MEF Standard
MEF 99

LSO Service Ordering Management API - Developer Guide

October 2023

Table of Contents

List of Contributing Members

The following members of the MEF participated in the development of this document and have requested to be included in this list.

Member
Amartus
Cisco
Lumen
Verizon

Table 1. Contributing Members

1. Abstract

This standard is intended to assist the implementation of the Application Programming Interfaces (APIs) for the Service Provisioning function of the Service Orchestration Functionality at the LSO Legato Interface Reference Point. The Legato Interface Reference Point is defined in the MEF 55.1 [MEF55.1] at the interface between the Business Application Systems layer and Service Orchestration Functionality layer.

This standard normatively incorporates the following files by reference as if they were part of this document from the GitHub repository:

MEF-LSO-Legato-SDK

commit id: 0e83943f529e87c036a083926a1b28a0a3523c5e

- `serviceApi/order/serviceOrderingManagement.api.yaml`
- `serviceApi/order/serviceOrderingNotification.api.yaml`

2. Terminology and Abbreviations

This section defines the terms used in this document. In many cases, the normative definitions to terms are found in other documents. In these cases, the third column is used to provide the reference that is controlling, in other MEF or external documents.

In addition, terms defined in the following documents are included in this document by reference, and are not repeated in the tables below.

- MEF 55.1, *Lifecycle Service Orchestration (LSO): Reference Architecture and Framework* February 2021 [MEF 55.1]

Term	Definition	Source
API Endpoint	The endpoint of a communication channel (the complete URL of an API Resource) to which the HTTP-REST requests are addressed in order to operate on the *API Resource*	rapidapi.com This document
API Resource	A REST Resource. In REST, the primary data representation is called Resource. In this document, *API Resource* is defined as a OAS *SchemaObject* with specified *API Endpoints*	restfulapi.net This document
Business Applications	The Service Provider functionality supporting Business Management Layer functionality	MEF 55.1
OAS Document	An API description document in the OpenAPI specification format.	openapis.org
OpenAPI	The OpenAPI 3.0 Specification, formerly known as the Swagger specification is an API description format for REST APIs.	spec.openapis.org
Operation	An interaction between the BUS and SOF, potentially involving multiple back and forth transactions.	This document
SchemaObject	The construct that allows the definition of input and output data types. These types can represent object classes, as well as primitives and arrays. specification	spec.openapis.org

Term	Definition	Source
Service Orchestration Functionality	The set of service management layer functionality supporting an agile framework to streamline and automate the service lifecycle in a sustainable fashion for coordinated management supporting design, fulfillment, control, testing, problem management, quality management, usage measurements, security management, analytics, and policy-based management capabilities providing coordinated end-to-end management and control of Services	MEF 55.1

Table 2. Terminology

Term	Definition	Source
API	Application Programming Interface. In this document, API is used synonymously with REST API.	This document
BUS	Business Applications	MEF 55.1
IRP	Interface Reference Point	This document
OAS	OpenAPI Specification	openapis.org
SOF	Service Orchestration Functionality	MEF 55.1

Table 3. Abbreviations

3. Compliance Levels

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 (RFC 2119 [RFC 2119], RFC 8174 [RFC8174]) when, and only when, they appear in all capitals, as shown here. All key words must be in bold text.

Items that are **REQUIRED** (contain the words **MUST** or **MUST NOT**) are labeled as **[Rx]** for required. Items that are **RECOMMENDED** (contain the words **SHOULD** or **SHOULD NOT**) are labeled as **[Dx]** for desirable. Items that are **OPTIONAL** (contain the words MAY or OPTIONAL) are labeled as **[Ox]** for optional.

A paragraph preceded by **[CRa]<** specifies a conditional mandatory requirement that **MUST** be followed if the condition(s) following the "<" have been met. For example, **"[CR1]<[D38]"** indicates that Conditional Mandatory Requirement 1 must be followed if Desirable Requirement 38 has been met. A paragraph preceded by **[CDb]<** specifies a Conditional Desirable Requirement that **SHOULD** be followed if the condition(s) following the "<" have been met. A paragraph preceded by **[COc]<**specifies a Conditional Optional Requirement that **MAY** be followed if the condition(s) following the "<" have been met.

4. Introduction

This standard specification document describes the Application Programming Interface (API) for Service Order Management functionality of the LSO Legato Interface Reference Point (IRP) as defined in the *MEF 55.1 Lifecycle Service Orchestration (LSO): Reference Architecture and Framework* [MEF55.1]. The LSO Reference Architecture is shown in Figure 1 with the IRP highlighted.

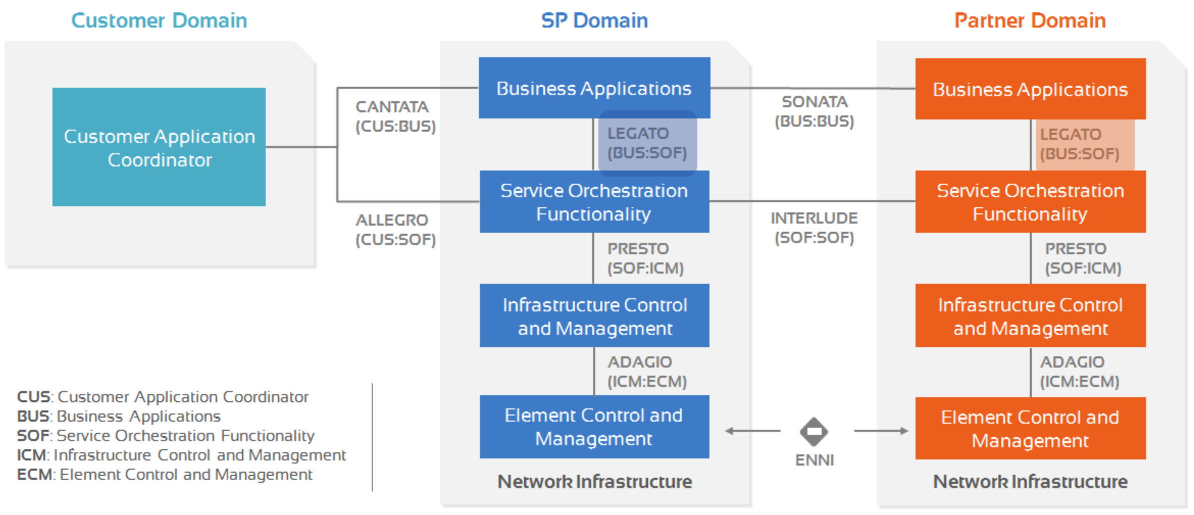

Customer Domain
SP Domain
Partner Domain
Customer Application Coordinator
Business Applications
Service Orchestration Functionality
Infrastructure Control and Management
Element Control and Management
Network Infrastructure
CANTATA (CUS:BUS)
ALLEGRO (CUS:SOF)
LEGATO (BUS:SOF)
PRESTO (SOF:ICM)
ADAGIO (ICM:ECM)
SONATA (BUS:BUS)
INTERLUDE (SOF:SOF)
ENNI
CUS: Customer Application Coordinator
BUS: Business Applications
SOF: Service Orchestration Functionality
ICM: Infrastructure Control and Management
ECM: Element Control and Management

Figure 1. The LSO Reference Architecture

4.1. Description

This standard is scoped to cover APIs for following Service Orchestration Functionalities:

- Service Ordering and Fulfillment
  - Includes Service Configuration & Activation functions
- Service Notification
  - Includes Event Subscription/Hub and Listener notification functions

Other Service Orchestration Functionalities not addressed in this standard include (but not limited to):

- Service Inventory Management
- Service Catalog Management
- Service Qualification
- Service Activation Testing
- Service Problem Management
- Service Quality Management
- Service Usage measurements and Reporting (in support of billing)
- License Management

This document primarily supports the requirements defined in section 8.2 (Order Fulfillment and Service Control) of MEF 55.1, LSO Reference Architecture for interactions over the Legato interface within a single operator. Both the Business Applications (BUS) and Service Orchestration Functionality (SOF) systems use the information contained within this document.

This standard is intended to support the design of API implementations that enable inter-operable SOF operations (in scope of this standard) across the Legato IRP.

This standard is based on TMF Open API (v4.1.0) for Service Ordering (TMF 641) TMF641.

4.2. Conventions in the Document

- Code samples are formatted using code blocks. When notation `<< some text >>` is used in the payload sample it indicates that a comment is provided instead of an example value and it might not comply with the OpenAPI definition.
- Model definitions are formatted as in-line code (e.g. `ServiceOrder`).
- In UML diagrams the default cardinality of associations is `0..1`. Other cardinality markers are compliant with the UML standard.
- In the API details tables and UML diagrams required attributes are marked with a `*` next to their names.
- In UML sequence diagrams `{{variable}}` notation is used to indicate a variable to be substituted with a correct value.

4.3. Relation to Other Documents

The API definition builds on *TMF641 Service Order Management API REST Specification v4.1.0* [TMF641]. Service Order Use Cases must support the use of any of MEF service specifications as payload, in particular those defined in:

- *LSO Legato Service Specification - SD-WAN Schema Guide* in MEF W100 [MEF W100].
- *LSO Legato Service Specification - Carrier Ethernet Schema Guide* in MEF W101 [MEF W101].
- *LSO Legato Service Specification - IP/IP-VPN Schema Guide* in MEF W102 [MEF W102].

4.4. Approach

As presented in Figure 2. the Legato API frameworks consist of three structural components:

- Generic API framework
- Service-independent information (Function-specific information and Function-specific operations)
- Service-specific information (MEF service specification data model)

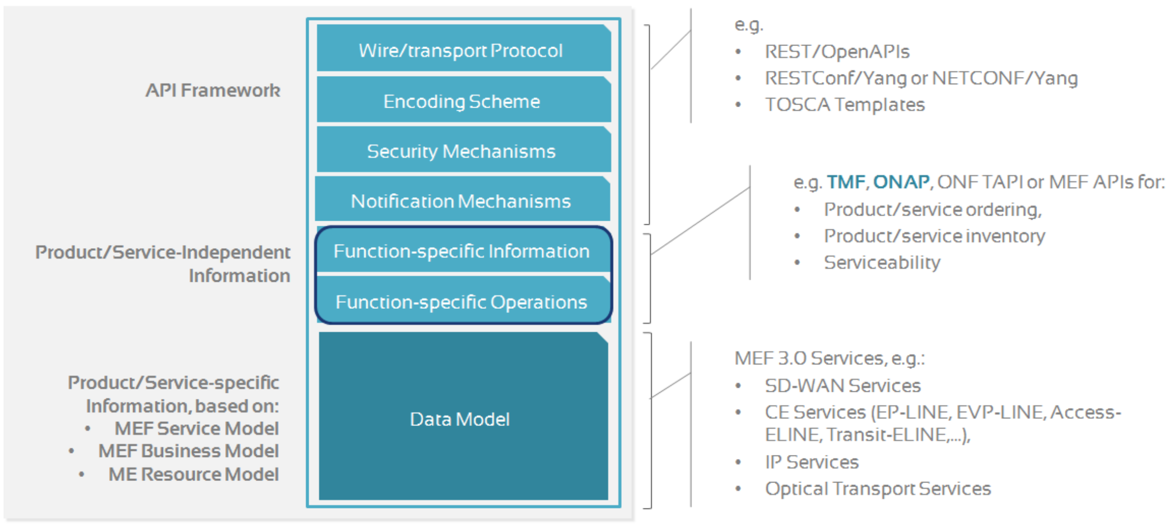

API Framework
Product/Service-Independent Information
Product/Service-specific Information, based on:
• MEF Service Model
• MEF Business Model
• ME Resource Model
Wire/transport Protocol
Encoding Scheme
Security Mechanisms
Notification Mechanisms
Function-specific Information
Function-specific Operations
Data Model
e.g.
• REST/OpenAPIs
• RESTConf/Yang or NETCONF/Yang
• TOSCA Templates
e.g. TMF, ONAP, ONF TAPI or MEF APIs for:
• Product/service ordering,
• Product/service inventory
• Serviceability
MEF 3.0 Services, e.g.:
• SD-WAN Services
• CE Services (EP-LINE, EVP-LINE, Access-ELINE, Transit-ELINE,...),
• IP Services
• Optical Transport Services

Figure 2. Legato API Structure

The essential concept behind the framework is to decouple the common structure, information, and operations from the specific service information content.
Firstly, the Generic API Framework defines a set of design rules and patterns that are applied across all Legato APIs.
Secondly, the service-independent information of the framework focuses on a model of a particular Legato functionality and is agnostic to any of the service specifications. For example, this standard is describing the Service Order model and operations that allow ordering of any service that is aligned with either MEF or custom service specifications.
Finally, the service-specific information part of the framework focuses on MEF service specifications that define business-relevant attributes and requirements for trading MEF subscriber and MEF operator services.

