Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 26 August 2026

K. Lambrechts
NetEdge

22 February 2026

Mapping YANG Schemas and Instance Data to TM Forum Data Models and APIs
draft-lambrechts-onsen-yang-tmf-mapping-00

Abstract

This document describes a mechanism for mapping between YANG schemas and instance data and TM Forum (TMF) Data Models and APIs. The mapping mechanism enables orchestrators to produce and expose TMF compliant interfaces that provide TMF API consumers direct access to a YANG data store.

Both a mechanism for the mapping of YANG schemas to TMF Catalog Specifications (i.e. TMF633/634) and a matching mechanism for the mapping of YANG instance data to TMF Order Management (i.e. TMF641/652) and Activation Management (i.e. TMF640/702) APIs are described.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 26 August 2026.

Copyright Notice

Table of Contents

1. Introduction

Service providers commonly use Operational Support Systems (OSS) with NETCONF/YANG interfaces (e.g., YANG models [RFC7950]) to describe configuration and operational state for network devices and services. Likewise, the TM Forum Open APIs are finding increased adoption in Business Support Systems (BSS) for a number of functions, include ordering and lifecycle management of services and resources.

These two ecosystems define different data models and interaction styles for overlapping concepts, which can lead to duplicated modeling effort and inconsistent behavior across platforms.

This document addresses this overlap by specifying a mechanism for mapping between YANG schemas and TM Forum Data Models and APIs. The goal is to enable consistent representation of resources and services across management interfaces and between domains, removing integration barriers and enables the use of YANG models directly in TMF-based BSS/OSS implementations.

The mapping mechanism is designed to preserve the semantics of the original YANG schema tree such as hierarchy, identity, and constraints while making them compatible with the TMF data structures and API patterns. A conscious effort is made to make minimal use of the support for polymorphism in the TMF APIs, as this would shift the implementation burden to the TMF consumers, hindering adoption and interoperability. Instead, the mechanism for mapping leverages existing TMF constructs such as Characteristics, Features, and Services/Resources as much as possible.

The mapping applies to YANG modules in general but is particularly relevant to IETF service and network models such as the L3SM [RFC8299], L3NM [RFC9182], L2SM [RFC8466], and L2NM [RFC9291].

The scope of this document is limited to the mechanism for mapping between YANG schemas/instance data and TM Forum data models and APIs, and the associated transformation rules. It does not define new protocols, nor does it mandate a specific implementation architecture. Operational concerns such as service orchestration workflows and business processes are out of scope, except where they affect the interpretation of the mapped data.

The mapping mechanism is only intended for translating YANG modules to TMF standards compliant Open APIs and payloads. The reverse translation (from OpenAPI to YANG) is not in scope.

1.1. Where the TMF Mapping Fits

The TM Forum Open APIs can be divided into several categories. Three of those categories are particularly relevant:

* Product
* Service
* Resource

The IETF separates service, network, and device models. All possible permutations on how these two architectures can be reconciled are out of scope for this document, but the mapping mechanism defined here is intended to be flexible enough to operate at either the TMF Service or Resource layer.

The following diagram illustrates a common architecture where TMF Open APIs are exposed at the northbound interface of an OSS or Service Orchestrator, while YANG-based models and automation drive the southbound interfaces to network controllers and devices. The mechanism for mapping defined in this document may operate.

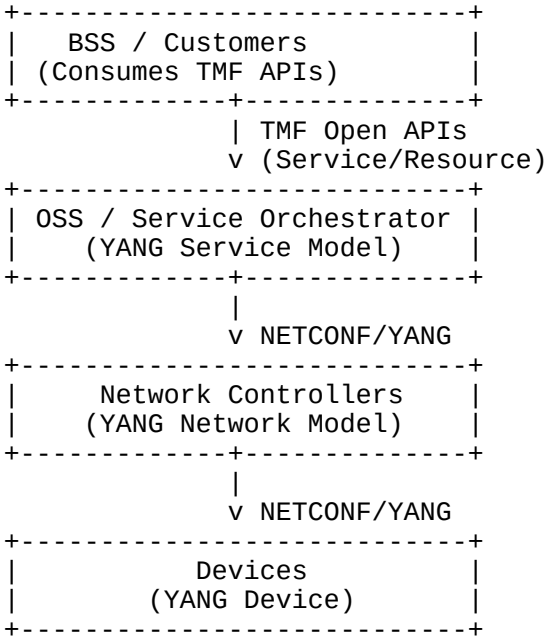

Figure 1: TMF mapping northbound of OSS/Service Orchestrator

For operators that handle more of the service lifecycle inside the TMF framework, the mapping mechanism can also be applied at the northbound interface of the Network Controller.

