OPEN NETWORKING FOUNDATION

UML to OpenAPI Mapping Guidelines

TR-543 v1.0-info

February 28, 2018

Disclaimer

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.

Content

1 Introduction

This Technical Recommendation defines the guidelines for a mapping from a protocol-neutral UML information model to an OpenAPI (a.k.a Swagger API), which is a RESTful API with JSON data schema. The UML information model has to be defined based on the UML Modeling Guidelines defined in [1]. The OpenAPI is defined in [4].

2 References

[1] ONF TR-514 "UML Modeling Guidelines 1.1" (https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/UML_Modeling_Guidelines_Version_1-1.pdf)
[2] OpenModelProfile (https://github.com/OpenNetworkingFoundation/EAGLE-Open-Model-Profile-and-Tools/tree/OpenModelProfile)
[3] JSON Schema(http://json-schema.org/)
[4] The OpenAPI Specification(https://github.com/OAI/OpenAPI-Specification)

3 Abbreviations

DS	Data Schema
IM	Information Model
JSON	JavaScript Object Notation
NA	Not Applicable
REST	Representational State Transfer
SMI	Structure of Management Information
UML	Unified Modeling Language

4 Overview

4.1 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 4.1 below:

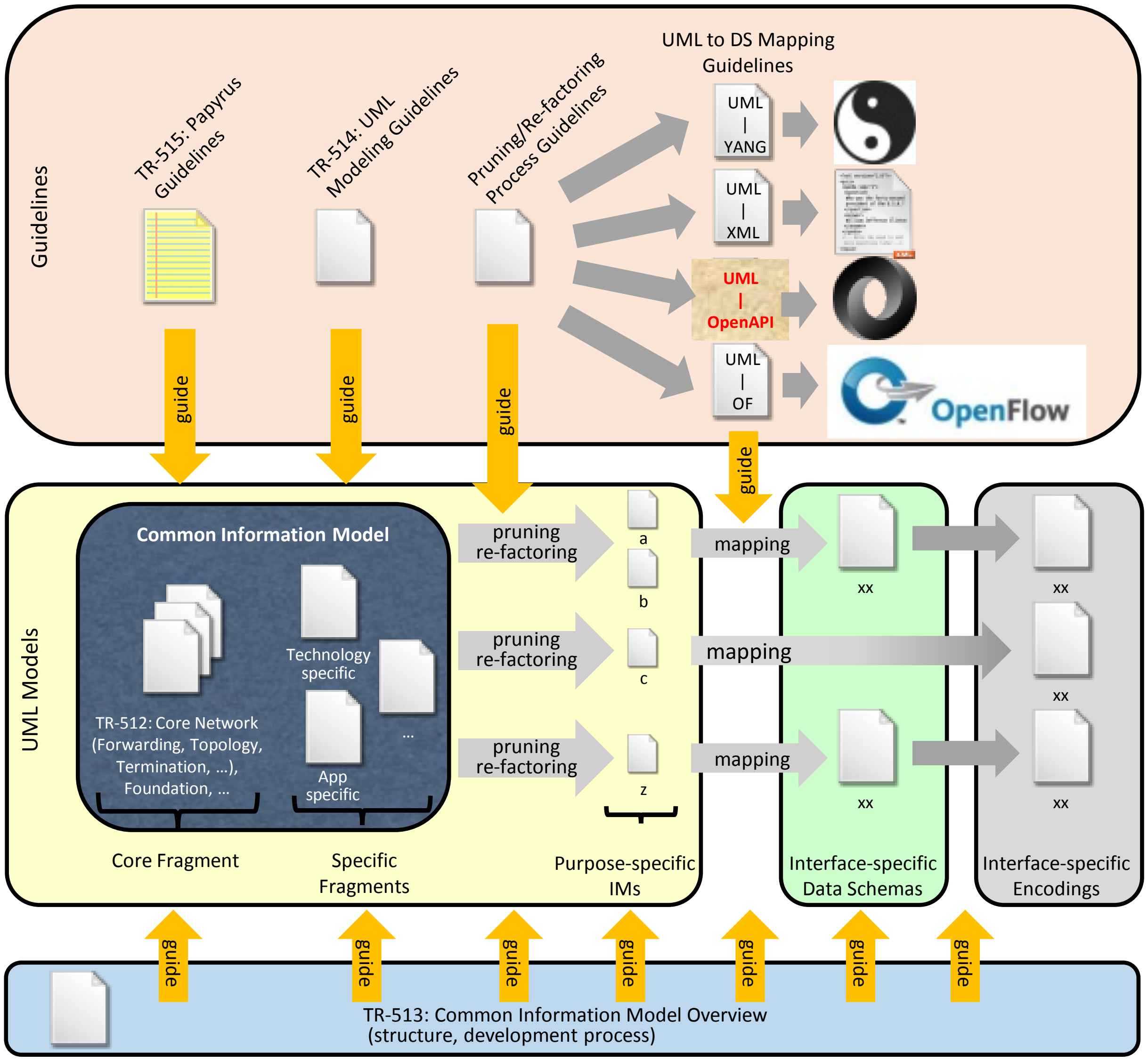

Guidelines
TR-515: Papyrus Guidelines
TR-514: UML Modeling Guidelines
Pruning/Re-factoring Process Guidelines
UML to DS Mapping Guidelines
UML | YANG
UML | XML
UML | OpenAPI
UML | OF
OpenFlow
guide
UML Models
Common Information Model
Technology specific
TR-512: Core Network (Forwarding, Topology, Termination, …), Foundation, …
App specific
…
Core Fragment
Specific Fragments
pruning re-factoring
a
b
c
z
Purpose-specific IMs
mapping
xx
Interface-specific Data Schemas
Interface-specific Encodings
TR-513: Common Information Model Overview (structure, development process)

Figure 4.1: ONF Specification Architecture

4.2 JSON Schema and JSON data

JSON Schema is a vocabulary that is used to annotate and validate JSON data documents. As stated in [3], the advantages of JSON Schema include:

- *describes your existing data format*
- *clear, human- and machine-readable documentation*
- *complete structural validation, useful for automated testing and validating client-submitted data*

On the other hand, JSON data or instance is the exact data exchanged over the API. Both JSON Schema and JSON data can be used for RESTful API specification.