This Developer Guide is not defining MEF service specifications but can be used in combination with any service specifications defined by or compliant with MEF. Examples of MEF Service Model (MSM) schema include:

- MEF W100: SD-WAN Services based on MEF 70 [MEF70]
- MEF W101: Carrier Ethernet services based on MEF 10.4 [MEF10.4] and MEF 26.2 [MEF26.2]
- MEF W102: IP Services based on MEF 61.1 [MEF61.1] and MEF 61.1.1 [MEF61.1.1]

Figure 3 presents the relations between the Legato API components and the Service Model. A Service Order contains one or more Service Order Items. Each Service Order Item is an intent of action on a given Service (add, modify or delete). A Service references Service Specification to identify the Service Type. The Service specification points to the schema of the Service, as provided by (but not limited to) MEF Standard. The Service also has the

`MefServiceConfiguration` attribute, which provides an instance of the configuration of a given Service (attributes of MEF Service model populated with desired values)

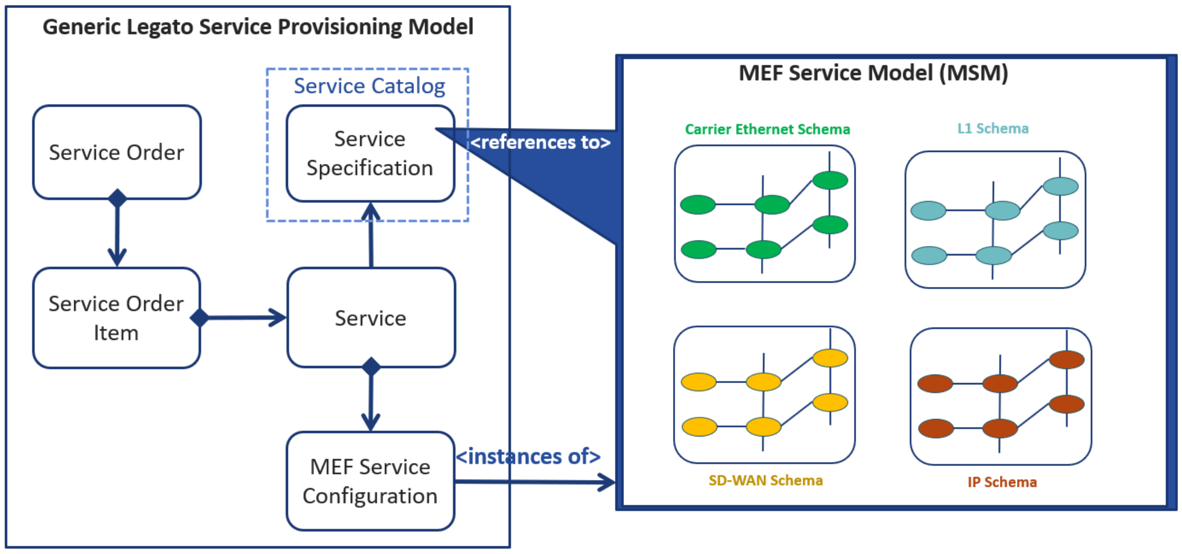


Figure 3. Legato and MSM Schema

4.5. High-Level Flow

The Legato Service Catalog, Service Order, Service Inventory, and Service Notification APIs in essence allow the BUS to request SOF to configure and activate one or more services as part of an order fulfillment process. Figure 4 presents a high-level flow of use of all of the above-mentioned APIs.

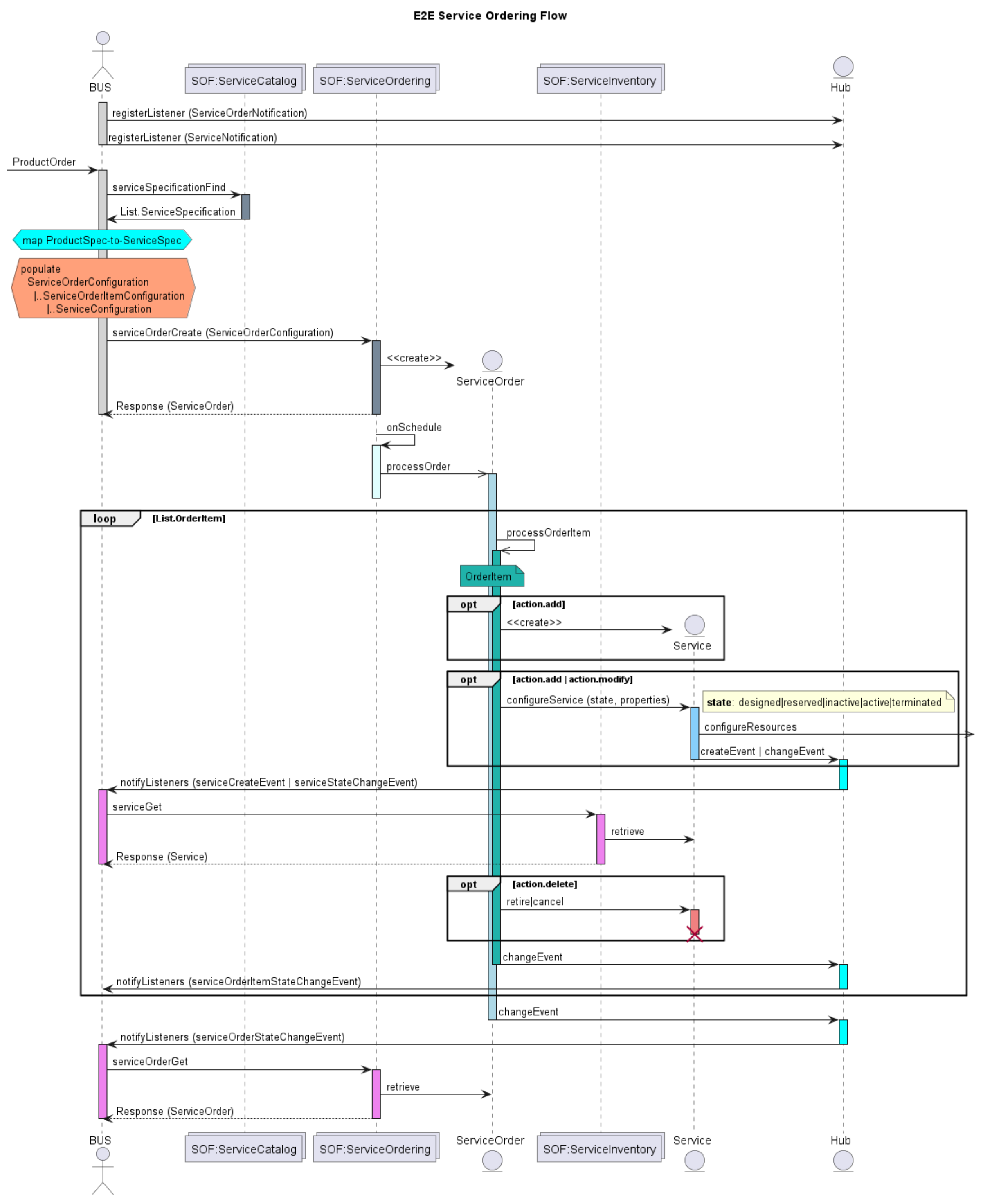

E2E Service Ordering Flow
BUS
SOF:ServiceCatalog
SOF:ServiceOrdering
SOF:ServiceInventory
Hub
registerListener (ServiceOrderNotification)
registerListener (ServiceNotification)
ProductOrder
serviceSpecificationFind
List.ServiceSpecification
map ProductSpec-to-ServiceSpec
populate
ServiceOrderConfiguration
|..ServiceOrderItemConfiguration
|..ServiceConfiguration
serviceOrderCreate (ServiceOrderConfiguration)
<<create>>
ServiceOrder
Response (ServiceOrder)
onSchedule
processOrder
loop
[List.OrderItem]
processOrderItem
OrderItem
opt
[action.add]
<<create>>
Service
opt
[action.add | action.modify]
configureService (state, properties)
state: designed|reserved|inactive|active|terminated
configureResources
createEvent | changeEvent
notifyListeners (serviceCreateEvent | serviceStateChangeEvent)
serviceGet
retrieve
Response (Service)
opt
[action.delete]
retire|cancel
changeEvent
notifyListeners (serviceOrderItemStateChangeEvent)
changeEvent
notifyListeners (serviceOrderStateChangeEvent)
serviceOrderGet
retrieve
Response (ServiceOrder)
BUS
SOF:ServiceCatalog
SOF:ServiceOrdering
ServiceOrder
SOF:ServiceInventory
Service
Hub

Figure 4. High-Level Flow

The following steps describe the high-level flow:

- The BUS system registers for notifications.
- As part of the ordering flow, the BUS system receives the product order (through Cantata or Sonata) which triggers the fulfillment processes in the BUS system.
- The BUS system first queries the *Service Catalog* to retrieve the `ServiceSpecifications` supported by the SOF
  Note1: *Service Catalog and the process of mapping and decomposing a product order to identify appropriate* `ServiceSpecifications` *is out of scope for this standard.* ***Note2***: *The*

mechanisms to design, construct and populate the `ServiceSpecifications` into SOF Service Catalog is out of scope for this standard.

- Each specific instance of a `ServiceSpecification` (retrieved from the *Service Catalog*) minimally contains a reference to target `Service` schema. A Service schema describes the set of properties that characterize that service and are exchanged over Legato IRP.

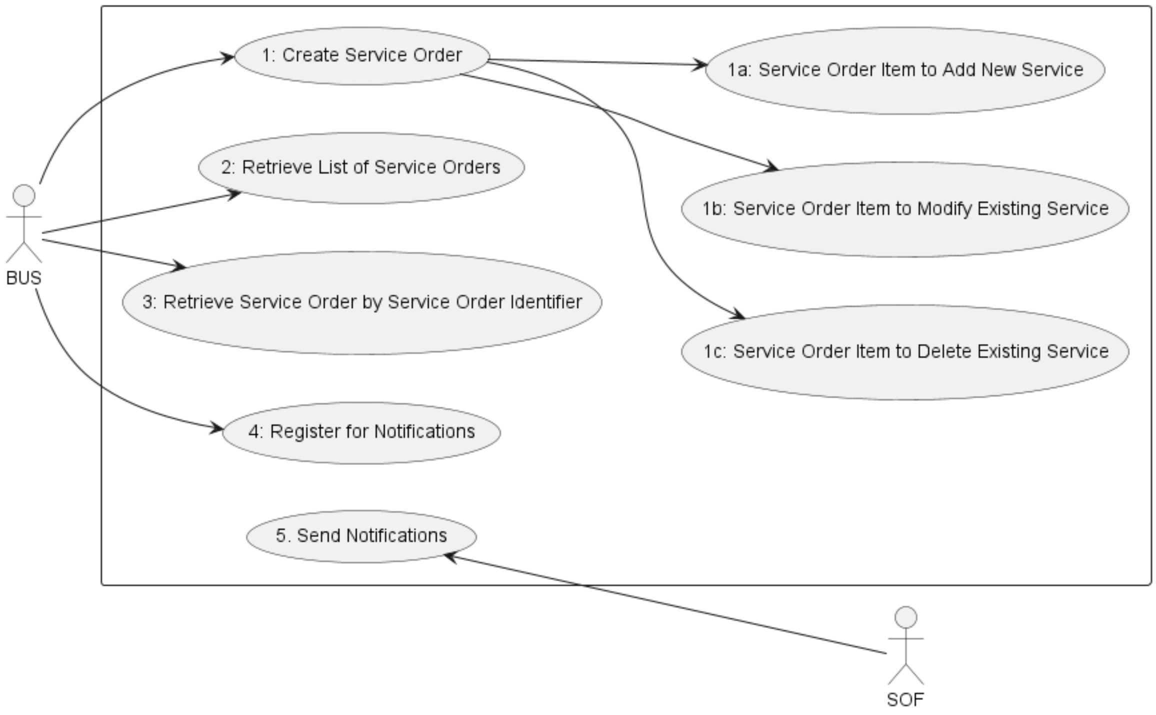
- During the service configuration and activation phase, the BUS system uses the *Service Order API* to instantiate the `Service` utilizing the `ServiceSpecifications` (retrieved from the *Service Catalog*).
    - The BUS achieves this by creating a `ServiceOrder` which contains a one or more `ServiceOrderItems`.
    - Each `ServiceOrderItem` carries some `ServiceConfiguration` data and the type of operation (*`add`*/*`modify`*/*`delete`*) to be performed (instructions to SOF).
    - The SOF utilizes `Service` schema referenced in the `ServiceSpecification` to validate the `ServiceConfiguration` data passed in by the BUS.
    - The `ServiceOrder` / `ServiceOrderItem` is processed by the SOF as per the state transition rules described in 6.1.7. Service Order and Service Order Items State Machine
    - The SOF reports the `ServiceOrder` and `ServiceOrderItem` state changes
    - The SOF performs the actions (*`add`*/*`modify`*/*`delete`*) specified in a `ServiceOrderItem` on the specified target `Service` instance in the *Service Inventory* as per the state transition rules described in 6.6. Service Lifecycle
    - The SOF reports the `Service` instance state changes
- The BUS system uses the same *Service Order API* to create **new** `Service` instances as well as update **existing** `Service` instance's properties or trigger state transitions, and delete **existing** `Service` instance.

5. API Description

This section presents the API structure and design patterns. It starts with the high-level use cases diagram. Then it describes the REST endpoints with use case mapping. Next, it gives an explanation of the design pattern that is used to combine service-agnostic and service-specific parts of API payloads. Finally, payload validation and API security aspects are discussed.

5.1. High-level Use Cases

Figure 5. presents a high-level use case diagram. It aims to help understand the endpoint mapping. Use cases are described extensively in chapter 6



1: Create Service Order
1a: Service Order Item to Add New Service
2: Retrieve List of Service Orders
1b: Service Order Item to Modify Existing Service
BUS
3: Retrieve Service Order by Service Order Identifier
1c: Service Order Item to Delete Existing Service
4: Register for Notifications
5. Send Notifications
SOF

Figure 5. Use cases

5.2. API Endpoints and Operations Summary

5.2.1. SOF Service Ordering API Endpoints

Base URL: `https://{{serverBase}}:{{port}}{{?/sof_prefix}}/mefApi/legato/serviceOrderingManagement/v5/`

The following API Endpoints are used by BUS to create and query for `ServiceOrder` instances and to subscribe/unsubscribe to `ServiceOrder` notifications. The endpoints and corresponding data model are defined in `serviceApi/order/serviceOrderingManagement.api.yaml`

API Endpoint	Description	Use Case mapping
`POST /serviceOrder`	A request initiated by the BUS to *create* new `Service` instances as well as *update* `Service` instance's properties or trigger their state transitions and/or *delete* existing `Service` instance.	UC 1: Create Service Order
`GET /serviceOrder`	A request initiated by the BUS to retrieve a list of `ServiceOrders` from the service order management system in SOF, that match the filter criteria provided as *`query`* parameters	UC 2: Retrieve List of Service Orders
`GET /serviceOrder/{{id}}`	A request initiated by the BUS to retrieve a specific `ServiceOrder` from the service order management system in SOF, that match the *`id`* provided as *`path`* parameter	UC 3: Retrieve Service Order by Service Order Identifier
`POST /hub`	A request initiated by the BUS to instruct the SOF to send notification	UC 4: Register for Notifications
`GET /hub/{{id}}`	A request initiated by the BUS to retrieve a specific `EventSubscription` from the service order management system in SOF, that matches the provided *`id`* provided as *`path`* parameter	UC 4: Register for Notifications
`DELETE /hub/{{id}}`	A request initiated by the BUS to instruct the SOF to stop sending notifications	UC 4: Register for Notifications

Table 4. SOF Service Ordering API Endpoints

[R1] SOF **MUST** support all API endpoints listed in Table 4.

