes made during the F2F Darmstadt Meeting. Section on XMI differences deleted. Proposals for mapping of recursion added in section 5.1. Comments provided by Victor taken into account. Object class mapping updated based on comments made on the IETF contribution. |
| 0.3 | Aug. 18, 2015 | All change marks from version 0.2 accepted. Comments provided by Yun Xiang on pre- and post-conditions taken into account. Open Model Profile YANG Extensions module added in section . Link to new draft YANG package statement added in section . Comments provided by Italo Busi on the netmod list taken into account. Comments provided by Ari Mark Sodhi on the netmod list taken into account. Section “Mapping Issues” moved to subsection in Mapping Guidelines. passedByReference example added to . Mapping examples added to all artifacts. Mapping of UML Support and Condition added in section 5.4. Re-engineered ietf-inet-types.yang and ietf-yang-types.yang added in Figure 4.1. |
| 0.4 | Sept.18, 2015 | All change marks from version 0.3 accepted. Comments provided by Chen Qiaogang added. Association Mapping Summary added as Table 4.19. IP Address Mapping Examples as Table 4.29 Mapping issue: Combination of different associations added in section 4.11.4. Section 4.11 on Using types defined in YANG updated. “Fragment and module structure Guidelines” document removed from Figure 3.1. Mapping tables split into two parts: (1) Mappings required by currently used UML artifacts, (2) Mappings for remaining YANG substatements. YANG module header definition added in new section 8 Addendum 1. Other patterns to be defined added in new section 9 Addendum 2. |

# Introduction

This Technical Recommendation defines the guidelines that have to be taken into account when parts of the ONF-wide UML information model needs to be mapped to YANG. The primary intent is to start with the protocol-neutral model using UML and then map this UML to YANG.

Note:  
The reverse direction from YANG to UML is possible for the object class artifacts but has some issues to be taken into account; see also section 7.

# References

1. YANG - A Data Modeling Language for the Network Configuration Protocol (draft-ietf-netmod-rfc6020bis-06)
2. Guidelines for Authors and Reviewers of YANG Data Model Documents (draft-ietf-netmod-rfc6087bis-04)
3. A Guide to NETCONF for SNMP Developers  
   (by Andy Bierman, v0.6 2014-07-10)
4. YANG Central ([http://www.yang-central.org](http://www.yang-central.org/))
5. NetConf Central ([http://www.netconfcentral.org](http://www.netconfcentral.org/))
6. YANG patterns (<https://tools.ietf.org/html/draft-schoenw-netmod-yang-pattern>)
7. ONF TR-514 “UML Modeling Guidelines 1.0” (<https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/UML_Modeling_Guidelines_V1.0.pdf>)

# Overview

## Documentation Overview

This document is part of a series of Technical Recommendations. The location of this document within the documentation architecture is shown in Figure 3.1 below:



Figure 3.1: ONF Specification Architecture

# Mapping Guidelines

The mappings are defined in table format and are structured based on the UML artefacts defined in [7]. The first part of the table (coloured cells) show the mapping of the required UML artefacts to YANG. The second part of the table (uncoloured cells) show the potential mapping of the remaining YANG substatements (not used in the first part of the table) to UML artefacts.  
Example mappings are shown below the mapping tables.

Open issues are either marked in yellow and/or listed in a light red box.  
General mapping issues are defined in section 4.11.

## Mapping of Object Classes

Table 4.1: Object Class Mapping  
(Mappings required by currently used UML artifacts)