In this document, UML-OpenAPI (RESTful API with JSON Schema) mapping guidelines will be specified.

5 UML- OpenAPI Mapping Guidelines

The UML- OpenAPI mapping rules are defined in table format and are structured based on the UML artifacts defined in [1]. For the JSON Schema Artifact in the table, <example text> means to replace the <example text> with “example text” in UML.

Example mappings are shown below the mapping tables.

Open issues are either marked in yellow and/or by comments.

5.1 Mapping of Classes

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

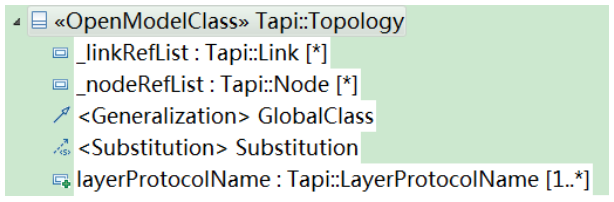
Class → “object” in "definitions" section		
UML Artifact	JSON Schema Artifact	Comments
documentation “Applied comments” (carried in XMI as “ownedComment”)	“description” substatement	Multiple “applied comments” defined in UML, need to be collapsed into a single “description” substatement.
Class Name	object name in "definitions" section: “<Class Name>-c”	The “-c” suffix indicates that this object is a class in UML.
attributes	Properties	See 5.2
object identifier	x-key/x-path	Note: Attributes used as object identifier are defined in UML by the attribute property “partOfObjectKey”>0. It is possible that the superclass or abstract class contains the key attribute for the instantiated subclass.

Class → “object” in "definitions" section		
UML Artifact	JSON Schema Artifact	Comments
Generalization Class	Combining schemas from the generalized Class and the target Class together with “allOf” statement.	
abstract	NA（Not Applicable）	
isLeaf	NA	
InterfaceModel_Profile:: objectCreationNotification [YES/NO/NA]	“object” in "definitions" section	See 5.7. Goes beyond the simple “a notification has to be sent”; a tool can construct the signature of the notification by reading the created object.
InterfaceModel_Profile:: objectDeletionNotification [YES/NO/NA]	“object” in "definitions" section	See 5.7. 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).
InterfaceModel_Profile::«RootElement»	NA	
multiplicity >1 on association to the class	See 5.2, the mapping of attributes with type= class	
OpenModel_Profile::«Reference»	NA	
OpenModel_Profile::«Example»	NA	
OpenModel_Profile::lifecycleState	NA	

Proxy Class;
XOR;
OpenModel_Profile::«Choice»
	NA	

Class → “object” in "definitions" section		
UML Artifact	JSON Schema Artifact	Comments
OpenModelClass::support	NA	
OpenModelClass::condition	NA	
Operation	See 5.5, 5.6	
Conditional Pac	NA	

Table 5.2: Class Mapping Example

 «OpenModelClass» Tapi::Topology
_linkRefList : Tapi::Link [*]
_nodeRefList : Tapi::Node [*]
<Generalization> GlobalClass
<Substitution> Substitution
layerProtocolName : Tapi::LayerProtocolName [1..*]
	"TapiTopology-c": {
	  "description": "The ForwardingDomain (FD) object class models... ",
	  "allOf": [
	    {
	      "$ref": "#/definitions/GlobalClass"
	    },
	    {
	      "properties": {
	        "_linkRefList": {
	          "items": {
	            "type": "string",
	            "x-path": "/Tapi_Link/uuid"
	          },
	          "type": "array"
	        },
	        "layerProtocolName": {
	          "items": {
	            "type": "string"
	          },
	          "type": "array"
	        },
	        "_nodeRefList": {
	          "items": {
	            "type": "string",
	            "x-path": "/Tapi_Node/uuid"
	          },
	          "type": "array"
	        }
	      }
	      "required": ["layerProtocolName"]
	    }
	  ]
	}

5.2 Mapping of Attributes

Table 5.3: Attribute Mapping
(Mappings required by currently used UML artifacts)

Attribute → property (key-value pair): Each key is the name of a property and each value is a JSON schema used to validate that property.		
UML Artifact	JSON Schema Artifact	Comments
documentation "Applied comments" (carried in XMI as "ownedComment")	"description" substatement	Multiple "applied comments" defined in UML, need to be collapsed into a single "description" substatement.
Attribute name	Property name	

Attribute type= Common Primitive Types	When Multiplicity⩽1:
	"<attribute name>": {
	"type": "<common primitive type>"
	}
	
	When Multiplicity>1:
	"<attribute name>": {
	"items": {
	"type": "<common primitive type>"
	},
	"type": "array",
	}
		Common PrimitiveType:
		1. string
		2. boolean
		3. integer

Attribute → property (key-value pair): Each key is the name of a property and each value is a JSON schema used to validate that property.		
UML Artifact	JSON Schema Artifact	Comments

Attribute type=
Complex Data Type
	When Multiplicity⩽1:
	"<attribute name>": {
	"$ref": "#/definitions/<attribute type>"
	}
	
	When Multiplicity>1:
	"<attribute name>": {
	"items": {
	"$ref": "#/definitions/<attribute type>"
	},
	"type": "array",
	"x-key":"<attribute name>"
	}
		The <attribute name> for "x-key" should be a Common Primitive Type attribute with partOfObjectKey >0 within the Complex Data Type.