5.2.2. BUS Service Ordering API Endpoints

Base URL: `https://{{serverBase}}:{{port}}{{?/bus_prefix}}/mefApi/legato/serviceOrderingNotification/v5/`

The following API Endpoints are used by SOF to post notifications to registered BUS listeners. The endpoints and corresponding data model are defined in `serviceApi/order/serviceOrderingNotification.api.yaml`

API Endpoint	Description	Use Case mapping
`POST /listener/serviceOrderCreateEvent`	A request initiated by the SOF to notify BUS on *`ServiceOrder`* instance creation	5. Send Notifications

API Endpoint	Description	Use Case mapping
`POST /listener/serviceOrderInformationRequiredEvent`	A request initiated by the SOF to notify BUS that additional information is required for given `ServiceOrder` instance	5. Send Notifications
`POST /listener/serviceOrderStateChangeEvent`	A request initiated by the SOF to notify BUS on `ServiceOrder` instance state change	5. Send Notifications
`POST /listener/serviceOrderItemStateChangeEvent`	A request initiated by the SOF to notify BUS on `ServiceOrderItem` instance state change	5. Send Notifications

Table 5. BUS Service Ordering API Endpoints

[O1] The BUS **MAY** support API endpoints listed in Table 5.

[O2] The BUS **MAY** register to receive service notifications.

[R2] The SOF **MUST** support sending notification to API endpoints listed in Table 5 to registered BUS.

5.3. Integration of Service Specifications into Service Order Management API

Service specifications are defined using JsonSchema (draft 7) format JSON Schema draft 7 and are integrated into the `ServiceOrder` using the TMF extension pattern.

The extension hosting type in the API data model is `MefServiceConfiguration`. The `@type` attribute of that type must be set to a value that uniquely identifies the service specification. A unique identifier for MEF standard service specifications is in URN format and is assigned by MEF. This identifier is provided as root schema `$id` and in service specification documentation. Use of non-MEF standard service definitions is allowed. In such a case the schema identifier must be agreed upon between the BUS and the SOF.

The example below shows a header of a Service Specification schema, which is describing the IP Uni, where `"$id": urn:mef:lso:spec:legato:ip-uni:v0.0.1:all` is the above-mentioned URN:

```
"$schema": http://json-schema.org/draft-07/schema#
"$id": $id": urn:mef:lso:spec:legato:ip-uni:v0.0.1:all
title: MEF LSO Legato - IP UNI Specification
```

Service specifications are provided as Json schemas without the `MefServiceConfiguration` context.

Service-specific attributes are introduced via the `ServiceValue` (defined by the BUS). This entity has the `serviceConfiguration` attribute of type `MefServiceConfiguration` which is used as an extension point for service-specific attributes.

Implementations might choose to integrate selected service specifications to data model during development. In such a case an integrated data model is built and service specifications are in an inheritance relationship with `MefServiceConfiguration` as described in the OAS specification. This pattern is called **Static Binding**. The SDK is additionally shipped with a set of API definitions that statically bind all service-related APIs (POQ, Quote, Order, Inventory) with all corresponding service specifications available in the release. The snippets below present an example of a static binding of the envelope API with several MEF service specifications, from both `MefServiceConfiguration` and service specification point of view:

```
MefServiceConfiguration:
  description:
    MefServiceConfiguration is used as an extension point for MEF-specific
    service payload. The `@type` attribute is used as a discriminator
  discriminator:
    mapping:
      urn:mef:lso:spec:legato:ip-enni:v0.0.1:all: '#/components/schemas/IpEnni'
      urn:mef:lso:spec:legato:ipvc-endpoint:v0.0.1:all: '#/components/schemas/IpvcEndpoint'
      urn:mef:lso:spec:legato:ip-uni:v0.0.1:all: '#/components/schemas/IpUni'
      urn:mef:lso:spec:legato:ethernet-uni-access-link-trunk:0.0.1:all:
'#/components/schemas/EthernetUniAccessLinkTrunk'
      urn:mef:lso:spec:legato:ip-uni-access-link:0.0.1:all: '#/components/schemas/IpUniAccessLink'
      urn:mef:lso:spec:legato:ipvc:v0.0.1:all: '#/components/schemas/Ipvc'
      urn:mef:lso:spec:legato:ip-uni-access-link-trunk.0.1:all: '#/components/schemas/IpUniAccessLinkTrunk'
      urn:mef:lso:spec:legato:ip-enni-link:v0.0.1:all: '#/components/schemas/IpEnniLink'
    propertyName: '@type'
  properties:
    '@type':
      description:
        The name of the type, defined in the JSON schema specified above, for
        the service that is the subject of the Request. The named type must be
        a subclass of MefServiceConfiguration.
      type: string
```

```
IpvcEndpoint:
  allOf:
    - $ref: '#/components/schemas/MefServiceConfiguration'
    - description:
        'An IPVC End Point is a logical entity at an EI, to which a subset of
        packets that traverse the EI is mapped. Reference MEF 61.1 Section 7.4
        IP Virtual Connections and IPVC End Points.'
```

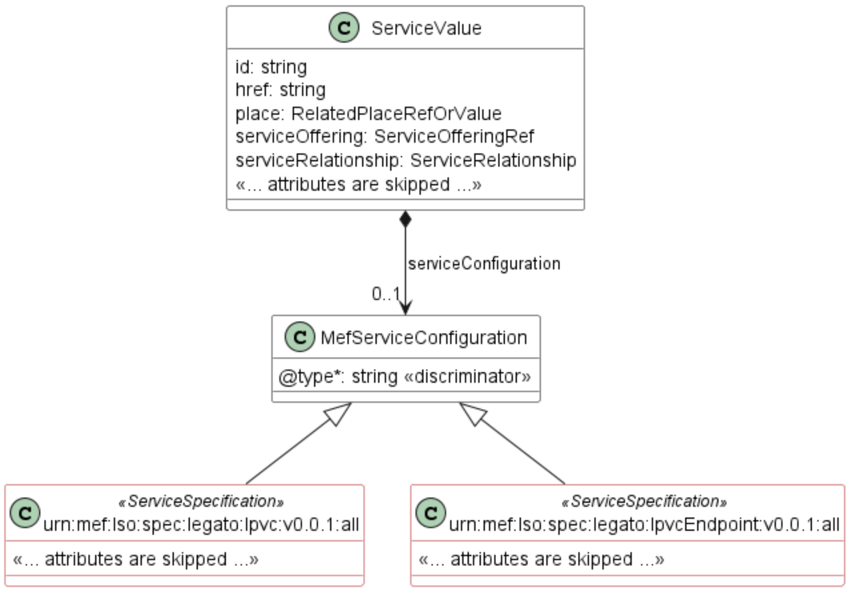
Alternatively, implementations might choose not to build an integrated model and choose a different mechanism allowing runtime validation of service-specific fragments of the payload. The system can validate a given service against a new schema without redeployment. This pattern is called **Dynamic Binding.**

Regardless of chosen implementation pattern, the HTTP payload is exactly the same. Both implementation approaches must conform to the requirements specified below.

[R3] `MefServiceConfiguration` type is an extension point that **MUST** be used to integrate service specifications' properties into a request/response payload.

[R4] The `@type` property of `MefServiceConfiguration` **MUST** be used to specify the type of the extending entity.

[R5] Service attributes specified in the payload must conform to the service specification specified in the `@type` property.



ServiceValue
id: string
href: string
place: RelatedPlaceRefOrValue
serviceOffering: ServiceOfferingRef
serviceRelationship: ServiceRelationship
«... attributes are skipped ...»
serviceConfiguration
0..1
MefServiceConfiguration
@type*: string «discriminator»
«ServiceSpecification»
urn:mef:lso:spec:legato:ipvc:v0.0.1:all
«... attributes are skipped ...»
«ServiceSpecification»
urn:mef:lso:spec:legato:ipvcEndpoint:v0.0.1:all
«... attributes are skipped ...»

Figure 6. The Extension Pattern with Sample Service-Specific Extensions

Figure 6 presents two MEF `<<ServiceSpecifications>>` that represent IPVC and IPVC Endpoint services. When these services are used as a Service Order payload the `@type` of `MefServiceConfiguration` takes `"urn:mef:lso:spec:legato:ipvc:v0.0.1:all"` or `"urn:mef:lso:spec:legato:ipvc-endpoint:v0.0.1:all"` value to indicate which service specification should be used to interpret a set of service-specific attributes included in the payload. An example of a service definition inside the `ServiceOrderItem` is presented in Section 6.1.4.

The *all* suffix after the service type name in the URN comes from the approach that the service schemas may differ depending on the function (POQ, Quote, Order, or Inventory) they are used with. The value *all* means that one version of the schema is shared by all functions.

5.4. Sample Service Specification

The Legato SDK contains service specification definitions, from which IPVC and IPVC End Point are used in the payload samples in this section. The schemas are located in the SDK package at:

- `serviceSchema\ip\ipvc.yaml`
- `serviceSchema\ip\ipvcEndPoint.yaml`

The service specification data model definitions are available as JsonSchema (version `draft 7`) documents. Figures 7 and 8 depict simplified UML views on these data models in which:

- the mandatory attributes are marked with `*`,
- the mandatory relations have a cardinality of `1` or `1..*`,
- some relations and attributes that are not essential to the understanding of the service specification model are omitted.

The red color in Figures 7 and 8 below highlights the data model of services. Some parts of the model are skipped for examples clarity. This is denoted by the `<<skipped>>` text in diagrams and in json snippets later in the document. Please note that this document uses service specifications just for the sake of example on how to use the Service Order API together with the Service payload. The detailed examples of any service specification are not in the scope of this document.

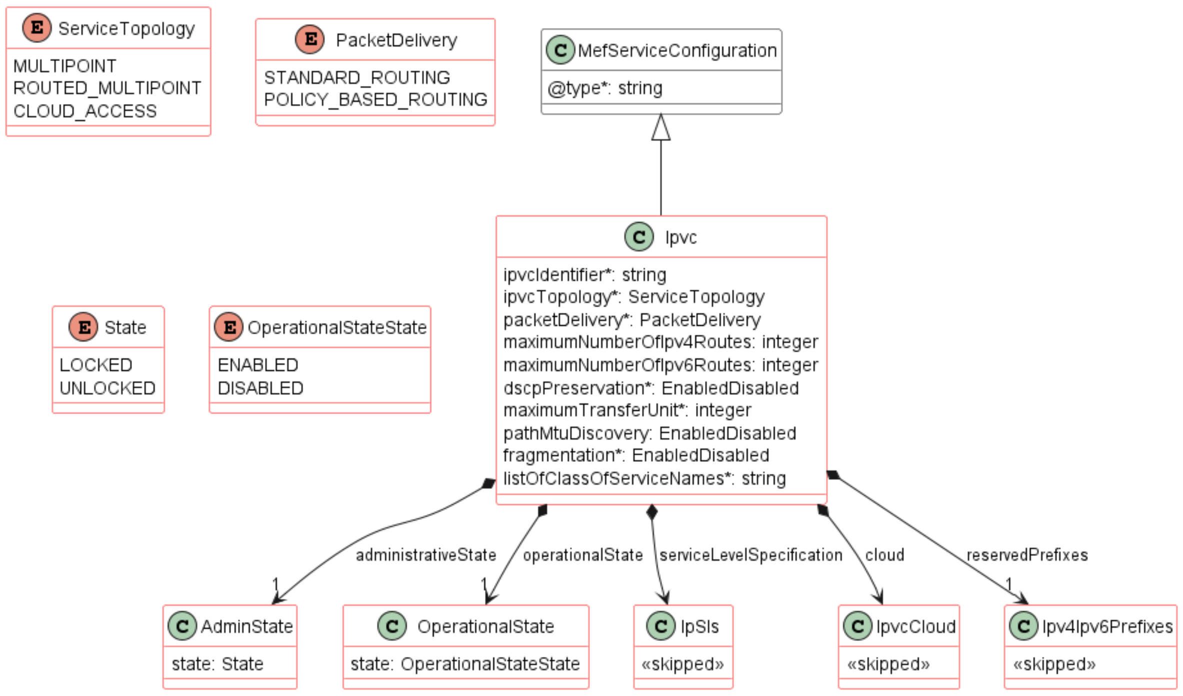


Figure 7. A simplified view of IPVC service specification data model

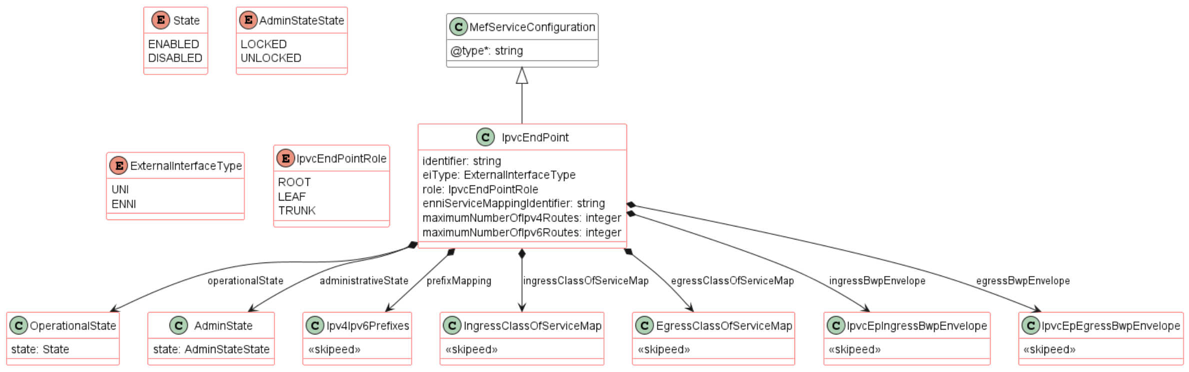


Figure 8. A simplified view of IPVC End Point service specification data model

Service specifications define several service-related and envelope-related requirements. For example:

- for an IPVC End Point service two mandatory relationships must be specified, one toward the IPVC (`IPUNI_ENDPOINT_OF_IPVC`), and a second towards the IP UNI (`CONNECTS_TO_IPUNI`) for the `add` action.