|  |  |  |
| --- | --- | --- |
| Object Class 🡪 “list” statement (key property, multiple instances) or “container” statement (single instance) or "grouping" statement | | |
| UML Artifact | YANG Artifact | Comments |
| documentation | “description” substatement |  |
| superclass(es) | "grouping" statement | Concrete superclasses are then mapped to container/list which uses these groupings. |
| abstract | "grouping" statement |  |
| objectCreationNotification [YES/NO/NA] | “notification” statement | See section 4.8 Goes beyond the simple “a notification has to be sent”; a tool can construct the signature of the notification by reading the created object. |
| objectDeletionNotification [YES/NO/NA] | “notification” statement | See section 4.8 Goes beyond the simple “a notification has to be sent”; a tool can construct the signature of the notification by providing the object identifier of the deleted object (i.e., not necessary to provide the attributes of the deleted object). |
| support | “if-feature” substatement | Support and condition belong together. If the “support” is conditional, then the “condition” explains the conditions under which the class has to be supported. |
| condition |
| operation | “action” substatement | YANG 1.0 supports only rpc 🡪 add prefix to the rpc name; i.e., objectClass::rpc; action requires YANG 1.1 |

Table 4.2: Object Class Mapping  
(Mappings for remaining YANG substatements)

|  |  |  |
| --- | --- | --- |
| Object Class 🡪 “list” statement (key property, multiple instances) or “container” statement (single instance) or "grouping" statement | | |
| UML Artifact | YANG Artifact | Comments |
| see section 5.3 | “choice” substatement |  |
|  | ~~“config” substatement~~ | not relevant to object class |
| constraints between attribute values 🡪 error notification? OCL? | “must” substatement |  |
| object identifier | list::“key” substatement |  |
| multiplicity on association | list::“min-elements” and “max-elements” substatements | min-elements default = 0 max-elements default = unbounded mandatory default = false |
| no need now | ~~list::“ordered-by” substatement~~ | not relevant to object class  ordered-by default = system |
| see section 5.2 | container::“presence” substatement |  |
| hyperlink? | “reference” substatement | Papyrus doesn’t support hyperlinks |
| lifecycle stereotypes | “status” substatement | "current", "deprecated", "obsolete", default = current |
| constraint property | list::“unique” substatement | UML is not able to define attributes as identifiers/keys for object instances  UML is not able to define a group of attributes to be unique as YANG can do using the “unique” substatement. |
| superclasses/inheritance  complex attribute | “uses” substatement | use of a complex data type as the type of the attribute; e.g., date and time, object creation data |
| {<constraint>} | “when” substatement |  |

Additional items to discuss:

- Inheritance via grouping and/or flatten the hierarchy 🡪 Grouping

- Do we want to map the “full blown” lifecycle stereotypes into YANG?

- Which YANG substatements are not necessary?

- Usage of the reference substatement need to be analysed in general.

- Usage of the must substatement need to be analysed in general.

Table 4.3: Object Class Mapping Example

|  | grouping SuperClass1 {  leaf attribute1 {  …  mandatory true;  }  leaf-list attribute2 {  …  min-elements 2;  max-elements 4;  } }  grouping SuperClass2 {  leaf attribute3 {  …  mandatory true;  }  leaf-list attribute4 {  …  min-elements 1;  } }  container SubClass {  …  uses SuperClass1;  uses SuperClass2;  leaf-list attribute5 {  …  }  leaf attribute6 {  …  } } |
| --- | --- |
|  | from IETF draft-dharini-netmod-g-698-2-yang-04:  module ietf-opt-if-g698-2 {  namespace "urn:ietf:params:xml:ns:yang:ietf-opt-if-  g698-2";  prefix ietf-opt-if-g698-2;   import ietf-interfaces {  prefix if;  }   …  augment "/if:interfaces/if:interface" {  description "Parameters for an optical interface";  container optIfOChRsSs {  description "RsSs path configuration for an   Interface";  container ifCurrentApplicationCode {  description "Current Application code of the  interface";  uses optIfOChApplicationCode;  }   container ifSupportedApplicationCodes {  config false;  description "Supported Application codes of  the interface";  uses optIfOChApplicationCodeList;  }   uses optIfOChPower;  uses optIfOChCentralFrequency;  } } … |