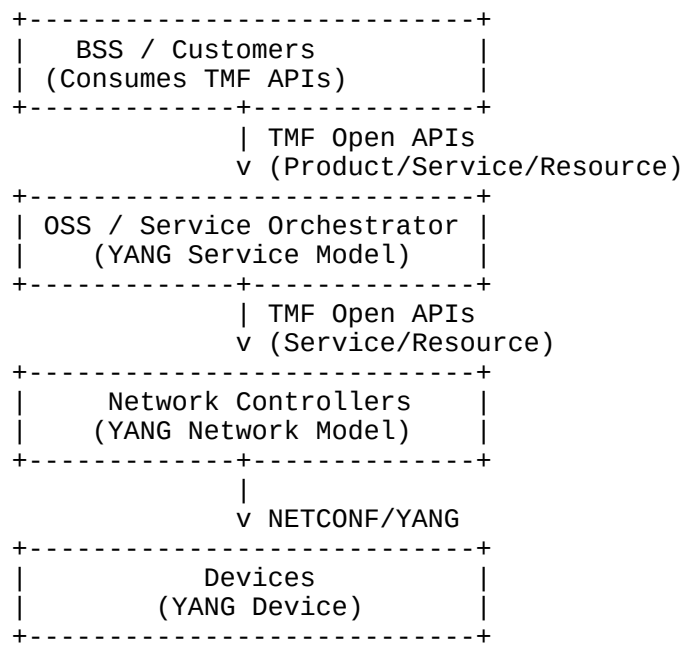

Figure 2: TMF mapping northbound of Network Controller

These boundaries are illustrative and can be adapted to operator architecture as needed. The key principle is that the mapping mechanism defined in this document can be applied at any point where YANG-based models and TMF Open APIs intersect.

1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. YANG Schema Trees and the TMF Data Models

YANG schema trees contain data nodes of the following types:

* container

* leaf

* leaf-list

* list

* anydata

* anyxml

The following sections describe how each of those constructs are mapped to the TMF Data Models and APIs.

2.1. Defining The Mapping Root

The mapping mechanism described in this document can be rooted at any container or list in the YANG schema tree. By default, the entire subtree starting from the mapping root SHOULD be mapped to TMF, except when the operator explicitly indicates that certain nodes or subtrees should be ignored by the mapping mechanism, e.g. by using the ignore extension in the YANG module below. Each schema node can be included in at most one mapping subtree, i.e., implementations MUST NOT allow overlapping mapping subtrees.

Implementations MAY use the following YANG extension to annotate the mapping root and the mapping method for a particular YANG subtree:

<CODE BEGINS> file "yang-tmf-map.yang"

```
module ietf-yang-tmf-map {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-tmf-map";
  prefix tmf;

  organization
    "IETF ONSEN Working Group.";
  contact
    "Kris Lambrechts
     Email: kris@netedge.plus";
  description
    "This YANG module defines a set of YANG extensions to annotate
     YANG schema nodes with mapping information to TMF Data Models.

     Copyright (c) 2026 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject to
     the license terms contained in, the Revised BSD License set
     forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
     (https://trustee.ietf.org/license-info).
```

```
        This version of this YANG module is part of RFC XXXX
        (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
        for full legal notices.";

    revision 2026-02-23 {
      description
        "Initial revision.";
      reference
        "RFC XXXX";
    }

    extension cfs-service {
      argument name;
      description
        "The TMF CFS Service that this YANG subtree maps to.";
    }

    extension rfs-service {
      argument name;
      description
        "The TMF RFS Service that this YANG subtree maps to.";
    }

    extension resource {
      argument name;
      description
        "The TMF Resource that this YANG subtree maps to.";
    }

    extension ignore {
      description
        "Indicates that the annotated YANG data node and its
         subtree are ignored by the mapping mechanism.";
    }
  }

  <CODE ENDS>
```

An operator can use the extension directly in the target YANG module, or they can use YANG augmentation to add the mapping to the relevant node(s). An example of the extension usage is as follows:

```
module example-toasters {
  yang-version "1.1";
  ...
  import yang-tmf-map {
    prefix tmf;
  }

  container toasters {
    list toaster {
      key name;
      tmf:cfs-service "Toaster";

      leaf name {
        type string;
        description "Toaster name";
      }
    }
  }
}
```

In this example, the toaster list is annotated with the tmf:cfs-service extension to indicate that it is mapped to a TMF CFS Service named "Toaster". The mapping mechanism will then map the child data nodes to the corresponding TMF schema elements based on the mapping rules defined in the next sections.