- in the case of a `modify` action, service relationships must have the same value as in the `add` action. They must not be changed
- for an IP UNI Access Link Trunk service a place relationship (`INSTALL_LOCATION`) must be specified
- in the case of a `modify` action, place relationships must have the same value as in the `add` action. They must not be changed

In case, some of these requirements are violated the SOF returns an error response to the BUS that indicates specific functional errors. These errors are listed in the response body (a list of `Error422` entries) for HTTP `422` response.

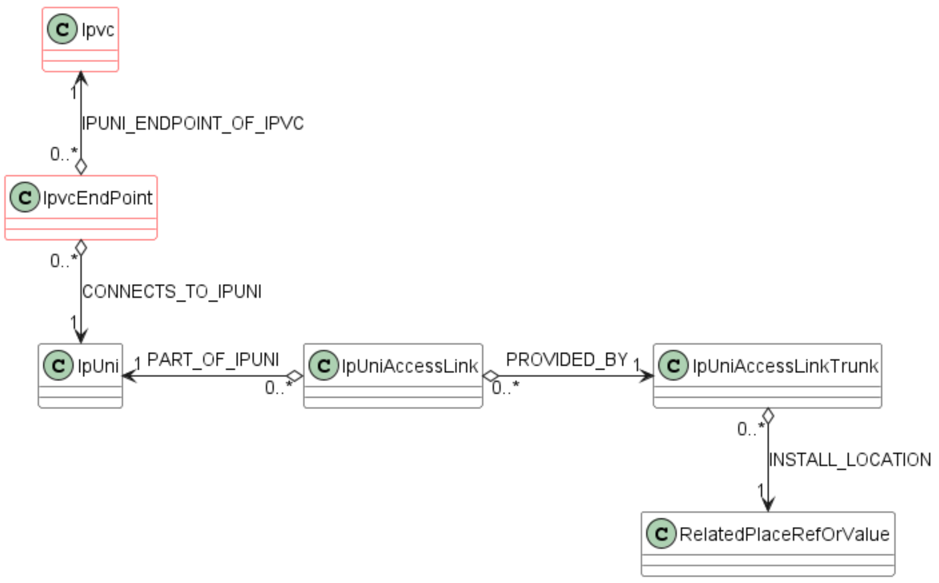


Figure 9. Example use case configuration

Figure 9 shows a setup of service configuration used by the example. The Advanced Internet Access is built from 5 services:

- IPVC
- IPVC End Point
- IP UNI
- IP UNI Access Link
- IP UNI Access Link Trunk

The example assumes a situation, where IP UNI, IP UNI Access Link, and IP UNI Access Link Trunk are already provisioned and are available in Service Inventory. They are marked with black lines. The Service Order includes requests to create 2 services: IPVC and IPVC End Point (marked with red lines). This means there are 2 Service Order Items with `action=add`. As mentioned earlier, there are 2 mandatory relations to be provided with IPVC End Point. In this case:

- `IPUNI_ENDPOINT_OF_IPVC` is provided with the use of `serviceOrderItemRelationship` as pointing to the `Ipvc` being part of the same Service Order,

- `CONNECTS_TO_IPUNI` is provided with the use of `serviceRelationship` as pointing to an `IpUni` service that is already provisioned and available in Service Inventory.

5.5. Model structure and validation

The structure of the payloads exchanged via Legato Service API endpoints is defined using:

- OpenAPI version 3.0 for the service-agnostic part of the payload
- JsonSchema (draft 7) for the service-specific part of the payload

[R6] Implementations **MUST** use payloads that conform to these definitions.

[R7] A service specification may define additional consistency rules and requirements that **MUST** be respected by implementations. These are defined for:

- required relation type, multiplicity to other items within the same or another Service Order request
- required relation type, multiplicity to entities in the SOF's service inventory
- related contact information roles that are to be defined at the Service Order Item level
- relations to places (locations) and their roles that are to be defined at the order item level

5.6. Security Considerations

Although the Legato IRP is internal to a Service Provider/Operator business boundary, it is expected that some minimal security mechanisms are in place for any communication over this IRP. There must also be authorization mechanisms in place to control what a particular BUS or SOF is allowed to do and what information may be obtained. However, the definition of the exact security mechanism and configuration is outside the scope of this document. The LSO Security mechanisms are defined by MEF 128 *LSO API Security Profiles* [MEF128].

6. API Interactions and Flows

This section provides a detailed insight into the API functionality, use cases, and flows. It starts with Table 6 presenting a list and short description of all business use cases then presents the variants of end-to-end interaction flows, and in the following subchapters describes the API usage flow and examples for each of the use cases.

Use Case #	**Use Case Name**	**Use Case Description**
1	Create Service Order	A request initiated by the BUS to order a new service or service component(s). A Service Order must contain at least one Service Order Item (Use Case # 1-a, 1-b, or 1-c) as shown below. A Service Order may contain more than one Service Order Item and Service Order Items within a Service Order are not required to have relationships between them.
1-a	Service Order Item to Add Service	Service Order Item adds a new Service.
1-b	Service Order Item to Modify Existing Service	Service Order Item modifies attributes of a specific active Service.
1-c	Service Order Item to Delete Existing Service	Service Order Item disconnects an active Service.
2	Retrieve List of Service Orders	A request initiated by the BUS to retrieve a list of Service Orders that match the provided filter criteria
3	Retrieve Service Order by Service Order Identifier	A request initiated by the BUS to retrieve the details associated with a specific Service Order with the given Service Order Identifier.

Use Case #	Use Case Name	Use Case Description
4	Register for Notifications	The BUS requests to subscribe to notifications.
5	Send Notification	A notification initiated by the SOF to the BUS

Table 6. Use cases description

6.1. Use case 1: Create Service Order

This is the initial step for Service Order processing.

6.1.1. Interaction flow

The flow of this use case is very simple and is described in Figure 10.

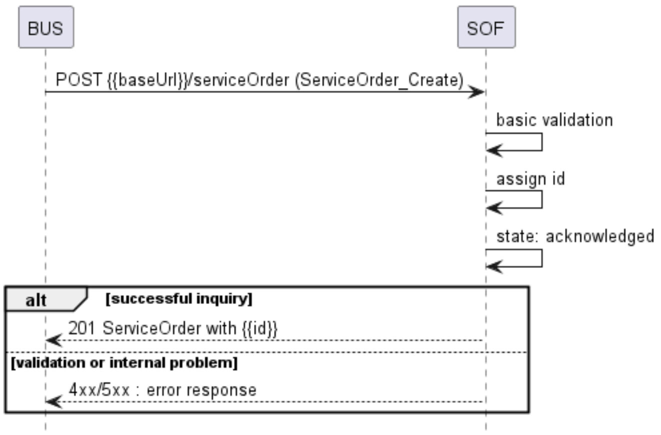

BUS
SOF
POST {{baseUrl}}/serviceOrder (ServiceOrder_Create)
basic validation
assign id
state: acknowledged
alt
[successful inquiry]
201 ServiceOrder with {{id}}
[validation or internal problem]
4xx/5xx : error response

Figure 10. Use Case 1 - Service Order create request flow

The BUS sends a request with a `ServiceOrder_Create` type in the body. The SOF performs request validation, assigns an `id`, and returns `ServiceOrder` type in the response body, with a `state` set to `acknowledged`. From this point, the Service Order is ready for further processing. The BUS can track the progress of the process either by subscribing for notifications or by periodically polling the `ServiceOrder`. The two patterns are presented in the following two diagrams.

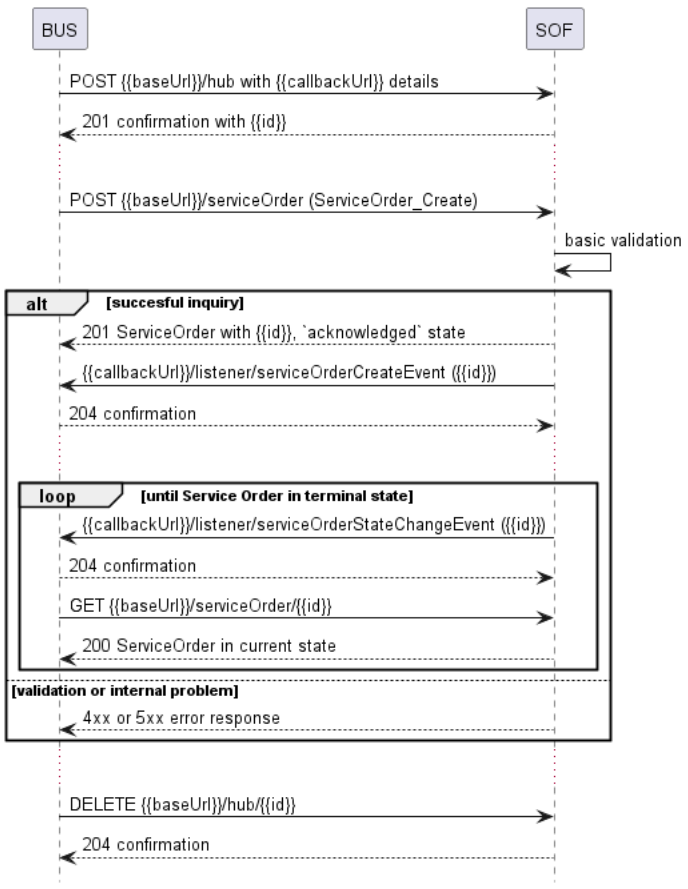

BUS
SOF
POST {{baseUrl}}/hub with {{callbackUrl}} details
201 confirmation with {{id}}
POST {{baseUrl}}/serviceOrder (ServiceOrder_Create)
basic validation
alt
[succesful inquiry]
201 ServiceOrder with {{id}}, `acknowledged` state
{{callbackUrl}}/listener/serviceOrderCreateEvent ({{id}})
204 confirmation
loop
[until Service Order in terminal state]
{{callbackUrl}}/listener/serviceOrderStateChangeEvent ({{id}})
204 confirmation
GET {{baseUrl}}/serviceOrder/{{id}}
200 ServiceOrder in current state
[validation or internal problem]
4xx or 5xx error response
DELETE {{baseUrl}}/hub/{{id}}
204 confirmation

Figure 11. Service Order progress tracking - Notifications

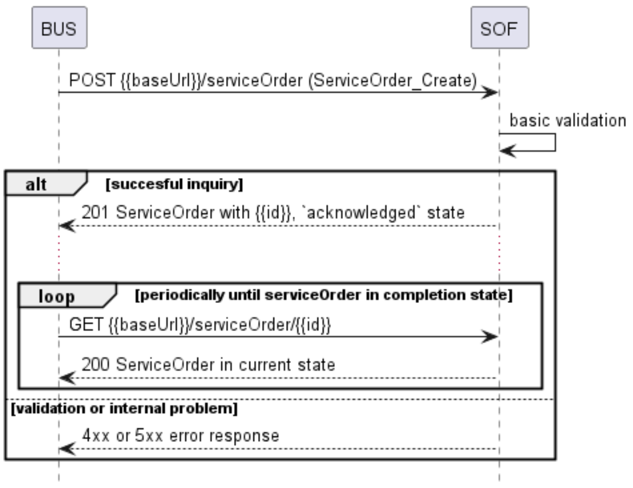

BUS
SOF
POST {{baseUrl}}/serviceOrder (ServiceOrder_Create)
basic validation
alt
[succesful inquiry]
201 ServiceOrder with {{id}}, `acknowledged` state
loop
[periodically until serviceOrder in completion state]
GET {{baseUrl}}/serviceOrder/{{id}}
200 ServiceOrder in current state
[validation or internal problem]
4xx or 5xx error response

Figure 12. Service Order progress tracking - Polling

Note: The context of notifications is not a part of the considered use case itself. It is presented to show the big picture of end-to-end flow. This applies also to all further use case flow diagrams with notifications.

so to all further use case flow diagrams with notifications.

6.1.2. Create Service Order Request

Figure 13 presents the most important part of the data model used during the Create Service Order request (`POST /serviceOrder`) and response. The model of the request message -

`ServiceOrder_Create` is a subset of the `ServiceOrder` model and contains only attributes that can (or must) be set by the BUS. The SOF then enriches the entity in the response with additional information.

Note: `ServiceOrder_Create` and `ServiceOrderItem_Create` are entities used by the BUS to make a request. `ServiceOrder` and `ServiceOrderItem` are entities used by the SOF to provide a response. The request entities have a subset of attributes of the response entities. Thus for visibility of these shared attributes `ServiceOrder_Common` and `ServiceOrderItem_Common` have been introduced. Though, these are not to be used directly in the exchange.

A `ServiceOrderItem_Create` defines details of the service(s) being subject of the ordering (in `ServiceValue` structure) and allows for the definition of additional information like related parties (`RelatedContactInformation`) or relations to other items (`ServiceOrderItemRelationship`, `ServiceOrderRelationship`).

`ServiceValue` allows for the introduction of service-specific properties as the Service Order payload. The extension mechanism is described in detail in Section 5.3. `ServiceValue` may be also used to specify relations to places (using specializations of `RelatedPlaceOrValue`, as described in Section 6.1.8.) and/or to a service that exists in the SOF's inventory (using `ServiceRelationship`).

The full list of attributes is available in Section 7 and in the API specification which is an integral part of this standard.