Attribute → property (key-value pair): Each key is the name of a property and each value is a JSON schema used to validate that property.		
UML Artifact	JSON Schema Artifact	Comments

Attribute type= class (Multiplicity≤1)	When stereotype≠ StrictComposite:
	"<attribute name>": {
	"type": "string",
	"x-path": "/<attribute type>/<object identifier> "
	}
	
	When Aggregation=composite and stereotype=StrictComposite:
	"<attribute name>": {
	"$ref": "#/definitions/<attribute type>"
	}
		Attributes used as object identifier are defined in UML by the attribute property "partOfObjectKey">0.

Attribute → property (key-value pair): Each key is the name of a property and each value is a JSON schema used to validate that property.		
UML Artifact	JSON Schema Artifact	Comments

Attribute type= class
(Multiplicity>1)
	When stereotype≠ StrictComposite:
	"<attribute name>": {
	"items":{
	"type": "string",
	"x-path": "/<attribute type>/<object identifier> "
	},
	"type": "array"
	}
	
	When Aggregation=composite and stereotype=StrictComposite:
	"<attribute name>": {
	"items": {
	"$ref": "#/definitions/<attribute type>"
	},
	"type": "array",
	"x-key": "<attribute name>"
	}
		The <attribute name> for "x-key" should be a Common Primitive Type attribute with partOfObjectKey >0 within the class.

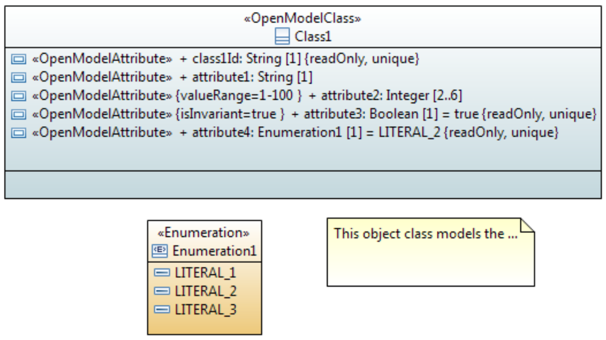
Multiplicity
(carried in XMI as lowerValue and upperValue)
	lowerValue => "minItems"
	upperValue=> "maxItems"	

Attribute → property (key-value pair): Each key is the name of a property and each value is a JSON schema used to validate that property.		
UML Artifact	JSON Schema Artifact	Comments
defaultValue	"default" : "<defaultValue>"	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.
isUnique	uniqueItems	Only apply to arrays. The value of this keyword MUST be a boolean. If this keyword has boolean value false, the instance validates successfully. If it has boolean value true, the instance validates successfully if all of its elements are unique. If not present, this keyword may be considered present with boolean value false.
isOrdered	Not supported by OpenAPI	
OpenModelAttribute:: valueRange	For integer and number typed attributes: --> minimum, maximum, exclusiveMinimum, exclusiveMaximum	When the value of “valueRange” is “null”, “NA”, “See data type”, the tool ignores it and doesn’t print out “range” substatement.

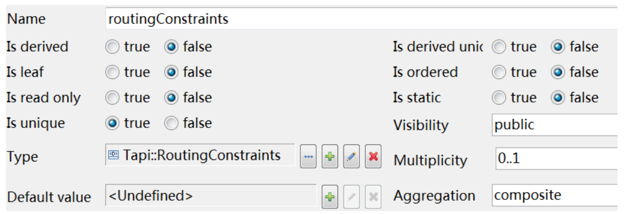
Attribute → property (key-value pair): Each key is the name of a property and each value is a JSON schema used to validate that property.		
UML Artifact	JSON Schema Artifact	Comments
OpenModelAttribute:: partOfObjectKey >0	Array::“x-key” substatement	It is possible that the (abstract) superclass contains the key attribute for the instantiated subclass.
OpenModelAttribute:: support=MANDATORY	Required Properties	
OpenModelAttribute:: partOfObjectKey >0	NA	See the mapping of attributes with type= class and type= Complex Data Type
OpenModelAttribute::isInvariant	NA	
OpenModelAttribute::unsigned	NA	
OpenModelAttribute::counter	NA	
InterfaceModelAttribute::unit	NA	
InterfaceModelAttribute::writeAllowed	NA	
InterfaceModelAttribute:: attributeValueChangeNotification	NA	
InterfaceModel_Profile::bitLength	NA	
InterfaceModel_Profile::encoding	NA	
OpenModel_Profile:: «PassedByReference»	NA	
OpenModel_Profile::«Reference»	NA	
OpenModel_Profile::«Example»	NA	
OpenModel_Profile::lifecycleState	NA	
OpenModelAttribute::support	NA	
OpenModelAttribute::condition	NA	