## Mapping of Attributes

Table 4.4: Attribute Mapping  
(Mappings required by currently used UML artifacts)

|  |  |  |
| --- | --- | --- |
| Attribute 🡪 “leaf” (single) or “leaf list” (multiple) statement | | |
| UML Artifact | YANG Artifact | Comments |
| documentation | “description” substatement |  |
| type | “type” substatement (built-in or derived type) |  |
| readOnly | “config” substatement (false) | config default = true |
| isOrdered | “ordered-by” substatement ("system" or "user”) | ordered-by default = system |
| multiplicity | “mandatory” or “min-elements” and “max-elements” substatements [0..1] => no mapping needed; is default [1] => mandatory substatement = true [0..x] => no mapping needed; is default [1..x] => min-elements substatement = 1 [0..3] => max-elements substatement = 3 | min-elements default = 0 max-elements default = unbounded mandatory default = false |
| defaultValue | "default" substatement | If a default value exists and it is the desired value, the parameter does not have to be explicitly configured by the user. |
| isInvariant | “extension” substatement 🡪 ompExt:isInvariant | See extensions YANG module in section 6.2. |
| valueRange | “range” or “length” substatement of “type” substatement |  |
| passedByReference | if passedByReference = true 🡪 type leafref { path “/<object>/<object identifier>"  if passedByReference = false 🡪 either “list” statement (key property, multiple instances) or “container” statement (single instance) | Relevant only to attributes that have an object class defined as their type. |
| support | “if-feature” substatement | Support and condition belong together. If the “support” is conditional, then the “condition” explains the conditions under which the class has to be supported. |
| condition |

Table 4.5: Attribute Mapping  
(Mappings for remaining YANG substatements)

|  |  |  |
| --- | --- | --- |
| Attribute 🡪 “leaf” (single) or “leaf list” (multiple) statement | | |
| UML Artifact | YANG Artifact | Comments |
| error notification? | “must” substatement |  |
| hyperlink? | “reference” substatement | Papyrus doesn’t support hyperlinks |
| lifecycle stereotypes | “status” substatement | "current", "deprecated", "obsolete", default = current |
| unit? | “units” substatement |  |
| {<constraint>} | “when” substatement |  |

Table 4.6: Attribute Type Mapping Example

|  | list Class1 {  key class1Id;  description "This object class models the...";   leaf class1Id {  type string;  mandatory true;  config false;  }   leaf attribute1 {  type string;  mandatory true;  }   leaf-list attribute2 {  type int8 {  range "1-100";  }  min-elements 2;  max-elements 6;  }   leaf attribute3 {  type boolean;  default true;  config false;  ompExt:isInvariant  }   leaf attribute4 {  type enumeration {  enum LITERAL\_1;  enum LITERAL\_2;  enum LITERAL\_3;  }  mandatory true;  default LITERAL\_2;  config false;  }  } |
| --- | --- |

## Mapping of Types

Table 4.7: Type Mapping

|  |  |  |
| --- | --- | --- |
| UML Artifact | YANG Artifact | Comments |
| Primitive Type | Built-In Type if defined; otherwise ?? | e.g., Integer, Boolean, String, Real  new built-in type? |
| Enumeration | “enum” statement |  |
| Basic Data Type | “typeDef” statement | e.g., MAC address, IP v4 address |
| Complex Data Type | “grouping” statement | e.g., date and time, object creation data |

Note:  
YANG allows also in-line enumerations which are not possible in UML

Additional items to discuss:

- 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 container and list statements.  
🡪 UML attributes mapped to leaf, leaf-list, container, list statements?

### Mapping of Primitive Types

Table 4.8: Primitive Type Mapping

|  |  |  |
| --- | --- | --- |
| Primitive Type 🡪 new built-in type? | | |
| UML Artifact | YANG Artifact | Comments |
| documentation | ?? |  |

### Mapping of Enumeration Types

Table 4.9: Enumeration Type Mapping  
(Mappings required by currently used UML artifacts)

|  |  |  |
| --- | --- | --- |
| Enumeration Type 🡪 “enum” statement typedef for reusable enumerations? identity statement? | | |
| UML Artifact | YANG Artifact | Comments |
| documentation | “description” substatement |  |
| literal name | enum name |  |
| literal integer | “value” substatement |  |
| hyperlink? | “reference” substatement | Papyrus doesn’t support hyperlinks |
| lifecycle stereotypes | “status” substatement | "current", "deprecated", "obsolete", default = current |