2.2. Mapping containers and lists to (sub)Features

Implementations MUST support mapping YANG schema trees of arbitrary depth below the mapping root. Each container or list in the schema tree can be represented as a TM Forum Service Feature or Resource Feature within the Service or Resource that it belongs to.

The current TM Forum Data Model does not support Feature nesting, i.e., there is no subFeature-like construct. This can be resolved, but requires a change to the TMF Data Model specification and the associated OpenAPI specifications. The proposed solution is described below.

DISCUSSION:

Should this document contain a proposed change to the TMF Data Model specification, or is there another preferred process for recommending such a change?

2.3. Mapping leafs and leaf-lists to Characteristics

YANG leafs and leaf-lists represent the fundamental data elements in a YANG data tree. Each leaf and leaf-list is to be mapped to a corresponding TMF Characteristic.

3. Rules for Mapping YANG Statements to the TMF Data Model

3.1. The "container" Statement

If the container is annotated with the tmf:cfs-service or tmf:rfs-service extension, it MUST be mapped to a TMF CFS Service or RFS Resource, respectively. Otherwise, it MUST be mapped to a TMF Feature.

Data nodes located directly under the mapping root MUST be mapped to Service or Resource Characteristics. Data nodes located in a first-level container or list are mapped to Feature Characteristics, while those located in deeper levels of the YANG schema tree are mapped to subFeature Characteristics and so on.

The Feature name SHOULD be derived from the YANG node name. The child data nodes of the container are mapped to the corresponding schema elements within the Feature.

3.2. The "leaf" Statement

Each YANG "leaf" statement MUST be mapped to a TMF Characteristic entry. The Characteristic name SHOULD be derived from the YANG node name. The Characteristic value MUST be encoded according to the JSON Schema rules for the JSON type matching the YANG type, as described in the next section.

3.2.1. The leaf’s "type" Statement

The YANG "type" statement MUST be fully resolved to a YANG built-in type and mapped to a JSON Schema type as follows:

YANG Type	JSON Schema Type
boolean	boolean
string	string
integer types (e.g., int32, uint64)	integer
decimal64	number
enumeration	string
identityref	string

Table 1

3.2.2. The leaf's "default" Statement

The YANG "default" statement MUST be mapped to a CharacteristicValueSpecification entry with the "isDefault" field set to true and the "value" field set to the default value specified in the YANG model.

3.2.3. The leaf's "mandatory" Statement

The YANG "mandatory" statement MUST be mapped to the CharacteristicSpecification "minCardinality" field with a value of 1 if the leaf is mandatory, and 0 otherwise. If a leaf is part of a list key, it MUST be considered mandatory regardless of the presence of the "mandatory" statement.

3.3. The "leaf-list" Statement

Each YANG "leaf-list" statement MUST be mapped to a TMF Characteristic entry. The Characteristic name SHOULD be derived from the YANG node name. The Characteristic value MUST be encoded as an array according to the JSON Schema rules for the JSON type matching the YANG type, as described in the previous section.

3.3.1. The leaf-list's "type" Statement

The YANG "type" statement for a leaf-list MUST be fully resolved to a YANG built-in type and mapped to a JSON Schema type as follows:

YANG Type	JSON Schema Type
boolean	booleanArray
string	stringArray
integer types (e.g., int32, uint64)	integerArray
decimal64	numberArray
enumeration	stringArray
identityref	stringArray

Table 2

3.3.2. The leaf-list's "default" Statement

The YANG "default" statement for a leaf-list MUST be mapped to a CharacteristicValueSpecification entry with the "isDefault" field set to true and the "value" field set to an array containing the default values specified in the YANG model.