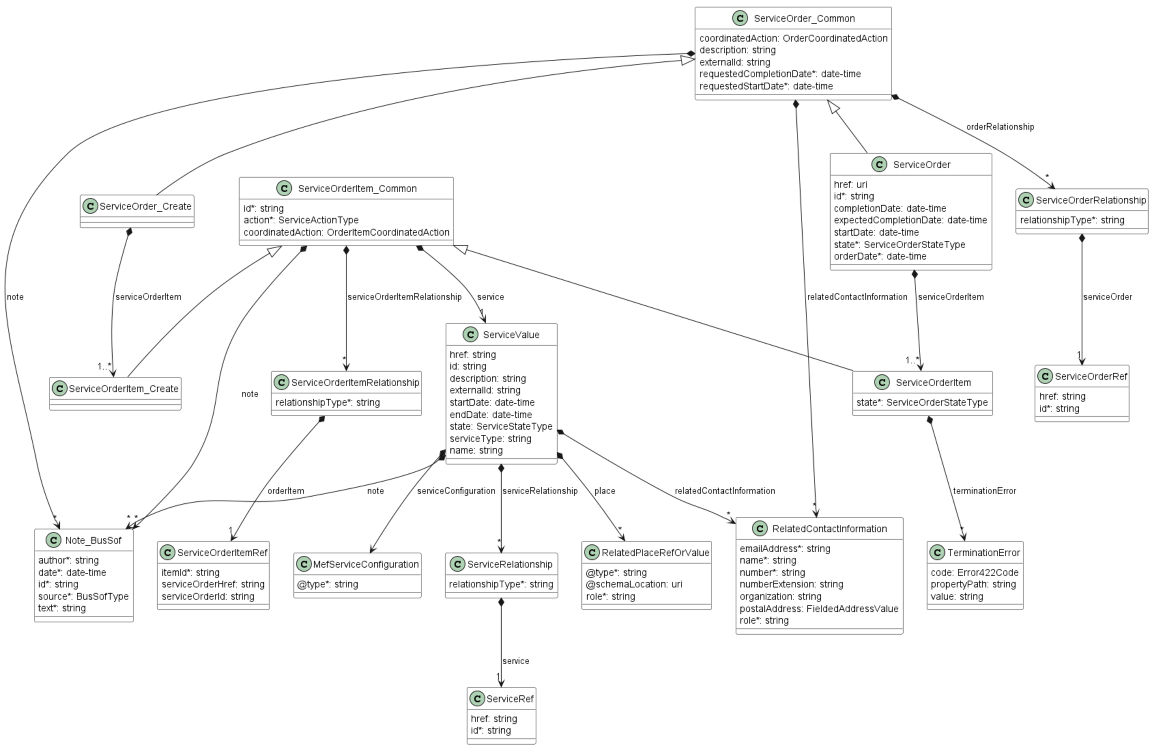


Figure 13. Service Order Key Entities

To send a Service Order request the BUS uses the `createServiceOrder` operation from the API: `POST /serviceOrder`. For clarity, some of the Service Order payload's attributes might be omitted to improve examples' readability. The `ServiceOrder_Create` is a simple structure that is common for all types of requests (`add`, `modify`, `delete`), most of the information is in the `ServiceOrderItem_Create`.

`Service Order` Create Request

```
{
  "description": "Example Service Order",
  "externalId": "busOrder-101",
  "requestedCompletionDate": "2023-01-28T20:45:23.796Z",
  "requestedStartDate": "2023-01-02T00:00:00.000Z",
  "relatedContactInformation": [
    {
      "emailAddress": "john.example@example.com",
      "name": "John Example",
      "number": "12-345-6789",
      "numberExtension": "1234",
      "organization": "Example Co.",
      "role": "serviceOrderContact"
    }
  ],
  "note": [
    {
      "author": "John Example",
      "date": "2022-12-28T20:45:23.796Z",
      "id": "note-001",
      "source": "bus",
      "text": "This is an example text"
    }
  ],
  "serviceOrderItem": [
    {
      "id": "item-001",
      "action": "add",
      "service": {
        "description": "IP Virtual Connection",
        "externalId": "BUS_IPVC-0001",
        "serviceType": "Internet Access",
        "name": "IPVC",
        "state": "feasibilityChecked",
        "relatedContactInformation": [
          {
            "emailAddress": "BUS.ServiceOrderItemContact@example.com",
            "name": "BUS Service Order Item Contact",
            "number": "+12-345-678-90",
            "role": "busServiceOrderItemContact"
          }
        ],
        "serviceConfiguration": {
          "@type": "urn:mef:lso:spec:legato:ipvc:v0.0.1:all",
          "administrativeState": {
            "state": "UNLOCKED"
          },
          "operationalState": {
            "state": "ENABLED"
          },
          "ipvcIdentifier": "IPVC-0000-0001",
          "ipvcTopology": "CLOUD_ACCESS",
          "packetDelivery": "STANDARD_ROUTING",
          "maximumNumberOfIpv4Routes": 1,
          "maximumNumberOfIpv6Routes": 0,
          "dscpPreservation": "ENABLED",
          "serviceLevelSpecification": {}, <<skipped>>
          "maximumTransferUnit": 1522,
          "pathMtuDiscovery": "ENABLED",
          "fragmentation": "DISABLED",
          "cloud": {}, <<skipped>>
          "reservedPrefixes": {}, <<skipped>>
          "listOfClassOfServiceNames": ["low"]
        }
      }
    },
    {
      "id": "item-002",
      "action": "add",
      "serviceOrderItemRelationship": [
        {
          "orderItem": {  << Relationship to IPVC in the same Service Order >>
            "itemId": "item-001"
          },
          "relationshipType": "IPUNI_ENDPOINT_OF_IPVC"
        }
      ],
      "service": {
        "description": "IPVC End Point",
        "externalId": "BUS_IPVC_END_POINT-0001",
        "serviceType": "Internet Access",
        "name": "IPVCEndpoint",
        "serviceRelationship": [
          { << Relationship to already configured IP UNI in Service Inventory >>
            "relationshipType": "CONNECTS_TO_IPUNI",
            "service": {
              "id": "IP_UNI_0000-0001"
            }
          }
        ],
        "relatedContactInformation": [
          {
            "emailAddress": "BUS.ServiceOrderItemContact@example.com",
            "name": "BUS Service Order Item Contact",
            "number": "+12-345-678-90",
            "role": "busServiceOrderItemContact"
          }
        ],
        "serviceConfiguration": {
          "@type": "urn:mef:lso:spec:legato:ipvc-end-point:v0.0.1:all",
          "administrativeState": {
            "state": "UNLOCKED"
          },
          "operationalState": {
            "state": "ENABLED"
          },
          "identifier": "IPVC-EndPoint-0000-0001",
          "eiType": "UNI",
          "role": "ROOT",
          "prefixMapping": {},
          "maximumNumberOfIpv4Routes": 1,
          "maximumNumberOfIpv6Routes": 0,
          "ingressClassOfServiceMap": {}, <<skipped>>
          "egressClassOfServiceMap": {}, <<skipped>>
          "ingressBwpEnvelope": {}, <<skipped>>
          "egressBwpEnvelope": {} <<skipped>>
        }
      }
    }
  ]
}
```

[R8] The BUS's request **MUST** contain `requestedStartDate`, `requestedCompletionDate` and at least one `serviceOrderItem`.

[R9] The BUS's request **MUST** contain at least one `serviceOrderItem`.

[D1] The BUS and SOF **SHOULD** agree on using specific contact `roles`.

Note: During the onboarding the SOF may require to provide an additional contact `role`.

Note: It is up to SOF's discretion on how to react in case the BUS provides a contact `role` that is not agreed upon during the onboarding. Preferably the SOF should return an error with a message stating which `roles` are accepted. It may also be ignored

For each `serviceOrderItem`:

[R10] The BUS's Create Service Order request **MUST** contain:

- `id`
- `action`
- `service`

[R11] When adding a note, BUS **MUST** add a `note` only with `source=bus`.

6.1.3. Create Service Order Response

Entities use for providing a response to Create Service Order request are presented in Figure 13. The main types used for response are `ServiceOrder` and `ServiceOrderItem`, which add attributes set by SOF (like `id` or `state`) `ServiceOrder` is the root entity of a response. The response echoes back all attributes as provided by the BUS and contains the same number of `ServiceOrderItems` as in the request.

The following snippet presents the SOF's response.

`Service Order` Create Response

```
{
  "id": "00000000-3333-4444-5555-000000004567", << added by SOF >>
  "href": "{{baseUrl}}/serviceOrder/00000000-3333-4444-5555-000000004567", << added by SOF >>
  "state": "acknowledged", << added by SOF >>
  "orderDate": "2022-12-28T20:45:24.796Z", << added by SOF >>
  "expectedCompletionDate": "2023-01-25T20:00:00.000Z", << added by SOF >>
  "description": "Example Service Order",
  "externalId": "busOrder-101",
  "requestedCompletionDate": "2023-01-28T20:45:23.796Z",
  "requestedStartDate": "2023-01-02T00:00:00.000Z",
  "relatedContactInformation": [
    {
      "emailAddress": "john.example@example.com",
      "name": "John Example",
      "number": "12-345-6789",
      "numberExtension": "1234",
      "organization": "Example Co.",
      "role": "serviceOrderContact"
    },
    { << added by SOF >>
      "emailAddress": "ella.sof@example.com",
      "name": "Ella SOF",
      "number": "98-765-4321",
      "organization": "SOF Co.",
      "role": "sofContact"
    }
  ],
  "note": [
    {
      "id": "note-001",
      "author": "John Example",
      "date": "2022-12-28T20:45:23.796Z",
      "source": "bus",
      "text": "This is an example text"
    },
    { << added by SOF >>
      "id": "note-002",
      "author": "Ella SOF",
      "date": "2022-12-28T20:45:24.796Z",
      "source": "sof",
      "text": "This is an example response text"
    }
  ],
  "serviceOrderItem": [
    {
      "id": "item-001",
      "action": "add",
      "state": "acknowledged", << added by SOF >>
```

```
        "service": {
          "id": "00000000-5555-6666-7777-000000008888", << added by SOF >>
          "href": "{{baseUrl}}/service/00000000-5555-6666-7777-000000008888", << added by SOF >>
          "state": "feasibilityChecked",
          "description": "IP Virtual Connection",
          "externalId": "BUS_IPVC-0001",
          "serviceType": "Internet Access",
          "name": "IPVC"
          ...
          << skipped, as provided by BUS >>
        }
      },
      {
        "id": "item-002",
        "action": "add",
        "state": "acknowledged", << added by SOF >>
        "serviceOrderItemRelationship": [
          {
            "orderItem": {
              "itemId": "item-001",
              "serviceOrderHref": "string",
              "serviceOrderId": "string"
            },
            "relationshipType": "IPUNI_ENDPOINT_OF_IPVC"
          }
        ],
        "service": {
          "id": "00000000-5555-6666-7777-000000009999", << added by SOF >>
          "href": "{{baseUrl}}/service/00000000-5555-6666-7777-000000009999", << added by SOF >>
          "state": "feasibilityChecked",
          "description": "IPVC End Point",
          "externalId": "BUS_IPVC_END_POINT-0001",
          "serviceType": "Internet Access",
          "name": "IPVCEndpoint",
          "serviceRelationship": [
            {
              "relationshipType": "CONNECTS_TO_IPUNI",
              "service": {
                "id": "IP_UNI_0000-0001"
              }
            }
          ]
          ...
          << skipped, as provided by BUS >>

        }
      }
    ]
  }
```

Attributes that are set by the SOF in the response are marked with the `<< added by SOF >>` tag. The response to the create request does not contain all possible attributes. Some of them are valid only in the future lifecycle of the `ServiceOrder` (e.g. `completionDate`, `startDate`).

[R12] The SOF's response **MUST** include all and unchanged attributes' values as provided by BUS in the request.

The SOF might append related contact information or notes if required, but cannot modify items set by the BUS.

[R13] The SOF **MUST** specify the following attributes in a response:

- `id`
- `state`
- `orderDate`

[R14] The `id` **MUST** remain the same value for the life of the Service Order.

[R15] When adding a note, SOF **MUST** add a `note` only with `source=sof`.

[R16] Notes **MUST NOT** be modified or deleted once entered.

For each `serviceOrderItem`:

[R17] The response **MUST** have the `state` attribute set.

[R18] If the Service Order Item `state` in the SOF's response is not `completed`, the response **MUST NOT** contain the `expectedCompletionDate`.

6.1.4. Use Case 1a: Service Order Item to Add Service

When requesting a new service installation (`action` equal to `add`) the BUS needs to provide all of its configuration information. The example for `add` action is already provided in the snippets above.

The following requirements apply when `serviceOrderItem.action` is `add`:

[R19] The BUS **MUST** provide:

- `service.state`
- `service.serviceConfiguration`

[R20] If there is a relationship with a Service Order Item within the same Service Order, the `serviceOrderItemRelationship.itemId` **MUST** be specified.

[R21] If there is a relationship with a Service Order Item within the same Service Order, the `serviceOrderItemRelationship.itemId` and `serviceOrderItemRelationship.serviceOrderId` **MUST NOT** be specified.

[R22] If there is a relationship with a Service Order Item of another Service Order, the `serviceOrderItemRelationship.itemId` and `serviceOrderItemRelationship.serviceOrderId` **MUST** be specified.

[R23] The BUS **MUST NOT** specify the `serviceOrderItem.service.id` in the request. It is the SOF who assigns this id.

Note: The service.id might not be assigned yet at the moment the SOF provides a response for the Create Service Order Request.

6.1.5. Use case 1b: Service Order Item to Modify Existing Service

The following example shows a request for an order for an existing IPVC End Point Service modification (`action` equal to `modify`). In particular, a change to `maximumNumberOfIpv4Routes` is introduced.
The IPVC End Point service exists in SOF's inventory and is identified as `00000000-5555-6666-7777-000000009999`, as provided in SOF response presented in Chapter 6.1.3.

The following requirements apply to `serviceOrderItem` when `action` is `modify`:

[R24] The modify request **MUST** specify a reference (provide `service.id`) to an existing service that is a subject of this order and provide the desired `service.serviceConfiguration`.