Table 4.10: Enumeration Type Mapping  
(Mappings for remaining YANG substatements)

|  |  |  |
| --- | --- | --- |
| Enumeration Type 🡪 “enum” statement typedef for reusable enumerations? identity statement? | | |
| UML Artifact | YANG Artifact | Comments |
| ?? | “if-feature” substatement |  |

Table 4.11: Enumeration Type Mapping Example

|  | direct usage:  container ClassH {  …  leaf attribute1 {  type enumeration {  enum LITERAL\_1;  enum LITERAL\_2;  enum LITERAL\_3;  default LITERAL\_2;  mandatory true;  }  } }  indirect usage:  Not clear how to define mandatory for attribute1.  container ClassH {  …  container attribute1 {  …  uses Enumeration1 {  refine attribute1 {  default LITERAL\_2;  }  }  } }  grouping Enumeration1 {  leaf attribute1 {  type enumeration {  enum LITERAL\_1;  enum LITERAL\_2;  enum LITERAL\_3;  }  } |
| --- | --- |

### Mapping of Basic Data Types

Table 4.12: Basic Data Type Mapping  
(Mappings required by currently used UML artifacts)

|  |  |  |
| --- | --- | --- |
| Basic Data Type 🡪 “typeDef” statement | | |
| UML Artifact | YANG Artifact | Comments |
| documentation | “description” substatement |  |
| type | “type” substatement (built-in 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. |
| hyperlink? | “reference” substatement | Papyrus doesn’t support hyperlinks |
| lifecycle stereotypes | “status” substatement | "current", "deprecated", "obsolete", default = current |

Table 4.13: Basic Data Type Mapping  
(Mappings for remaining YANG substatements)

|  |  |  |
| --- | --- | --- |
| Basic Data Type 🡪 “typeDef” statement | | |
| UML Artifact | YANG Artifact | Comments |
| unit? | “units” substatement | unit is not an explicit property of a data type |

Table 4.14: Basic Data Type Mapping Example

|  | from ietf-yang-types.yang:  typedef mac-address {  type string {  pattern '[0-9a-fA-F]{2}(:[0-9a-fA-F]{2}){5}';   }  description  "The mac-address type represents an IEEE 802 MAC address. The canonical representation uses lowercase characters.  In the value set and its semantics, this type is equivalent to the MacAddress textual convention of the SMIv2.";  reference  "IEEE 802: IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture RFC 2579: Textual Conventions for SMIv2";  } |
| --- | --- |

### Mapping of Complex Data Types

Table 4.15: Complex Data Type Mapping

|  |  |  |
| --- | --- | --- |
| Complex Data Type 🡪 “grouping” statement | | |
| UML Artifact | YANG Artifact | Comments |
| documentation | “description” substatement |  |
| not used | “action” substatement |  |
| see section 5.3 | “choice” substatement |  |
| hyperlink? | “reference” substatement | Papyrus doesn’t support hyperlinks |
| lifecycle stereotypes | “status” substatement | "current", "deprecated", "obsolete", default = current |
| complex attribute | “uses” substatement |  |

Table 4.16: Complex Data Type Mapping Example

|  | grouping DataTypeA {  …  leaf attribute1 {  …  type string;  mandatory true;  }  leaf attribute2 {  …  type int8;  }  leaf-list attribute3 {  …  type boolean;  } } |
| --- | --- |

## Mapping of Associations

Table 4.17: Association Mapping

|  |  |  |
| --- | --- | --- |
| Associations | | |
| UML Artifact | YANG Artifact | Comments |
| Inheritance | “grouping” or “augment” statement | Multiple inheritance can also be mapped using “groupings”.  Need to define when augment is used. Note: Augmentation can be conditional. |
| Composition with “passed by value” | “container” statement containing “list” statement(s) (multiple contained instances) or “container” statement(s) (single contained instances) | How to map “passed by reference”? |
| Aggregation with “passed by reference” | “leafref” statement | How to map “passed by value”? |