3.3.3. The leaf-list's "min-elements" Statement

The YANG "min-elements" statement for a leaf-list MUST be mapped to the CharacteristicSpecification "minCardinality" field with a value corresponding to the constraint specified in the YANG model.

3.3.4. The leaf-list's "max-elements" Statement

The YANG "max-elements" statement for a leaf-list MUST be mapped to the CharacteristicSpecification "maxCardinality" field with a value corresponding to the constraint specified in the YANG model.

3.4. The "list" Statement

If the list is annotated with the tmf:cfs-service or tmf:rfs-service extension, it MUST be mapped to a TMF CFS Service or RFS Resource, respectively. Otherwise, it MUST be mapped to a TMF Feature.

Data nodes located directly under the mapping root MUST be mapped to Service or Resource Characteristics. Data nodes located in a first-level container or list are mapped to Feature Characteristics, while those located in deeper levels of the YANG schema tree are mapped to subFeature Characteristics and so on.

The Feature name SHOULD be derived from the YANG node name. The child data nodes of the list are mapped to the corresponding schema elements within the Feature.

3.4.1. The list’s "key" Statement

The list key(s) MUST be considered mandatory and mapped accordingly to the TMF CharacteristicSpecification "minCardinality" field.

3.4.2. The list’s "unique" Statement

The YANG "unique" statement for a list MUST be mapped to the CharacteristicSpecification "isUnique" field with a value of true.

4. Mapping YANG Built-In Types

4.1. The Integer Built-In Types

All YANG integer built-in types (i.e., int8, int16, int32, int64, uint8, uint16, uint32, uint64) MUST be mapped to the integer JSON Schema type.

4.1.1. The "range" Statement

The YANG "range" statement MUST be mapped to one or more CharacteristicValueSpecification entries with the "valueType" field set to "integer" and the "rangeInterval" set to "open". The "minValue" and "maxValue" fields MUST be set according to the range constraints specified in the YANG model. If the YANG "range" statement includes multiple intervals (separated by "|"), each interval MUST be mapped to a separate CharacteristicValueSpecification entry.

4.2. The String Built-In Types

The YANG built-in string type MUST be mapped to the string JSON Schema type.

4.2.1. The "length" Statement

The YANG "length" statement MUST be mapped to one or more CharacteristicValueSpecification entries with the "valueType" field set to "string" and the "regex" field set to a regular expression that enforces the length constraint. For example, a length constraint of "1..10" would be mapped to a regex of "^.{1,10}$". If the YANG "length" statement includes multiple intervals (separated by "|"), each interval MUST be mapped to a separate CharacteristicValueSpecification entry with the corresponding regex.

4.2.2. The "pattern" Statement

The YANG "pattern" statement MUST be mapped to the CharacteristicSpecification "regex" field with the same regular expression as specified in the YANG model.

4.3. The Boolean Built-In Type

The YANG boolean type MUST be mapped to the boolean JSON Schema type.

4.4. The Enumeration Built-In Type

The YANG enumeration type MUST be mapped to the string JSON Schema type.

4.4.1. The "enum" Statement

Each enum value defined in the YANG model MUST be mapped to a CharacteristicValueSpecification entry with the "valueType" field set to "string" and the "value" field set to the enumeration value.

5. Implementation Guidelines

An implementation of the mapping mechanism defined in this document SHOULD expose one or more of the following TMF Open APIs to allow TMF API comsumers to query the Catalog Service/ResourceSpecifications:

* TMF633 Service Catalog Management API [TMF633]

* TMF634 Resource Catalog Management API [TMF634]

The implementation SHOULD also expose one or more of the following TMF Open APIs to allow TMF API consumers to place orders and activate services/resources based on the mapped YANG schema tree:

* TMF640 Service Activation API [TMF640]

* TMF641 Service Order Management API [TMF641]

* TMF652 Resource Activation API [TMF652]

* TMF702 Resource Order Management API [TMF702]

6. IANA Considerations

6.1. YANG Module Registry

The YANG module defined in this document, yang-tmf-map, is intended to be registered in the IANA YANG Module registry. The module defines extensions that can be used to annotate YANG modules with TMF mapping information.

7. Security Considerations

This document defines a mapping between YANG schemas/instance data and TM Forum data models and APIs. The mapping itself introduces no new protocol elements, but it can affect how existing security properties are preserved when translating data between representations.