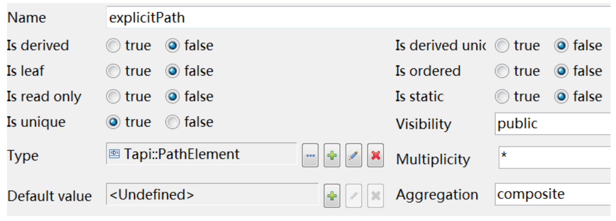
Table 5.4: Attribute Type Mapping Example

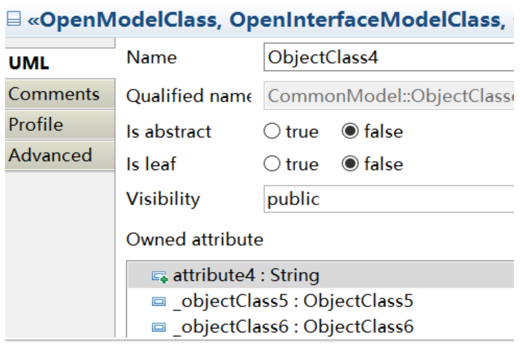
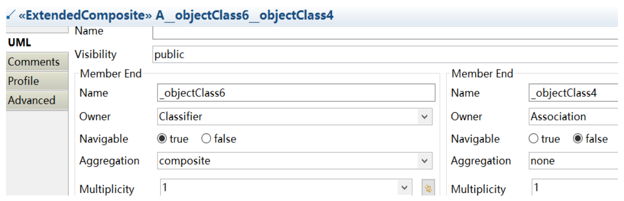
Attribute type= Common Primitive Types:

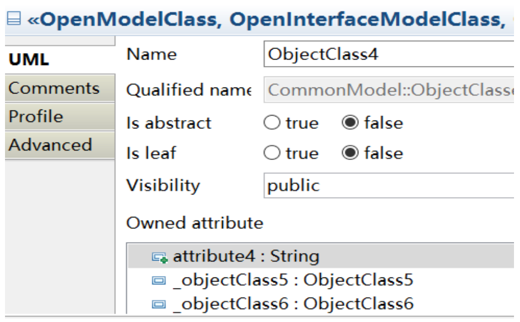
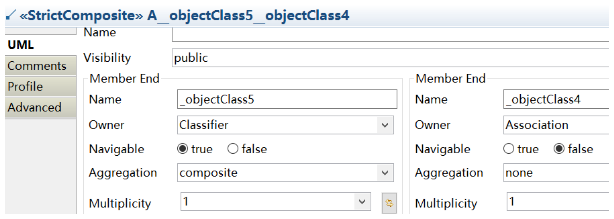

«OpenModelClass»
Class1
«OpenModelAttribute» + class1Id: String [1] {readOnly, unique}
«OpenModelAttribute» + attribute1: String [1]
«OpenModelAttribute» {valueRange=1-100 } + attribute2: Integer [2..6]
«OpenModelAttribute» {isInvariant=true } + attribute3: Boolean [1] = true {readOnly, unique}
«OpenModelAttribute»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

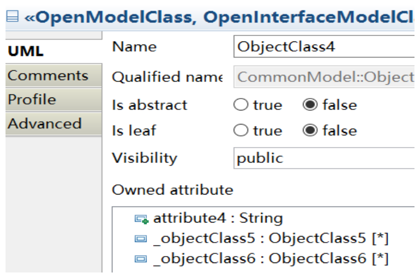
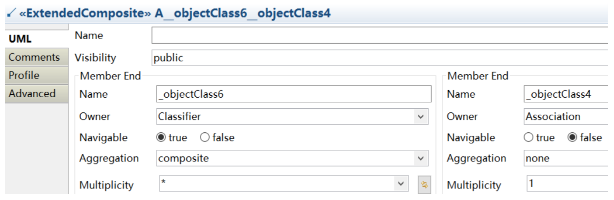
```
“Class1”: {
   “description”:"This class models the...",
    "properties": {
        “class1Id”: {"type": "string"},
        “attribute1”: {"type": "string"},
        “attribute2” : {
             "items": {
               "type": "integer"
                "minimum": 0,
                "maximum": 100,
               },
              "type": "array",
                "minItems": 2,
                "maxItems": 6
             }
        “attribute3” : {
            “type”: " boolean",
            "default" : true
        }

        “attribute4”: {
            "type": "string",
             "enum":
                ["LITERAL_1","LITERAL_2",
                  "LITERAL_3"],
            "default" : "LITERAL_2"
          }
    }
  "required": [“class1Id”, “attribute1”,
                 “attribute2”, “attribute3”,
                 “attribute4”]
}
```

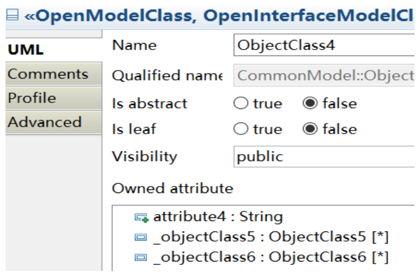
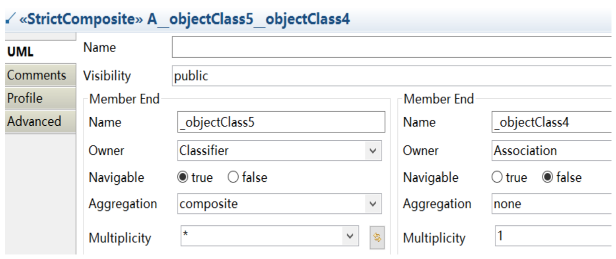
Attribute type= Complex Data Type (Multiplicity⩽1):