Table 4.18: Association Mapping Examples

|  | container ClassA {  …  leaf attribute1 {  …  }  leaf attribute2 {  …  }  container ClassB {  …  leaf attribute3 {  …  }  leaf attribute4 {  …  }  } } |
| --- | --- |
|  | container ClassC {  …  leaf attribute1 {  …  }  leaf attribute2 {  …  }  list ClassD {  key "name";  leaf name {  type string;  }  leaf attribute4 {  …  }  } } |
|  | container ClassC {  …  leaf attribute1 {  …  }  leaf attribute2 {  …  }  leaf-list classD {  type leafref {  path “/ClassD/name";  }  } } |

The following table summarizes the association mappings.

Table 4.19: Association Mapping Summary

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

## Mapping of Interfaces (grouping of operations)

Table 4.20: UML Interface Mapping

|  |  |  |
| --- | --- | --- |
| UML Interface 🡪 Container? | | |
| UML Artifact | YANG Artifact | Comments |
| documentation | “description” substatement |  |
| abstract | ?? |  |
| support | “if-feature” substatement | Support and condition belong together. If the “support” is conditional, then the “condition” explains the conditions under which the class has to be supported. |
| condition |

## Mapping of Operations

Table 4.21: 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 | “description” substatement |  |
| pre-condition | “extension” substatement🡪 ompExt: preCondition | RFC 6020: During the NETCONF <edit-config> processing errors are already send for: - Delete requests for non-existent data. - Create requests for existent data. - Insert requests with "before" or "after"  parameters that do not exist.  See extensions YANG module in section 6.2. |
| post-condition | “extension” substatement🡪 ompExt: postCondition | See extensions YANG module in section 6.2. |
| input parameter | “input” substatement |  |
| output parameter | “output” substatement |  |
| operation exceptions | “extension” substatement🡪 ompExt:operationExceptions | See extensions YANG module in section 6.2. |
| isOperationIdempotent | “extension” substatement🡪 ompExt:isOperationIdempotent | See extensions YANG module in section 6.2. |
| isAtomic | “extension” substatement🡪 ompExt:isAtomic | See extensions YANG module in section 6.2 Necessary?  Not in UML Guidelines (TR-514); need to be added. |
| support | “if-feature” substatement | Support and condition belong together. If the “support” is conditional, then the “condition” explains the conditions under which the class has to be supported. |
| condition |
| hyperlink? | “reference” substatement | Papyrus doesn’t support hyperlinks |
| lifecycle stereotypes | “status” substatement | "current", "deprecated", "obsolete", default = current |

Table 4.22: Interface/Operation Mapping Example

|  | container InterfaceA {  …  action operation1 {  …  }  action operation2 {  …  } } |
| --- | --- |

## Mapping of Operation Parameters

Table 4.23: Parameter Mapping

|  |  |  |
| --- | --- | --- |
| Operation Parameters 🡪 “input” substatement or “output” substatement | | |
| UML Artifact | YANG Artifact | Comments |
| documentation | “description” substatement |  |
| direction | “input” / “output” substatement |  |
| type | see mapping of attribute types (grouping, leaf, leaf-list, container, list, typedef, uses) |  |
| isOrdered |  |
| multiplicity |  |
| defaultValue |  |
| valueRange |  |
| passedByReference | if passedByReference = true 🡪 type leafref { path “/<object>/<object identifier>"  if passedByReference = false 🡪 either “list” statement (key property, multiple instances) or “container” statement (single instance) | Relevant only to parameters that have an object class defined as their type. |
| support | “if-feature” substatement not defined for input and output substatements in YANG | Support and condition belong together. If the “support” is conditional, then the “condition” explains the conditions under which the class has to be supported. |
| condition |
| see section 5.3 | “choice” substatement |  |
| error notification? | “must” substatement |  |
| complex parameter | “uses” substatement |  |