Implementations of this mapping MUST preserve the confidentiality, integrity, and authorization semantics of the source model. In particular, access control decisions tied to YANG nodes (including read, write, and notification permissions) MUST be enforced equivalently on the mapped TM Forum resources and operations. Failing to do so can result in privilege escalation or unintended disclosure.

When mapping data across administrative or trust boundaries, operators MUST ensure that transport security (e.g., TLS) and mutual authentication are used as appropriate for the deployed APIs. The mapping process MUST NOT bypass auditing, logging, or rate-limiting controls that are required for the underlying interfaces.

Finally, any reference data used by the mapping (such as schema registries, profiles, or transformation rules) SHOULD be protected from unauthorized modification, as tampering could lead to incorrect data interpretation or security policy violations.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8299] Wu, Q., Ed., Litkowski, S., Tomotaki, L., and K. Ogaki, "YANG Data Model for L3VPN Service Delivery", RFC 8299, DOI 10.17487/RFC8299, January 2018, <https://www.rfc-editor.org/info/rfc8299>.

[RFC8466] Wen, B., Fioccola, G., Ed., Xie, C., and L. Jalil, "A YANG Data Model for Layer 2 Virtual Private Network (L2VPN) Service Delivery", RFC 8466, DOI 10.17487/RFC8466, October 2018, <https://www.rfc-editor.org/info/rfc8466>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC9182] Barguil, S., Gonzalez de Dios, O., Ed., Boucadair, M., Ed., Munoz, L., and A. Aguado, "A YANG Network Data Model for Layer 3 VPNs", RFC 9182, DOI 10.17487/RFC9182, February 2022, <https://www.rfc-editor.org/info/rfc9182>.

[RFC9291] Boucadair, M., Ed., Gonzalez de Dios, O., Ed., Barguil, S., and L. Munoz, "A YANG Network Data Model for Layer 2 VPNs", RFC 9291, DOI 10.17487/RFC9291, September 2022, <https://www.rfc-editor.org/info/rfc9291>.

8.2. Informative References

[TMF633] TM Forum, "TMF633 Service Catalog Management API", TMF 633, 2019, <https://www.tmforum.org/oda/open-apis/directory/service-catalog-management-api-TMF633/>.

[TMF634] TM Forum, "TMF634 Resource Catalog Management API", TMF 634, 2019, <https://www.tmforum.org/oda/open-apis/directory/resource-catalog-management-api-TMF634/>.

[TMF640] TM Forum, "TMF640 Service Activation and Configuration API", TMF 640, 2019, <https://www.tmforum.org/oda/open-apis/directory/service-activation-and-configuration-api-TMF640/>.

[TMF641] TM Forum, "TMF641 Service Ordering API", TMF 641, 2019, <https://www.tmforum.org/oda/open-apis/directory/service-ordering-api-TMF641/>.

[TMF652] TM Forum, "TMF652 Resource Ordering Management API", TMF 652, 2019, <https://www.tmforum.org/oda/open-apis/directory/resource-ordering-management-api-TMF652/>.

[TMF702] TM Forum, "TMF702 Resource Activation and Configuration API", TMF 702, 2019, <https://www.tmforum.org/oda/open-apis/directory/resource-activation-and-configuration-api-TMF702/>.

[TMF630] TM Forum, "TMF630 REST API Design Guidelines", TMF 630, 2021, <https://www.tmforum.org/resources/specifications/tmf630-rest-api-design-guidelines-4-2-0/>.

Appendix A. Evaluation of mapping approaches for nested structures

YANG schema trees commonly represent complex data structures by composing containers, lists, leafs, and leaf-lists into YANG data trees that capture hierarchy, optionality, and repeated elements. Mapping those constructs into TM Forum Service and Resource Data Models can be approached in several ways, each with different trade-offs in expressiveness, TMF Service/Resource Catalog behavior, and operational clarity. Five possible approaches are identified in this section with their respective advantages and disadvantages.

The TMF630 REST API Design Guidelines [TMF630] allow for two overall patterns. The "strongly typed" pattern is described in the first approach. The Characteristic / CharacteristicSpecification pattern, has found wide adoption in TMF APIs and it is the basis for the remaining approaches.