Name: routingConstraints
Is derived: true [✓ false]
Is leaf: true [✓ false]
Is read only: true [✓ false]
Is unique: [✓ true] false
Type: Tapi::RoutingConstraints
Default value: <Undefined>
Is derived unic: true [✓ false]
Is ordered: true [✓ false]
Is static: true [✓ false]
Visibility: public
Multiplicity: 0..1
Aggregation: composite
	"routingConstraints": {
	"$ref":
	#/definitions/TapiRoutingConstraints-d"
	}

Attribute type= Complex Data Type (Multiplicity>1):

Name: explicitPath
Is derived: true [✓ false]
Is leaf: true [✓ false]
Is read only: true [✓ false]
Is unique: [✓ true] false
Type: Tapi::PathElement
Default value: <Undefined>
Is derived unic: true [✓ false]
Is ordered: true [✓ false]
Is static: true [✓ false]
Visibility: public
Multiplicity: *
Aggregation: composite
	"explicitPath": {
	"items": {
	"$ref":
	"#/definitions/TapiPathElement-d"
	},
	"type": "array",
	"x-key": "_nodeEdgePointRef"
	}

Attribute type= class (Multiplicity⩽1), stereotype≠<StrictComposite >:

«OpenModelClass, OpenInterfaceModelClass,
UML
Comments
Profile
Advanced
Name: ObjectClass4
Qualified name: CommonModel::ObjectClass
Is abstract: true [✓ false]
Is leaf: true [✓ false]
Visibility: public
Owned attribute
attribute4 : String
_objectClass5 : ObjectClass5
_objectClass6 : ObjectClass6

«ExtendedComposite» A__objectClass6__objectClass4
UML
Comments
Profile
Advanced
Name:
Visibility: public
Member End
Name: _objectClass6
Owner: Classifier
Navigable: [✓ true] false
Aggregation: composite
Multiplicity: 1
Member End
Name: _objectClass4
Owner: Association
Navigable: true [✓ false]
Aggregation: none
Multiplicity: 1
	"_objectClass6": {
	"type": "string",
	"x-path": "/ObjectClass6-c/attribute61"
	}

Attribute type= class (Multiplicity⩽1),
Aggregation=composite, stereotype=<StrictComposite >:

«OpenModelClass, OpenInterfaceModelClass,
UML | Comments | Profile | Advanced
Name: ObjectClass4
Qualified name: CommonModel::ObjectClass
Is abstract: ○ true ● false
Is leaf: ○ true ● false
Visibility: public
Owned attribute:
attribute4 : String
_objectClass5 : ObjectClass5
_objectClass6 : ObjectClass6

«StrictComposite» A_objectClass5_objectClass4
UML | Comments | Profile | Advanced
Name:
Visibility: public
Member End
Name: _objectClass5; Owner: Classifier; Navigable: ● true ○ false; Aggregation: composite; Multiplicity: 1
Member End
Name: _objectClass4; Owner: Association; Navigable: ○ true ● false; Aggregation: none; Multiplicity: 1
	"_objectClass5": {
	"$ref": "#/definitions/ObjectClass5-c"
	}

Attribute type= class (Multiplicity>1)
stereotype≠<StrictComposite >:

«OpenModelClass, OpenInterfaceModelCl
UML | Comments | Profile | Advanced
Name: ObjectClass4
Qualified name: CommonModel::Object
Is abstract: ○ true ● false
Is leaf: ○ true ● false
Visibility: public
Owned attribute:
attribute4 : String
_objectClass5 : ObjectClass5 [*]
_objectClass6 : ObjectClass6 [*]

«ExtendedComposite» A_objectClass6_objectClass4
UML | Comments | Profile | Advanced
Name:
Visibility: public
Member End
Name: _objectClass6; Owner: Classifier; Navigable: ● true ○ false; Aggregation: composite; Multiplicity: *
Member End
Name: _objectClass4; Owner: Association; Navigable: ○ true ● false; Aggregation: none; Multiplicity: 1
	"_objectClass6": {
	"items":{
	"type": "string",
	"x-path": "/ObjectClass6-c/attribute61"
	},
	"type": "array"
	}

Attribute type= class (Multiplicity>1),
Aggregation=composite,
stereotype=<StrictComposite >:

«OpenModelClass, OpenInterfaceModelCl
UML
Comments
Profile
Advanced
Name: ObjectClass4
Qualified name: CommonModel::Object
Is abstract: true [✓ false]
Is leaf: true [✓ false]
Visibility: public
Owned attribute
attribute4 : String
_objectClass5 : ObjectClass5 [*]
_objectClass6 : ObjectClass6 [*]

«StrictComposite» A_objectClass5_objectClass4
UML
Comments
Profile
Advanced
Name:
Visibility: public
Member End
Name: _objectClass5
Owner: Classifier
Navigable: [✓ true] false
Aggregation: composite
Multiplicity: *
Member End
Name: _objectClass4
Owner: Association
Navigable: true [✓ false]
Aggregation: none
Multiplicity: 1
	"_objectClass5": {
	"items": {
	"$ref": #/definitions/ObjectClass5-c"
	},
	"type": "array",
	"x-key": "attribute51"
	}

5.3 Mapping of Data Types

Various kinds of data types are defined in UML:

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

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

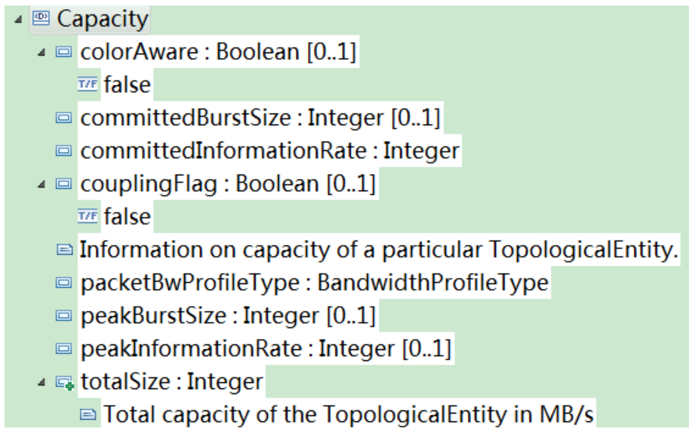
5.3.1 Generic Mapping of Complex Data Types

Table 5.5: Complex Data Type Mapping

Complex Data Type → "object" in "definitions" section		
UML Artifact	JSON Schema 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.

Complex Data Type → "object" in "definitions" section		
UML Artifact	JSON Schema Artifact	Comments
Complex Data Type Name	object name in "definitions" section: "<Complex Data Type Name>-d"	The "-d" suffix indicates that this object is a Complex Data Type in UML.
attributes	Properties	See 5.2
XOR OpenModel_Profile::«Choice»	NA	
OpenModel_Profile::«Reference»	NA	
OpenModel_Profile::«Example»	NA	
OpenModel_Profile::lifecycleState	NA	

Table 5.6: Complex Data Type Mapping Example



```
Capacity
  colorAware : Boolean [0..1]
    false
  committedBurstSize : Integer [0..1]
  committedInformationRate : Integer
  couplingFlag : Boolean [0..1]
    false
  Information on capacity of a particular TopologicalEntity.
  packetBwProfileType : BandwidthProfileType
  peakBurstSize : Integer [0..1]
  peakInformationRate : Integer [0..1]
  totalSize : Integer
    Total capacity of the TopologicalEntity in MB/s
```

```
    "Capacity": {
        "description": "Information on capacity of a particular TopologicalEntity.",
        "properties": {
            "committedInformationRate": {
                "type": "string"
            },
            "peakBurstSize": {
                "type": "string"
            },
            "totalSize": {
                "type": "string",
                "description": "Total capacity of the TopologicalEntity in MB/s"
            },
            "committedBurstSize": {
                "type": "string"
```

	}, "packetBwProfileType": { "type": "string" }, "peakInformationRate": { "type": "string" }, "couplingFlag": { "type": "boolean", "default" : false }, "colorAware": { "type": "boolean" "default" : false } } "required": ["totalSize","packetBwProfile Type", "committedInformationRate"] },

5.3.2 Mapping of Common Primitive Data Types

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

Table 5.7: Common Primitive Data Type Mapping

UML CommonDataTypes → JSON Schema Types		
UML Artifact	JSON Schema Artifact(type/format)	Comments
Boolean	boolean	
String	string	The length of a string can be constrained using the minLength and maxLength keywords. For both keywords, the value must

UML CommonDataTypes → JSON Schema Types		
UML Artifact	JSON Schema Artifact(type/format)	Comments
		be a non-negative number.
«LENGTH_32_BIT» Integer	integer/int32	Signed 32 bits
«LENGTH_64_BIT» Integer	integer/int64	Signed 64 bits
Integer		
«UNSIGNED, LENGTH_8_BIT» Integer	integer/int8	
«UNSIGNED, LENGTH_16_BIT» Integer	integer/int16	
«UNSIGNED, LENGTH_32_BIT» Integer	integer/int32	
«UNSIGNED, LENGTH_64_BIT» Integer	integer/int64	
«LENGTH_32_BIT» Real (float)	number/float	
«LENGTH_64_BIT» Real (double)	number/double	
DateTime	string/date-time	As defined by date-time - RFC3339
Uuid	Uuid	

5.3.3 Mapping of Enumeration Types

Table 5.8: Enumeration Type Mapping
(Mappings required by currently used UML artifacts)

Fixed Enumeration Type → "enum" statement		
UML Artifact	JSON Schema 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"

Fixed Enumeration Type → "enum" statement		
UML Artifact	JSON Schema Artifact	Comments
		substatement.
literal name	enum value within enum array	The enum keyword is used to restrict a value to a fixed set of values. It must be an array with at least one element, where each element is unique. Default type for enum value is string.
OpenModel_Profile::«Reference»	NA	
OpenModel_Profile::«Example»	NA	
OpenModel_Profile::lifecycleState	NA	

5.4 Mapping of Relationships

5.4.1 Mapping of Associations

All associations (i.e., pointers, composition aggregations and shared aggregations) 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); except the «StrictComposite» and «ExtendedComposite» associations which 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 3 kinds of association scenarios:

1. Pointers, composition aggregations and shared aggregations which are passed by reference
2. «StrictComposite» associations which are passed by value
3. «ExtendedComposite» associations which can also be somehow treated as passed by value.

Please refer to Table 5.3 for the examples of associations mapping.