[R25] The modify request **MUST** provide:

- `service.id` - a reference to an existing service that is a subject of this order
- `service.state`
- `service.serviceConfiguration`

[R26] The modify request **MUST** repeat the same values (specified or empty) of `service.serviceRelationship`, and `service.place` as they are available in the inventory for a given service instance. These values cannot be updated or deleted.

[R27] If there is a relationship with another Service Order Item, the `serviceOrderItemRelationship` **MUST** be also specified unchanged.

There is no possibility to send an update to single attributes. The BUS must send a full service description (the whole `service.serviceConfiguration` section and if set previously or to be set: `service.serviceRelationship` and `service.place`), which means all attributes that represent the desired state, even if some of them do not change.
If SOF does not allow for some of the attributes to change an appropriate error response (`422`) must be returned to the BUS.

Please also note, that in the `add` case, a reference to the IPVC service used the `serviceOrderItemRelationship` pointing to another `serviceOrderItem` in the same Service Order Request. This is because the IPVC did not exist at that moment and was also a part of the order. In the case of ordering the update of an existing IPVC End Point, the IPVC is also existing and it must be referenced with the use of `serviceRelationship`. This example assumes that the IPVC service is available in SOF's Inventory with the `id` equals `"00000000-5555-6666-7777-000000008888"` (as provided in SOF response presented in Chapter 6.1.3.

Service Order Item to Modify Existing Service

```
{
  "description": "Example Service Order to Modify IPVC End Point service",
  "externalId": "busOrder-102",
  "requestedCompletionDate": "2023-02-03T20:45:23.796Z",
  "requestedStartDate": "2023-02-02T00:00:00.000Z",
  "relatedContactInformation": [
    {
      "emailAddress": "john.example@example.com",
      "name": "John Example",
      "number": "12-345-6789",
      "numberExtension": "1234",
      "organization": "Example Co.",
      "role": "serviceOrderContact"
    }
  ],
  "serviceOrderItem": [
    {
      "id": "item-001",
      "action": "modify",
      "service": {
        "id": "00000000-5555-6666-7777-000000009999", << id to point to service instance >>
        "description": "IPVC End Point",
        "externalId": "BUS_IPVC_END_POINT-0001",
        "serviceType": "Internet Access",
        "name": "IPVCEndpoint",
        "state": "active",
        "serviceRelationship": [
          { << relation to IP UNI - not changed >>
            "relationshipType": "CONNECTS_TO_IPUNI",
            "service": {
              "id": "IP_UNI_0000-0001"
            }
          },
          { << relation to IPVC - not changed, but provided with serviceRelationship instead of
serviceOrderItemRelationship >>
            "relationshipType": "IPUNI_ENDPOINT_OF_IPVC",
            "service": {
              "id": "00000000-5555-6666-7777-000000008888"
            }
          }
        ],
        "serviceConfiguration": {
          "@type": "urn:mef:lso:spec:legato:ipvc-end-point:v0.0.1:all",
          "administrativeState": {
            "state": "UNLOCKED"
          },
          "operationalState": {
            "state": "ENABLED"
          },
          "identifier": "IPVC-EndPoint-0000-0001",
          "eiType": "UNI",
          "role": "ROOT",
          "prefixMapping": {},
          "maximumNumberOfIpv4Routes": 2, << modified value >>
          "maximumNumberOfIpv6Routes": 0,
          "ingressClassOfServiceMap": {},
          "egressClassOfServiceMap": {},
          "ingressBwpEnvelope": {},
          "egressBwpEnvelope": {}
        }
      }
    }
  ]
}
```

6.1.6. Use case 1c: Service Order Item to Delete Existing Service

The example below represents a single Service Order request for deletion (`action=delete`) of an existing IPVC End Point service.

Service Order to Delete Existing Service

```
{
  "description": "Example Service Order to Delete IPVC End Point service",
  "externalId": "busOrder-103",
  "requestedCompletionDate": "2023-03-03T20:45:23.796Z",
  "requestedStartDate": "2023-03-02T00:00:00.000Z",
  "serviceOrderItem": [
    {
      "id": "item-001",
      "action": "delete",
      "service": {
        "id": "00000000-5555-6666-7777-000000009999" << id to point to service instance >>
      }
    }
  ]
}
```

The following requirements apply to `serviceOrderItem` when `action` is `delete`:

[R28] `service.id` **MUST** be provided.

[R29] The BUS **MUST NOT** provide any `service` attributes other than `service.id`.

6.1.7. Service Order and Service Order Items State Machine

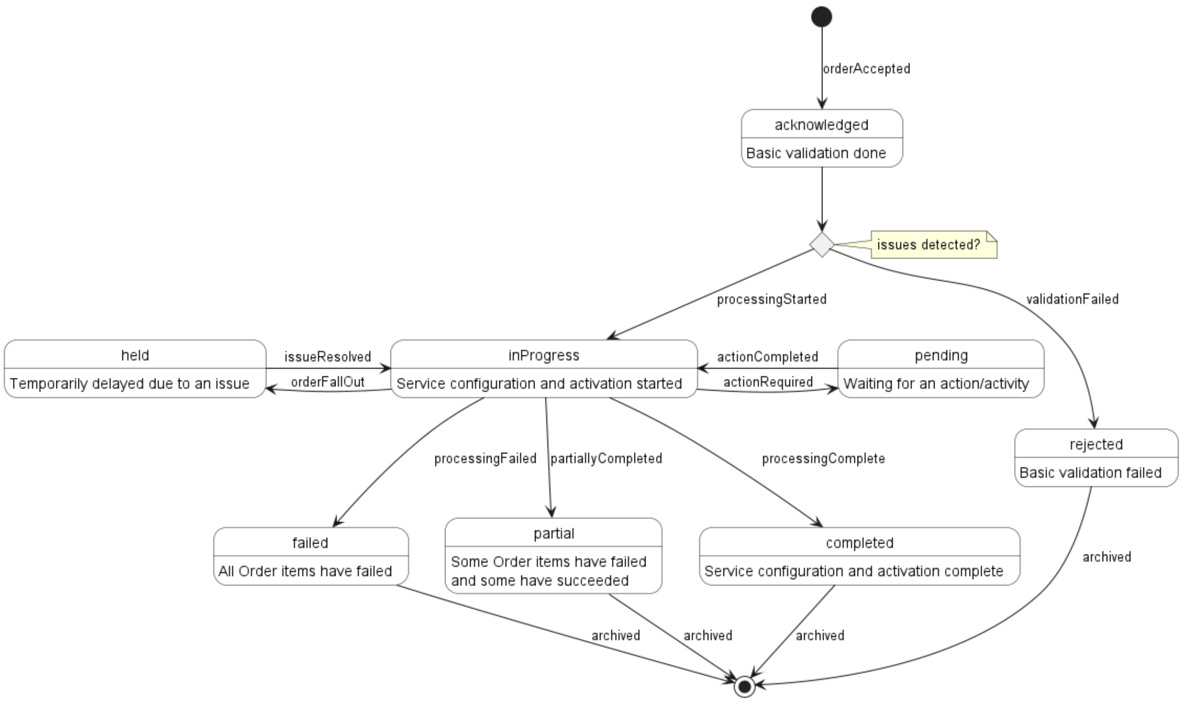


Figure 14. Service Order and Service Order Items State Machine

Service Order and Service Order Item share the same list of possible states and states' transitions. They are presented in Figure 14.

After receiving the request, the SOF performs basic checks of the message. If any problem is found an Error response is provided. If the validation passes a response is provided with `ServiceOrder` and all `ServiceOrderItems` in the `acknowledged` state. Before moving the order to the `inProgress` state, the BUS performs all the remaining business and time-consuming validations. At this point, an Error response cannot be provided anymore so the order moves to a `rejected` state if some issues are found. The `serviceOrderItem.terminationError` acts as a placeholder to provide a detailed description of what caused the problem.

Table 7 presents the states' descriptions.

State	Description
acknowledged	A `ServiceOrder` request has been received and has passed message and basic validations and a *Success Response* has been sent.

State	Description
rejected	This state indicates that: - Invalid information is provided through the `ServiceOrder` / `ServiceOrderItem` request - The request fails to meet validation rules for `Service` delivery (processing) If one `ServiceOrderItem` is rejected, then the entire `ServiceOrder` request is rejected and a *Error Response* is sent.
inProgress	This state indicates that all `ServiceOrderItems` have successfully passed the validations checks and the scheduled `Service` delivery/processing has started. The `ServiceOrder` will be in *`inProgress`* state if *at least one* `ServiceOrderItem` is in *`inProgress`* state
pending	This state indicates that a `ServiceOrderItem` is currently in a waiting stage for an action/activity to be completed before the order-processing can progress further, pending order amend or cancel assessment. A *`pending`* state can lead into auto cancellation of an `ServiceOrderItem`, if no action is taken within the agreed timeframe. The `ServiceOrder` will be in *`pending`* state if *at least one* `ServiceOrderItem` is in *`pending`* state
held	This state indicates that a `ServiceOrderItem` cannot be progressed due to an issue. The `Service` delivery (processing) has been temporarily delayed to resolve an infrastructure shortfall to facilitate the supply of order. Upon resolution of the issue, the `ServiceOrderItem` will continue to progress. A *`held`* state can lead into auto cancellation of a `ServiceOrderItem` if no action is taken within the agreed timeframe. The `ServiceOrder` will be in *`held`* state if at least one `ServiceOrderItem` is in *`held`* state
failed	This state indicates that `Service` delivery (processing) associated with a `ServiceOrderItem` has failed. This indicates an irrecoverable error as opposed to *`held`* or *`pending`* issues. The `ServiceOrder` will be in *`failed`* state if at *ALL* `ServiceOrderItems` are in *`failed`* state
completed	This state indicates that `Service` delivery (processing) associated with a `ServiceOrderItem` has completed. The `ServiceOrder` will be in *`completed`* state if at *ALL* `ServiceOrderItems` are in *`completed`* state
partial	This state indicates that some `ServiceOrderItem` are in *`completed`* state while others are in *`cancelled`* and/or *`failed`* states, so the entire `ServiceOrder` is in a *`partial`* state.

Table 7. Service Order and Service Order Item states

6.1.8. Specifying Place Details

Some service specifications may define requirements concerning place definition in case `add` or `modify` action is used. For example, an IP UNI Access Link Trunk service specification requires an `INSTALL_LOCATION` place definition.

There are different formats in which place information may be provided: `MEFGeographicPoint`, `FieldedAddress`, `FormattedAddress`, `GeographicAddressLabel`, `GeographicSiteRef`, `GeographicAddressRef`. The first four of them can be used to provide place description by value. The site and address reference allow specifying the place information as a reference to previously validated address or site available through SOF's Addressing and Site API endpoints, which definition is provided in the SDK:

- `productApi/serviceability/address/geographicAddressManagement.api.yaml`
- `productApi/serviceability/site/geographicSiteManagement.api.yaml`

The Address Validation and Site APIs are standardized by:

- Address, Service Site, and Product Offering Qualification Management, Requirements and Use Cases MEF 79
- Amendment to MEF 79: Address, Service Site, and Product Offering Qualification Management, Requirements, and Use Cases MEF 79.0.1
- Amendment to MEF 79: Address Validation MEF 79.0.2
- LSO Cantata and LSO Sonata Address Management API - Developer Guide MEF 121
- LSO Cantata and LSO Sonata Site Management API - Developer Guide MEF 122

The superclass for all address types is the `RelatedPlaceRefOrValue` which adds the `role` to add more context to the specified address. To distinguish between place types the `@type` discriminator is used.

Note: The *RefOrValue* stands for a pattern where an address can be provided either by `id` (using `GeographicSiteRef` or `GeographicAddressRef`) OR by value (with use of `MEFGeographicPoint`, `FieldedAddress`, `FormattedAddress`, `GeographicAddressLabel`). There is no way to specify an address with use both ref AND value at the same time.

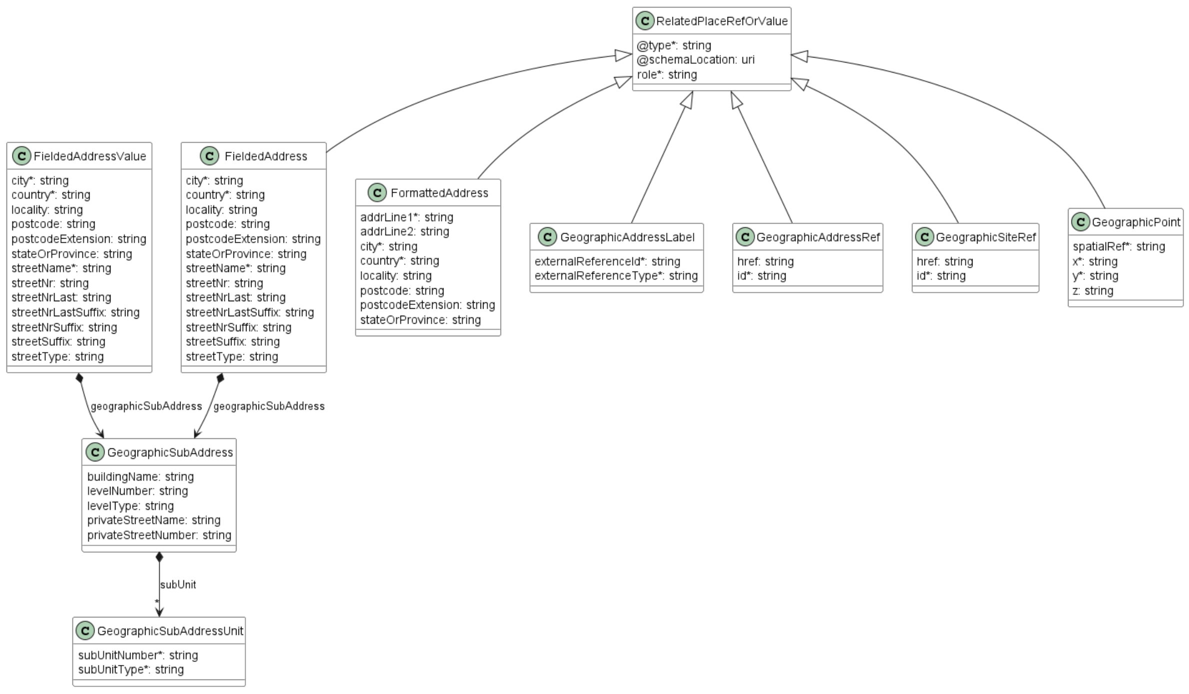


Figure 15. The data model for place representation

Examples of different place specification formats are provided below.