Table 4.24: Interface/Operation/Parameter Mapping Example

|  | container InterfaceA {  …  action operation1 {  …  input {  leaf parameter1 {  type string;  mandatory true;  }  leaf parameter2 {  type boolean;  mandatory true;  }  }  output {  leaf parameter2 {  type boolean;  mandatory true;  }  leaf-list parameter3 {  type int16;  min-elements 3; }  }  }  action operation2 {  …  output {  leaf-list parameter4 {  type string;  }  }  } } |
| --- | --- |

## Mapping of Notifications

Table 4.25: Notification Mapping

|  |  |  |
| --- | --- | --- |
| Signal 🡪 “notification“ statement | | |
| UML Artifact | YANG Artifact | Comments |
| documentation | “description” substatement |  |
| support | “if-feature” substatement | Support and condition belong together. If the “support” is conditional, then the “condition” explains the conditions under which the class has to be supported. |
| condition |
| see section | “choice” substatement |  |
| error notification? | “must” substatement |  |
| hyperlink? | “reference” substatement | Papyrus doesn’t support hyperlinks |
| lifecycle stereotypes | “status” substatement | "current", "deprecated", "obsolete", default = current |
| attributes | see mapping of attribute types (grouping, leaf, leaf-list, container, list, typedef, uses) |  |
| complex attribute | “uses” substatement |  |

Table 4.26: Notification Mapping Example

|  | notification NotificationA {  …  leaf attribute1 {  type string  …  }  leaf attribute2 {  type integer  …  } } |
| --- | --- |

## Mapping of Lifecycle

Table 4.27: Lifecycle Mapping

|  |  |  |
| --- | --- | --- |
| UML Lifecycle | | |
| UML Artifact | YANG Artifact | Comments |
| Lifecycle Stereotypes | “status“ substatement | YANG: "current", "deprecated", "obsolete", default = current  UML: «Example», «Experimental», «Faulty», «LikelyToChange», «Deprecated», «Obsolete», «Preliminary»  How to map or enhance? |

## Other Mappings

Table 4.28: Other Mappings

|  |  |  |
| --- | --- | --- |
| UML Lifecycle | | |
| UML Artifact | YANG Artifact | Comments |
| Conditional Package | “container“ statement with “presence” substatement | See section 5.2. |
|  |  |  |
| Package | Submodule | See section 4.11.3. |

## Mapping Issues

### Using types defined in YANG?

Many common types (primitive and complex) are already defined in YANG. E.g., ietf-inet-types, ietf-yang-types (others to be investigated):

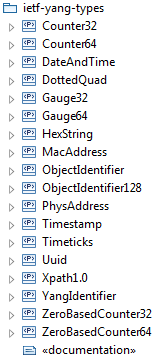
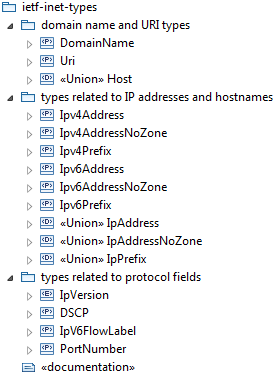


Figure 4.1: Example: Re-engineered ietf-inet-types.yang and ietf-yang-types.yang

It is proposed to define for the commonly used YANG types corresponding UML primitive or complex data types respectively. These types will be available (by default) for use in all UML information models. This “re-engineering” needs to be done without making the UML models YANG-depended.

Table 4.29: IP Address Mapping Examples

|  | ietf-inet-types.yang:: typedef ipv4-address |
| --- | --- |
|  | ietf-inet-types.yang:: ip-Address  {  type union {  type inet:ipv4-address;  type inet:ipv6-address;  } |

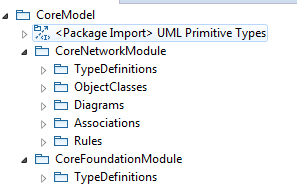
### YANG 1.0 or YANG 1.1?