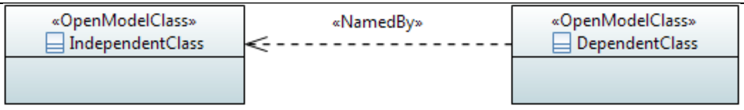
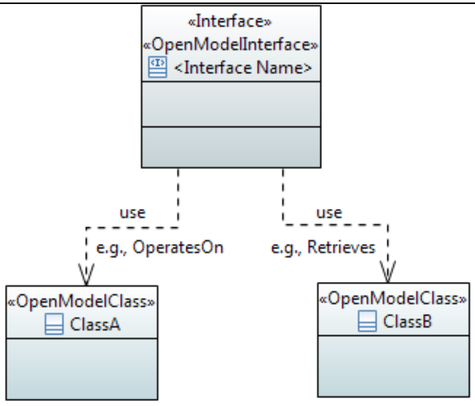
5.4.2 Mapping of Dependencies

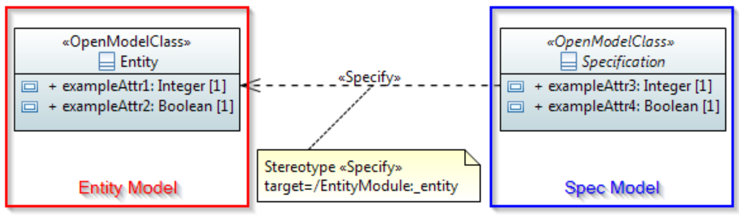
Three different kinds of dependency scenarios need to be mapped:

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

The mapping rules for the first two kinds are for further study. For the third one, a new combined schema will be created by using “allOf” statement to combine the schemas from the specify Class and the entity Class together.

Table 5.9: Dependency Mapping Examples

 «OpenModelClass» IndependentClass «NamedBy» «OpenModelClass» DependentClass	FFS
 «Interface» «OpenModelInterface» <Interface Name> use e.g., OperatesOn use e.g., Retrieves «OpenModelClass» ClassA «OpenModelClass» ClassB	FFS

 «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
	"entity_schema": {
	"allOf": [
	{
	"$ref": "#/definitions/entity"
	},
	{
	"properties": {
	" example-attr-3": {
	"type": "integer" },
	" example-attr-4": {
	"type": "boolean" }
	}
	]
	},
	
	"entity": {
	"properties": {
	" example-attr-1": {
	"type": "integer" },
	" example-attr-2": {
	"type": "boolean" }
	}
	}

5.5 Mapping of Interfaces and Operations

Table 5.10: Interface and Operation Mapping

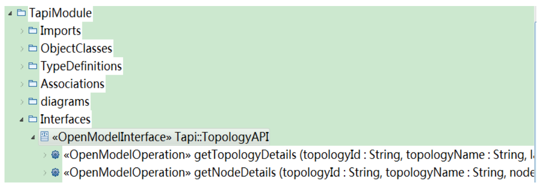
Interface and Operation →Paths, Path Item and Operation Object

UML Artifact	JSON Schema Artifact	Comments
documentation “Applied comments” (carried in XMI as “ownedComment”)	Operation summary	Multiple “applied comments” defined in UML, need to be collapsed into a single “summary”
Model name, Interface name	"description" and "title" field	

Model name, Interface name, Operation name	"paths":{
	 "/operations/<Model name>-<Interface name>:<Operation name>/”
	 }	

Operation name	Operation description	
Operation name	Operation parameters description: “<Operation name> body object”	
input parameter	Operation parameters schema:{"$ref": "#/definitions/<Operation name>RPCInputSchema"}	
Operation name	Operation parameters name	
output parameter	Response 200 schema:{"$ref": "#/definitions/<Operation name>RPCOutputSchema"}	
Operation name	Operation operationId	

Table 5.11: Interface/Operation Mapping Example



```
TapiModule
  Imports
  ObjectClasses
  TypeDefinitions
  Associations
  diagrams
  Interfaces
    «OpenModelInterface» Tapi::TopologyAPI
      «OpenModelOperation» getTopologyDetails (topologyId : String, topologyName : String, l
      «OpenModelOperation» getNodeDetails (topologyId : String, topologyName : String, node
```

```
  "paths": {
      "/operations/TapiModule-Interfaces-Tapi_TopologyAPI:getTopologyDetails/":
          {  "post": {
            "responses": {
              "200": {
                "description": "Successful operation",
                "schema": {
                  "$ref": "#/definitions/GetTopologyDetailsRPCOutputSchema"
                }
              },
              "400": {  "description": "Internal Error"    }
            },
            "description": "Create operation of resource: getTopologyDetails",
            "parameters": [
              {  "required": true,
                "description": "getTopologyDetailsbody object",
                "schema": { "$ref": "#/definitions/GetTopologyDetailsRPCInputSchema"
                },
                "name": "getTopologyDetails",
                "in": "body"
              }
            ],
            "produces": [ "application/json" ],
```

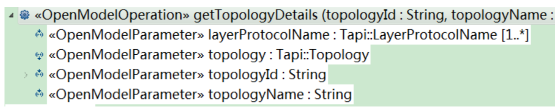
	"summary": "Create getTopologyDetails by ID", "consumes": ["application/json"], "operationId": "createGetTopologyDetailsById" } } "/operations/TapiModule-Interfaces-Tapi_TopologyAPI:getNodeDetails/": { …} },

5.6 Mapping of Operation Parameters

Table 5.12: Parameter Mapping

Operation Parameters → "RPCInputSchema" or "RPCOutputSchema" properties		
UML Artifact	JSON Schema 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	"RPCInputSchema" or "RPCOutputSchema"	
Parameter	property	see 5.2 Mapping of Attributes