6.1.8.1. Fielded Address

```
{
  "@type": "FieldedAddress",
  "streetType": "ul.",
  "streetName": "Edmunda Wasilewskiego",
  "streetNr": "20",
  "streetNrSuffix": "14",
  "city": "Kraków",
  "stateOrProvince": "Lesser Poland",
  "postcode": "30-305",
  "country": "Poland",
  "geographicSubAddress": {
    "levelType": "floor",
    "levelNumber": "4"
  },
  "role": "INSTALL_LOCATION"
}
```

Fielded address example of a place specification. The type discriminator has the value `FieldedAddress`. A subset of available attributes is used to describe the place. The fielded address has an optional `geographicSubAddress` structure that defines several attributes that can be used in case precise address information has to be provided. In the example above, a floor in the building at the given address is specified using this structure. The role of the place is assigned according to the requirements of the Operator UNI service specification.

6.1.8.2. Formatted Address

```
{
  "@type": "FormattedAddress",
  "addrLine1": "ul. Edmunda Wasilewskiego 20/14",
  "addrLine2": "Floor 4",
  "city": "Kraków",
  "stateOrProvince": "Lesser Poland",
  "postcode": "30-305",
  "country": "Poland",
  "role": "INSTALL_LOCATION"
}
```

Place information in a form of a formatted address. The type discriminator has the value `FormattedAddress`. This example contains the same information as the previous `FieldedAddress` example.

6.1.8.3. Geographic Point

```
  {
    "@type": "MEFGeographicPoint",
    "spatialRef": "EPSG:4326 WGS 84",
    "x": "50.048868",
    "y": "19.929523",
    "role": "INSTALL_LOCATION"
  }
```

Place information in a form of a geographic point. `spatialRef` determines the standard that has to be used to interpret coordinates provided in the required `x` (latitude), `y` (longitude), and optional `z` (elevation) values.

This type allows only providing a point. It cannot carry more detailed information like the floor number from previous examples.

[R30] The `spatialRef` value that can be used **MUST** be agreed between BUS and SOF.

6.1.8.4. Geographic Address Label

```
  {
    "@type": "GeographicAddressLabel",
    "externalReferenceType": "CLLI",
    "externalReferenceId": "PLTXCL01",
    "role": "INSTALL_LOCATION"
  }
```

The Geographic Address Label represents a unique identifier controlled by a generally accepted independent administrative authority that specifies a fixed geographical location. The example above is a place that represents a CLLI (Common Language Location Identifier) identifier which is commonly used to refer locations in North America for network equipment installations.

6.1.8.5. Geographic Site Reference

```
  {
    "@type": "GeographicSiteRef",
    "id": "18d3bb74-997a-4a62-8198-84250766765a",
    "role": "INSTALL_LOCATION"
  }
```

`GeographicSiteRef` type is used to specify a `GeographicSite` by reference in the request. In the above example, a `GeographicSite` identified as `18d3bb74-997a-4a62-8198-84250766765a` in the SOFs Service Site API is used.

6.1.8.6. Geographic Address Reference

```
  {
    "@type": "GeographicAddressRef",
    "id": "8198bb74-18d3-9ef0-4913-66765a842507",
    "role": "INSTALL_LOCATION"
  }
```

`GeographicAddressRef` type is used to specify a `GeographicAddress` by reference in the request. In the above example, a `GeographicAddress` identified as `8198bb74-18d3-9ef0-4913-66765a842507` in the SOFs Service Site API is used.

6.2. Use Case 2: Retrieve List of Service Orders

The BUS can retrieve a list of `ServiceOrders` by using a `GET /serviceOrder` operation with desired filtering criteria.

[O3] The BUS's request **MAY** contain none or more of the following attributes:

- `state`
- `orderDate.gt`
- `orderDate.lt`
- `completionDate.gt`
- `completionDate.lt`
- `expectedCompletionDate.gt`
- `expectedCompletionDate.lt`
- `startDate.gt`
- `startDate.lt`

A response to retrieve a list of results can be paginated. The BUS can specify following query attributes related to pagination:

- `limit` - number of expected list items
- `offset` - offset of the first element in the result list

The filtering and pagination attributes must be specified in URI query format RFC3986. The SOF returns a list of elements that comply with the requested `limit`. If the requested `limit` is higher than the supported list size the smaller list result is returned. In that case, the size of the result is returned in the header attribute `X-Result-Count`. The SOF can indicate that there are additional results available using:

- `X-Total-Count` header attribute with the total number of available results
- `X-Pagination-Throttled` header set to `true`

```
https://serverRoot/mefApi/legato/serviceOrderingManagement/v5/serviceOrder?state=completed&limit=10&offset=0
```

The example above shows a BUS's request to get all `ServiceOrders` that are in the `completed` state. Additionally, the BUS asks only for a first (`offset=0`) pack of 10 results (`limit=10`) to be returned. The correct response (HTTP code `200`) in the response body contains a list of `ServiceOrder` objects matching the criteria.

[R31] In case no items matching the criteria are found, the SOF **MUST** return a valid response with an empty list.

6.3. Use Case 3: Retrieve Service Order by Service Order Identifier

The BUS can get detailed information about the Service Order from the SOF by using a `GET /serviceOrder/{{id}}` operation. The payload returned in the response includes all attributes the BUS has provided while sending a Service Order create request. The attributes provided by the SOF depend on the status of the `ServiceOrder` and may require some time to be set.

Both Get List and Get by Identifier operations return the same `ServiceOrder` representation, so a response to a get by id for a `ServiceOrder` with `id=00000000-3333-4444-5555-000000004567` would return exactly sae response as presented in section 6.1.3.

[R32] In case `id` does not allow finding a `ServiceOrder` in SOF's system, an error response `Error404` **MUST** be returned.

[R33] Once the service identifier (`serviceOrder.serviceOrderItem.service.id`) is assigned, it **MUST** be provided in the SOF's response.

6.4. Use case 4: Register for Notifications

The SOF communicates with the BUS with Notifications provided that:

- BUS supports a notification mechanism
- BUS has registered to receive notifications from the SOF

[O4] BUS **MAY** register for Notifications.

Supporting Notification is mandatory for SOF.

To register for notifications the BUS uses the `registerListener` operation from the API: `POST /hub`. The request contains only 2 attributes:

- `callback` - mandatory, to provide the callback address the events will be notified to,
- `query` - optional, to provide the required types of event.

Figure 16 shows all entities involved in the Notification use cases.

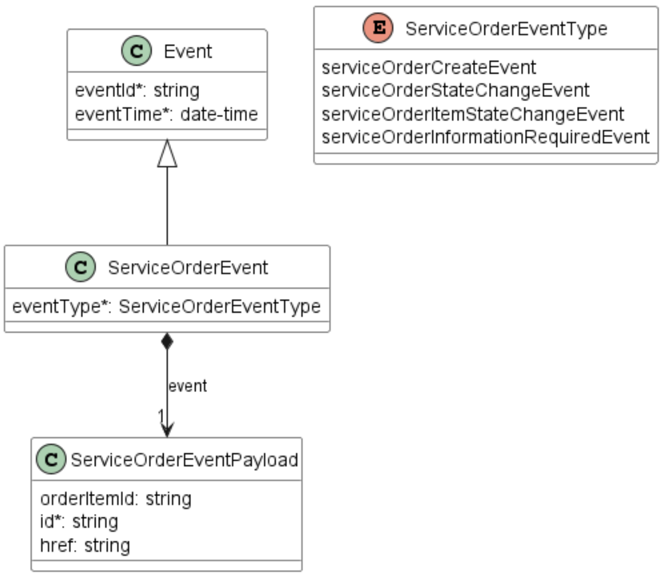


Figure 16. Service Ordering Notification Data Model

By using a simple request:

```
{
  "callback": "https://bus.com/listenerEndpoint"
}
```

The BUS subscribes for notification of all types of events. Those are:

- `serviceOrderCreateEvent`
- `serviceOrderStateChangeEvent`
- `serviceOrderItemStateChangeEvent`
- `serviceOrderInformationRequiredEvent`

If the BUS wishes to receive only notifications of a certain type, a `query` must be added:

```
{
  "callback": "https://bus.com/listenerEndpoint",
  "query": "eventType=serviceOrderStateChangeEvent"
}
```

If the BUS wishes to subscribe to 2 different types of events, there are 2 possible syntax variants [TMF630]:

```
eventType=serviceOrderStateChangeEvent,serviceOrderItemStateChangeEvent
```

or

```
eventType=serviceOrderStateChangeEvent&eventType=serviceOrderItemStateChangeEvent
```

The `query` formatting complies with RFC3986 RFC3986. According to it, every attribute defined in the Event model (from notification API) can be used in the `query`. However, this standard requires only `eventType` attribute to be supported.

[R34] `eventType` is the only attribute that the SOF **MUST** support in the query.

The SOF responds to the subscription request by adding the `id` of the subscription to the message that must be further used for unsubscribing.

```
{
  "id": "00000000-0000-0000-0000-000000000678",
  "callback": "https://bus.com/listenerEndpoint",
  "query": "eventType=serviceOrderStateChangeEvent"
}
```

Example of a final address that the Notifications will be sent to (for `serviceOrderStateChangeEvent`):

- `https://bus.com/listenerEndpoint/mefApi/legato/serviceOrderingNotification/v5/listener/serviceOrderStateChangeEvent`

6.5. Use case 5: Send Notification

Notifications are used to asynchronously inform the BUS about the respective objects and attributes changes.

For sake of readability, all previous flow diagrams presented only cases of using only the `serviceOrderStateChangeEvent`. Figure 17 presents the an end-to-end sequence of communication in Use Case 1 - Create Service Order with BUS's subscription to both `serviceOrderStateChangeEvent` and `serviceOrderItemStateChangeEvent` event types.

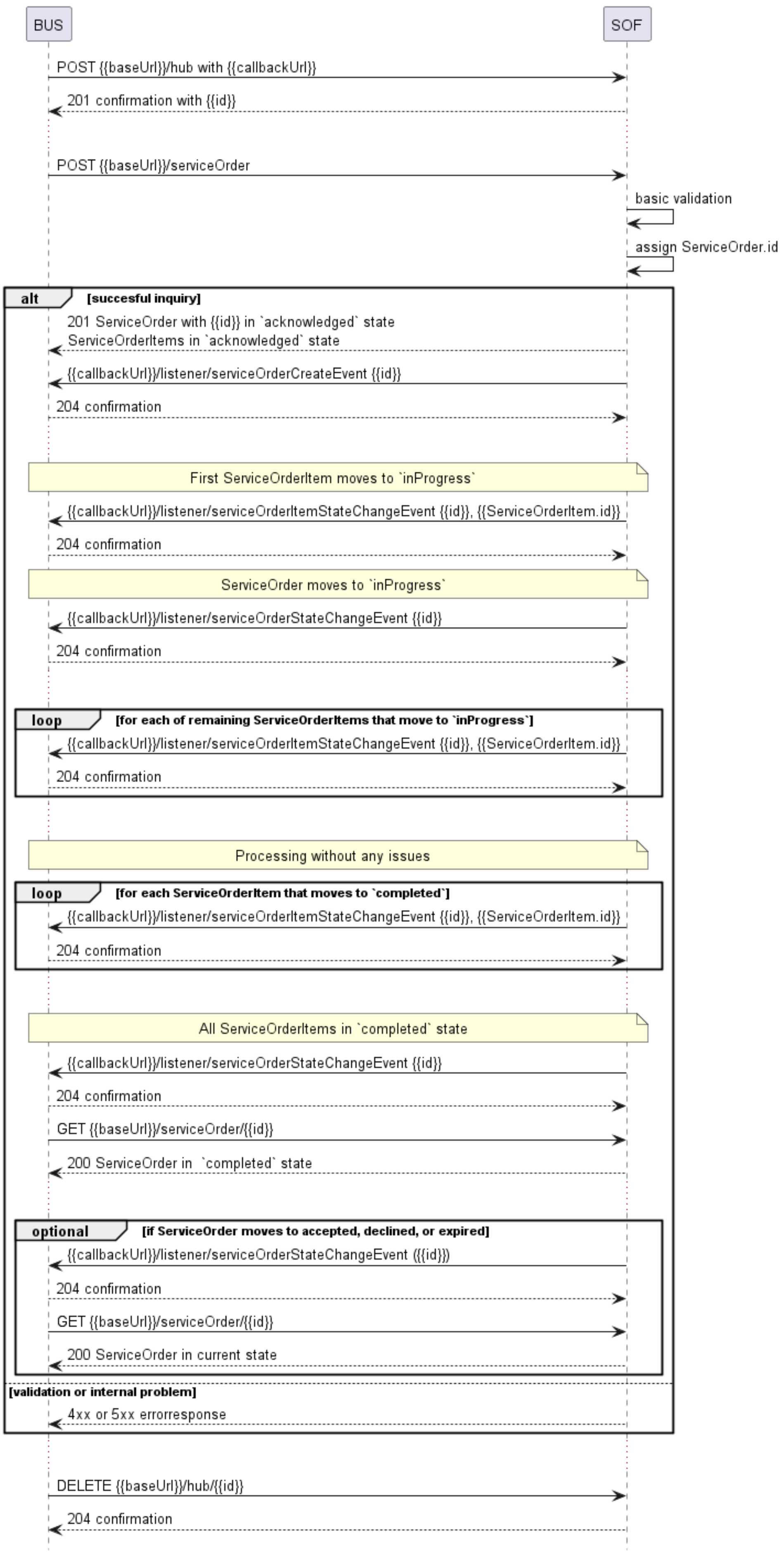

BUS
SOF
POST {{baseUrl}}/hub with {{callbackUrl}}
201 confirmation with {{id}}
POST {{baseUrl}}/serviceOrder
basic validation
assign ServiceOrder.id
alt
[succesful inquiry]
201 ServiceOrder with {{id}} in `acknowledged` state
ServiceOrderItems in `acknowledged` state
{{callbackUrl}}/listener/serviceOrderCreateEvent {{id}}
204 confirmation
First ServiceOrderItem moves to `inProgress`
{{callbackUrl}}/listener/serviceOrderItemStateChangeEvent {{id}}, {{ServiceOrderItem.id}}
204 confirmation
ServiceOrder moves to `inProgress`
{{callbackUrl}}/listener/serviceOrderStateChangeEvent {{id}}
204 confirmation
loop
[for each of remaining ServiceOrderItems that move to `inProgress`]
{{callbackUrl}}/listener/serviceOrderItemStateChangeEvent {{id}}, {{ServiceOrderItem.id}}
204 confirmation
Processing without any issues
loop
[for each ServiceOrderItem that moves to `completed`]
{{callbackUrl}}/listener/serviceOrderItemStateChangeEvent {{id}}, {{ServiceOrderItem.id}}
204 confirmation
All ServiceOrderItems in `completed` state
{{callbackUrl}}/listener/serviceOrderStateChangeEvent {{id}}
204 confirmation
GET {{baseUrl}}/serviceOrder/{{id}}
200 ServiceOrder in `completed` state
optional
[if ServiceOrder moves to accepted, declined, or expired]
{{callbackUrl}}/listener/serviceOrderStateChangeEvent ({{id}})
204 confirmation
GET {{baseUrl}}/serviceOrder/{{id}}
200 ServiceOrder in current state
[validation or internal problem]
4xx or 5xx errorresponse
DELETE {{baseUrl}}/hub/{{id}}
204 confirmation