YANG 1.0 is approved and defined in RFC 6020.  
YANG 1.1 is not approved and its definition is ongoing in draft-ietf-netmod-rfc6020bis (currently version 06). Main enhancements are the action and anydata statements.

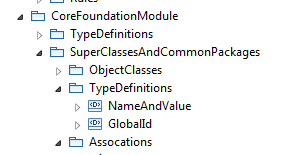
### Mapping of UML Packages?

Need to define mapping rules for UML package into YANG modules or the new draft YANG package statement (<http://datatracker.ietf.org/doc/draft-bierman-netmod-yang-package/>)?.

#1: For example, are CoreNetworkModule and CoreFoundationModule UML packages to be transformed into YANG modules? Are the TypeDefinitions and ObjectClasses UML packages then submodules? Or do we merge these two UML packages into one YANG module?



#2: Similar to #1, but what to do with embedded first-level UML packages, i.e. SuperClassesAndCommonPackages w/in CoreFoundationModule?



#3: As Karthik stated at the last meeting, we should have a separate module for the TypeDefinitions, perhaps one for each UML package, e.g. CoreFoundationModule-TypeDefinitions.yang. This would allow easier re-use across the YANG modules.

#4: Also, we need to discuss namespace and prefix for these YANG modules. There are certain IETF guidelines for the registry of prefixes…

### Combination of different Associations?

Table 4.30: Combination of Associations Mapping Examples

|  | container ClassJ {  …  list ClassL {  …  uses ClassK;  }  list ClassM {  …  uses ClassK;  } } |
| --- | --- |
|  | container ClassN {  …  container ClassP {  …  leaf-list classO {  type leafref {  path “/ClassO/name";  }  }  leaf classQ {  type leafref {  path “/ClassQ/name";  }  }  }  list ClassO {  …  container ClassQ {  …  {  } } |

# Mapping Patterns

## 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.

### Reference Based Approach

Table 5.1: Recursion Mapping Examples

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

## UML Conditional Pacs

May use the “presence” property of the container statement?

Table 5.2: Mapping of Conditional Packages

|  |  |
| --- | --- |
|  | list ClassE {  key "objectIdentifier";  leaf objectIdentifier {  type string;  }  …  leaf attribute2 {  …  }  container ClassF\_Pac {  presence " <condition for ClassF\_Pac attributes>";  …  leaf attribute3 {  …  }  leaf attribute4 {  …  }  }  container ClassG\_Pac {  presence " <condition for ClassG\_Pac attributes>";  …  leaf attribute5 {  …  }  leaf attribute6 {  …  }  } } |

## XOR Relationship

Use the “choice” property of the container statement.

## Mapping of UML Support and Condition

The UML Modeling Guidelines [7] define support and condition for all UML artifacts (M – Mandatory, O – Optional, C – Conditional, CM – Conditional-Mandatory, CO – Conditional-Optional). Support qualifies the support of the artifact at the management interface. Condition contains the condition for the condition-related support qualifiers.

M – Mandatory maps to the “mandatory” substatement in choice and leaf or to the “min-elements” substatement in leaf-list and list.

O – Optional need not be mapped since the per default the “mandatory” and “min-elements” substaments define optional.

All conditional UML support qualifiers are mapped to the “if-feature” substatement.