Table 5.13: Interface/Operation/Parameter Mapping Example



```
«OpenModelOperation» getTopologyDetails (topologyId : String, topologyName :
    «OpenModelParameter» layerProtocolName : Tapi::LayerProtocolName [1..*]
    «OpenModelParameter» topology : Tapi::Topology
    «OpenModelParameter» topologyId : String
    «OpenModelParameter» topologyName : String
```

```
"GetTopologyDetailsRPCInputSchema": {
    "properties": {
        "topologyId": {
            "type": "string"
        },
        "layerProtocolName": {
            "items": {
                "type": "string"
            },
            "type": "array"
        },
        "topologyName": {
            "type": "string"
        }
    }
},
"GetTopologyDetailsRPCOutputSchema": {
    "properties": {
        "topology": {
            "$ref": "#/definitions/TapiTopology"
        }
    }
},
```

5.7 Mapping of Notifications

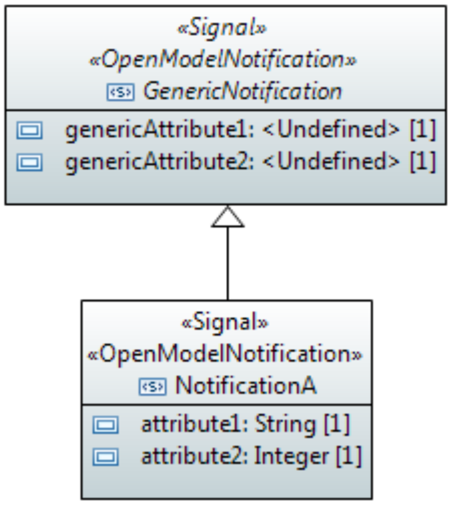
Like the class mapping, the signals are mapped to “object” in OpenAPI’s "definitions" section.

Table 5.14: Notification Mapping
(Mappings required by currently used UML artifacts)

Signal → “object” in "definitions" section		
UML Artifact	JSON Schema Artifact	Comments

Signal → “object” in "definitions" section		
UML Artifact	JSON Schema 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.
Signal Name	object name in "definitions" section: “< Signal Name>-s”	The “-s” suffix indicates that this object is a Signal in UML.
attributes	Properties	See 5.2
OpenModel_Profile::«Reference»	NA	
OpenModel_Profile::«Example»	NA	
OpenModel_Profile::lifecycleState	NA	
OpenModelNotification:: triggerConditionList	NA	
OpenModelNotification::support	NA	
OpenModelNotification::condition		
Proxy Class: See section **Error! Reference source not found.**. XOR: See section **Error! Reference source not found.**.	NA	

Table 5.15: Notification Mapping Example

 «Signal»
«OpenModelNotification»
GenericNotification
genericAttribute1: <Undefined> [1]
genericAttribute2: <Undefined> [1]
«Signal»
«OpenModelNotification»
NotificationA
attribute1: String [1]
attribute2: Integer [1]
	“NotificationA-s”: {
	"properties": {
	“attribute1”: {"type": "string"},
	“attribute2”: {"type": "integer"},
	}
	}

5.8 Mapping of UML Packages

The mapping tool shall generate a JSON Schema file per UML model.

According to the UML Modeling Guidelines [1], each UML model is basically structured into the following packages:

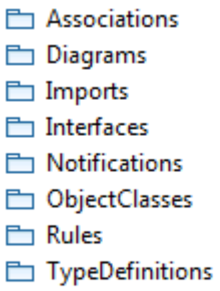


Figure 5.1: Pre-defined Packages in a UML Module

Table 5.16: UML Packages Mapping Example

TypeDefinitions ObjectClasses	″definitions″:{ … }
Interfaces	″paths″:{ … }

UML Package information (e.g. Model name, Interface name, Operation name, version, etc.) will be mapped to OpenAPI's Info Object. The generator shall generate the following information by using Info Object at the top of each JSON Schema file:

Table 5.17: UML Package Information Mapping Example

UML Artifact	JSON Schema Artifact	JSON Schema Type	Description
Model name, Interface name, Operation name	title	string	**Required.** The title of the application, e.g. "Model name+ Interface name+ Operation name".
Model name, Interface name, Operation name	description	string	A short description of the application. GFM syntax can be used for rich text representation. E.g. "Model name+ Interface name+ Operation name generated from UML by UML2OpenAPI tool".
Tbd	termsOfService	string	The Terms of Service for the API.
Tbd	contact	Contact Object	The contact information for the exposed API.
Tbd	license	License Object	The license information for the exposed API.
Tbd	version	string	**Required** Provides the version of the application API (not to be confused with the specification version).

For example:

```
title: Swagger Sample App
description: This is a sample server Petstore server.
termsOfService: http://swagger.io/terms/
contact:
  name: API Support
  url: http://www.swagger.io/support
  email: support@swagger.io
license:
  name: Apache 2.0
  url: http://www.apache.org/licenses/LICENSE-2.0.html
version: 1.0.1
```

6 Tool – User Interactions

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

6.1 General items

The tool needs to ask the user for the following information:

1. Add suffix “-c” to the object classes: No|Yes (default: No).
2. Add suffix “-d” to the complex data types: No|Yes (default: No).

6.2 Lifecycle State Treatment

UML elements are annotated by at least one of the following lifecycle states:

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

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

7 Contributors

Ruiquan Jing (Editor, China Telecom)

Guoyong Zhao(China Telecom)

Hing-Kam Lam (FiberHome)

Nigel Davis (Ciena)

Bernd Zeuner (Deutsche Telekom)

Chris Hartley (Cisco)

Italo Busi (Huawei)

Karthik Sethuraman (NEC)