A.1. Approach 1: Strongly Typed Pattern

When using the "strongly typed" pattern, it is assumed that each Service or Resource has a corresponding YAML or JSON schema defined in the @schemaLocation property. An example payload for a strongly typed Service may look like this:

```
{
  "id" : "id1234567890",
  "href" : "http//..",
  "state" : "active",
  "@type" : "conferenceBridgeEquipment",
  "@schemaLocation" : "http://../conferenceBridgeEquipment.json",
  "serviceSpecification":{
    "id":"conferenceBridgeEquipment",
    "href":"http://serverlocation:port/catalogManagement/
            serviceSpecification/conferenceBridgeEquipment"
  },
  "numberOfVc500Units": "1",
  "numberOfVc100Units": "2",
  "routerType": "CiscoASR1000",
  "powerSupply": "UK"
}
```

The downside of this approach is that it relies on the TMF API consumer to understand and process the schema referenced in the @schemaLocation property. Since this places a burden on the consumer, it can hinder adoption and interoperability, especially if the schema is complex or not widely supported. Therefore, the mechanism for mapping defined in this document focuses on the Characteristic-based approaches described in the following sections.

A.2. Approach 2: Complex Characteristics

A complete data structure can be embedded into a single Characteristic by using the TM Forum "Object" Value Type.

```
{
  "id": "123-456",
  "href": "http://serverlocation:port/
           serviceActivationConfiguration/v4/service/123-456",
  "category": "MobileService",
  "description": "Mobile Line ",
  "hasStarted": true,
  "isBundle": false,
  "isServiceEnabled": true,
  "isStateful": true,
  "serviceDate": "020-04-01T12:15:39.434Z",
  "startDate": "2020-04-03T12:15:39.434Z",
  "serviceCharacteristic": [
    {
      "id": "1",
      "name": "VoiceCharacteristic",
      "valueType": "object",
      "value": [
        {
          "voiceStandard": "enable",
          "voWifi": "disable",
          "roaming": "enable",
          "@schemaLocation": "../serviceSpecification/11//VC.json",
          "@type": "VoiceFeatureCharacteristic"
        }
      ],
      "@type": "ServiceCharacteristic"
    }
  ],
  "@type": "Service"
}
```

This approach has the advantages of fitting in with the Characteristic-based model which most TMF API implementations follow, but otherwise suffers from the same issue as the strongly typed pattern. This approach, too, relies on the consumer to understand and process the schema referenced in the @schemaLocation property. Additionally, it can make the Characteristic value opaque and difficult to validate or manipulate without understanding the schema.

A.3. Approach 3: Decomposition into Services/Resources

Each data tree layer could be mapped to its own TM Forum Service or Resource, linked through supportingService or corresponding Resource relationships. This approach enables explicit decomposition and reuse. However, each Service or Resource will appear in the Service Catalog or Resource Catalog as a separately orderable item.

If this approach was uniformly used for each YANG container or list, implementations would then need to expose and manage the appropriate relationships (e.g., parent-child, requires), and the resulting granularity can become too fine to be operationally meaningful. Especially for deeply nested YANG models such as the L3SM, this can lead to a large number of Services/Resources in the Catalog, making it difficult for operators to manage and for consumers to understand the available offerings.

This document uses this approach at the mapping root only.

A.4. Approach 4: Decomposition into Features

Each data tree layer can be represented as a TM Forum Service Feature or Resource Feature within a Service or Resource. This approach keeps Catalog entries manageable while still allowing for a structural and hierarchical representation inside the Service or Resource.

The current TM Forum Data Model does not support Feature nesting, i.e., there is no supportingFeature-like construct. Such construct COULD be added via a schemaLocation extension to represent feature hierarchy. However, it is the author’s opinion that for maximum interoperability, such nesting SHOULD be standardized in a future TM Forum Open API release.

This document uses this approach for each YANG container or list that is mapped below the root.

A.5. Approach 5: Encode Schema in Characteristic Names

The structure can be flattened by encoding path-like information in Characteristic names (e.g., a YANG path fragment). This approach would be simple to implement. However, it cannot represent cardinality accurately and tends to obscure the original YANG data tree, making validation and processing ambiguous.

This document does not use this approach, but it is mentioned here for completeness as the approach was evaluated during the design of the mapping mechanism.

Author’s Address

Kris Lambrechts
NetEdge
Email: kris@netedge.plus
URI: https://www.netedge.plus