Table 5.2: Support and Condition Mapping Example

|  | feature ethernet {  description  "This feature means the network element provides Ethernet.";  }  container ClassI {  …  if-feature ethernet  leaf attribute1 {  …  } } |
| --- | --- |

# Mapping Basics

## UML 🡪 YANG or XMI 🡪 YANG

Figure 6.1: Example UML to YANG Mapping



Figure 6.2: Example XMI (Papyrus) to YANG Mapping

## Open Model Profile YANG Extensions

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

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

// Contents of "OpenModelProfileExtensions"

module OpenModelProfileExtensions {

namespace "urn:ONF:OpenModelProfileExtensions";

prefix "ompExt";

organization

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

description

"This module defines the Open Model Profile extensions for

usage in other YANG modules.";

revision 2015-07-28 {

description "Initial revision";

}

// extension statements

extension isInvariant {

description

"Used with attribute definitions to indicate that the value

of the attribute cannot be changed after it has been created.";

}

extension preCondition {

description

"Used with operation definitions to indicate the conditions

that have to be true before the operation can be started

(i.e., if not true, the operation will not be started at all

and a general “precondition not met” error will be returned,

i.e., exception is raised).";

argument "condition-list";

}

extension postCondition {

description

"Used with operation definitions to indicate the state of

the system after the operation has been executed (if

successful, or if not successful, or if partially successful).

Note that partially successful post-condition(s) can only

be defined in case of non-atomic operations.

Note that when an exception is raised, it should not be

assumed that the post-condition(s) are satisfied.";

argument "condition-list";

}

extension operationExceptions {

description

"Used with operation definitions to indicate the allowed

exceptions for the operation.

The model uses predefined exceptions which are split in

2 types:

- generic exceptions which are associated to all operations

by default

- common exceptions which needs to be explicitly associated

to the operation.

Note: These exceptions are only relevant for a protocol

neutral information model. Further exceptions may be

necessary for a protocol specific information model.

Generic exceptions:

• Internal Error: The server has an internal error.

• Unable to Comply: The server cannot perform the operation.

Use Cases may identify specific conditions that will result

in this exception.

• Comm Loss: The server is unable to communicate with an

underlying system or resource, and such communication is

required to complete the operation.

• Invalid Input: The operation contains an input parameter

that is syntactically incorrect or identifies an object

of the wrong type or is out of range (as defined in the

model or because of server limitation).

• Not Implemented: The entire operation is not supported

by the server or the operation with the specified input

parameters is not supported.

• Access Denied: The client does not have access rights

to request the given operation.

Common exceptions:

• Entity Not Found: Is thrown to indicate that at least

one of the specified entities does not exist.

• Object In Use: The object identified in the operation

is currently in use.

• Capacity Exceeded: The operation will result in resources

being created or activated beyond the capacity supported

by the server.

• Not In Valid State: The state of the specified object is

such that the server cannot perform the operation. In

other words, the environment or the application is not in

an appropriate state for the requested operation.

• Duplicate: Is thrown if an entity cannot be created because

an object with the same identifier/name already exists.";

argument "exception-list";

}

extension isOperationIdempotent {

description

"Used with operation definitions to indicate that the operation

is idempotent.";

}

// extension isAtomic {

// description

// "Used with operation definitions to indicate that the operation

// is atomic; i.e., has to be successful/unsuccessful as a whole.";

// }

}

<CODE ENDS>

# Reverse Mapping From YANG to UML

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

# Proposed Addendum 1: Requirements for the YANG Module header

<CODE BEGINS> file "<filename>@2015-<mm>-<dd>.yang"

// Content of "<filename>"

module <module name> {

namespace "urn:ONF:<module name>";

prefix "<prefix>";

import ietf-yang-types {

prefix yang;

}

import ietf-inet-types {

prefix inet;

}

import OpenModelProfileExtensions {

prefix ompExt;

}

organization

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

contact

"WG Web: <https://www.opennetworking.org/technical-communities/areas/services/>

WG List: <mailto: <wg list name>@opennetworking.org>

WG Chair: your-WG-chair

<mailto:your-WG-chair@example.com>

Editor: your-name

<mailto:your-email@example.com>";

description

"This module defines …

….";

revision <yyyy>-<mm>-<dd> {

description "<revision description>";

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Typedefs

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

…

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Groupings

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

…

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Main containers

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

…

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* RPC definitions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

…

}

<CODE ENDS>

# Proposed Addendum 2

Stereotype:Do not generate DS?

Identification of the root of the model?

Using spanning tree algorithm?

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