Figure 17. Use Case 1 - Create Service Order with all Notifications

After a successful Notification subscription, the BUS sends a Service Order create request. The SOF responds with Service Order and all items in `acknowledged` state. Creation of Service Order is notified with a `serviceOrderCreateEvent`. When the first Service Order Item moves to `inProgress`, a `serviceOrderItemStateChangeEvent` is sent. Immediately the Service Order also changes its state to `inProgress` and the `serviceOrderStateChangeEvent` is sent. Then the rest (if any) of the Service Order Items are processed. When particular items are done processing they reach the `completed` state. Once all are successfully done, the Service Order also changes state to `completed`. The BUS will likely now ask for the Service Order details.

Note: The state change notification are sent only when the state attribute actually changes it's value. There are no status change notifications sent upon Service Order or Service Order Item creation.

[R35] The SOF **MUST NOT** send Notifications to BUS that have not registered for them.

[R36] The SOF **MUST** send Notifications to BUS that have registered for them.

Following snippets present examples of `serviceOrderStateChangeEvent` and `serviceOrderItemStateChangeEvent`:

```
{
  "eventId": "event-001",
  "eventType": "serviceOrderStateChangeEvent",
  "eventTime": "2022-12-28T20:45:24.796Z",
  "event": {
    "id": "00000000-3333-4444-5555-000000004567"
  }
}
```

[R37] An event triggered by the Service Order Item (`serviceOrderItemStateChangeEvent`) **MUST** additionally contain the relative `orderItemId`.

```
{
  "eventId": "event-002",
  "eventType": "serviceOrderItemStateChangeEvent",
  "eventTime": "2023-01-15T20:45:24.796Z",
  "event": {
    "id": "00000000-3333-4444-5555-000000004567",
    "orderItemId": "item-001"
  }
}
```

Note: the body of the event carries only the source object's `id`. The BUS needs to query it later by `id` to get details.

To stop receiving events, the BUS has to use the `unregisterListener` operation from the `DELETE /hub/{id}` endpoint. The `id` is the identifier received from the SOF during the listener registration.

6.6. Service Lifecycle

Above chapters focus on the requirements and the lifecycle of `ServiceOrder` and `ServiceOrderItem`. It is also very important to understand the lifecycle of the `Service` itself and how to manage it with the Service Ordering.

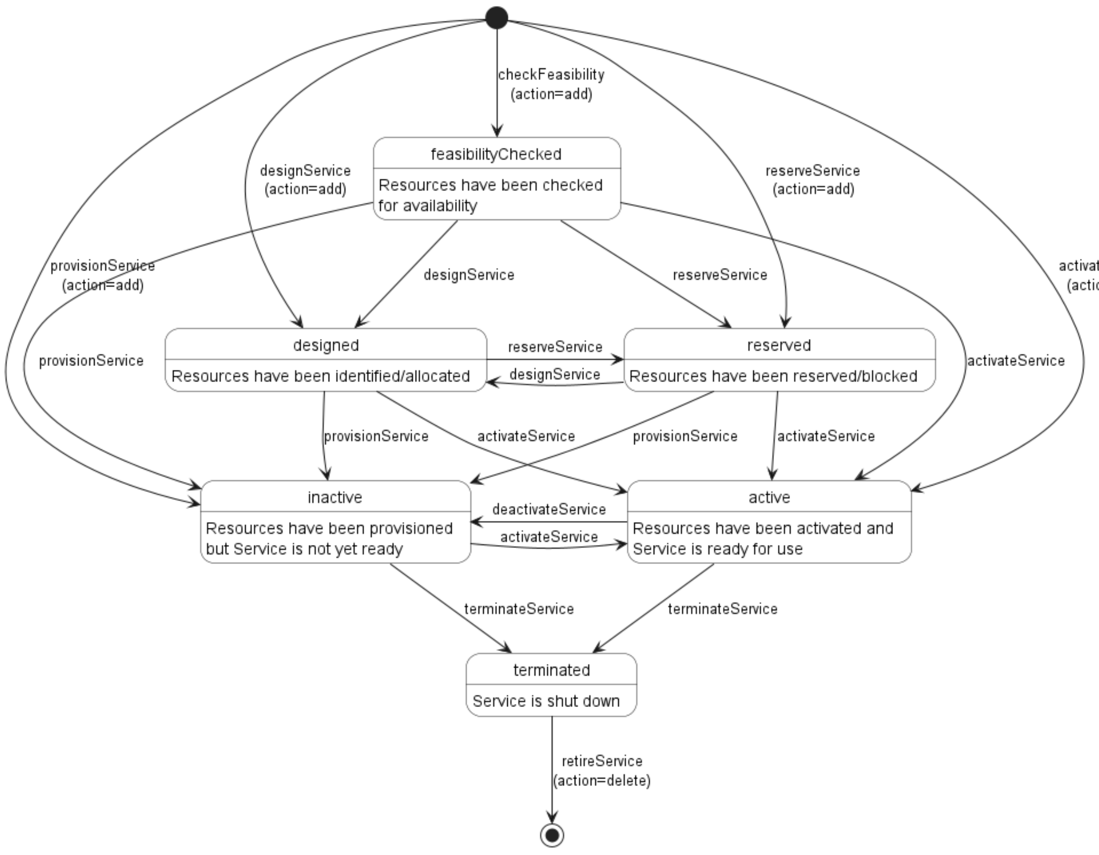

checkFeasibility (action=add)
feasibilityChecked
Resources have been checked for availability
designService (action=add)
reserveService (action=add)
provisionService (action=add)
activateService (action=add)
designService
reserveService
provisionService
activateService
designed
Resources have been identified/allocated
reserveService
designService
reserved
Resources have been reserved/blocked
provisionService
activateService
provisionService
activateService
inactive
Resources have been provisioned but Service is not yet ready
deactivateService
activateService
active
Resources have been activated and Service is ready for use
terminateService
terminateService
terminated
Service is shut down
retireService (action=delete)

Figure 18. Service Lifecycle

Figure 18 depicts the Service available states and their transitions.

The service lifecycle starts with the `state` provided in the add request. All but `terminated` can be the initial state.

BUS can order Service state transition by placing a `ServiceOrderItem` with `action=modify` and providing the desired `service.state` attribute. Transitions triggered by the same desired state form sort of use cases that can be performed on a Service. They are gathered in Table 8 together with requirements on the Service state they are applicable for.

Use case	action	state	pre-condition
checkFeasibility	add	feasibilityChecked	N/A
designService	add	designed	N/A
	modify	designed	feasibilityChecked reserved

Use case	action	state	pre-condition
reserveService	add	reserved	N/A
	modify	reserved	feasibilityChecked designed
provisionService	add	inactive	N/A
	modify	inactive	feasibilityChecked designed reserved
activateService	add	active	N/A
	modify	active	feasibilityChecked designed reserved inactive
deactivate	modify	inactive	active
terminateService	modify	terminated	inactive active

Table 9. Service Life Use Cases

A Service in `state=terminated` can be retired (deleted) with a `ServiceOrderItem` with `action=delete`.

Table 10 summarizes the states and their descriptions:

State	Description
feasibilityChecked	Initial check whether the necessary resources are available and sufficient for the installation of a given service.
designed	The Service is designed. The resources are identified and/or allocated, but not reserved.
reserved	All required resources for given service are reserved and ready.
inactive	The service is deactivated and is no longer available.
active	The service is fully available and active
terminated	The service is 'logically deleted'. All associated resources are freed and made available for service to other users.

Table 10. Service states

7. API Details

7.1. API patterns

7.1.1. Indicating errors

Erroneous situations are indicated by appropriate HTTP responses. An error response is indicated by HTTP status 4xx (for client errors) or 5xx (for server errors) and appropriate response payload. The Service Order API uses the error responses as depicted and described below.

Implementations can use HTTP error codes not specified in this standard in compliance with rules defined in RFC 7231 [RFC7231]. In such a case, the error message body structure might be aligned with the `Error`.

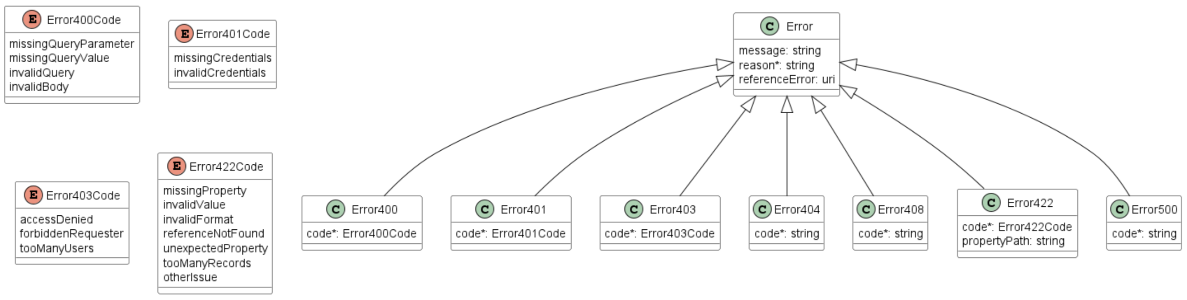


Figure 19. Data model types to represent an erroneous response

7.1.1.1. Type Error

Description: Standard Class used to describe API response error Not intended to be used directly. The `code` in the HTTP header is used as a discriminator for the type of error returned in runtime.

Name	Type	Description
message	string	Text that provides mode details and corrective actions related to the error. This can be shown to a client user.
reason*	string	Text that explains the reason for the error. This can be shown to a client user.
referenceError	uri	URL pointing to documentation describing the error

7.1.1.2. Type Error400

Description: Bad Request. (https://tools.ietf.org/html/rfc7231#section-6.5.1)

Inherits from:

- Error

Name	Type	Description
code*	Error400Code	One of the following error codes: - missingQueryParameter: The URI is missing a required query-string parameter - missingQueryValue: The URI is missing a required query-string parameter value - invalidQuery: The query section of the URI is invalid. - invalidBody: The request has an invalid body

7.1.1.3. `enum` Error400Code

Description: One of the following error codes:

- missingQueryParameter: The URI is missing a required query-string parameter
- missingQueryValue: The URI is missing a required query-string parameter value
- invalidQuery: The query section of the URI is invalid.
- invalidBody: The request has an invalid body

Value
missingQueryParameter
missingQueryValue
invalidQuery
invalidBody

7.1.1.4. Type Error401

Description: Unauthorized. (https://tools.ietf.org/html/rfc7235#section-3.1)

Inherits from:

- Error

Name	Type	Description
code*	Error401Code	One of the following error codes: - missingCredentials: No credentials provided. - invalidCredentials: Provided credentials are invalid or expired

7.1.1.5. `enum` Error401Code

Description: One of the following error codes:

- missingCredentials: No credentials provided.
- invalidCredentials: Provided credentials are invalid or expired

Value
missingCredentials
invalidCredentials

7.1.1.6. Type Error403

Description: Forbidden. This code indicates that the server understood the request but refuses to authorize it. (https://tools.ietf.org/html/rfc7231#section-6.5.3)

Inherits from:

- Error

Name	Type	Description
code*	Error403Code	This code indicates that the server understood the request but refuses to authorize it because of one of the following error codes: - accessDenied: Access denied - forbiddenRequester: Forbidden requester - tooManyUsers: Too many users

7.1.1.7. enum Error403Code

Description: This code indicates that the server understood the request but refuses to authorize it because of one of the following error codes:

- accessDenied: Access denied
- forbiddenRequester: Forbidden requester
- tooManyUsers: Too many users

Value
accessDenied
forbiddenRequester
tooManyUsers

7.1.1.8. Type Error404

Description: Resource for the requested path not found. (https://tools.ietf.org/html/rfc7231#section-6.5.4)

Inherits from:

- Error

Name	Type	Description
code*	string	The following error code: - notFound: A current representation for the target resource not found

7.1.1.9. Type Error422

The response for HTTP status `422` is a list of elements that are structured using the `Error422` data type. Each list item describes a business validation problem. This type introduces the `propertyPath` attribute which points to the erroneous property of the request, so that the BUS may fix it easier. It is highly recommended that this property should be used, yet remains optional because it might be hard to implement.

Description: Unprocessable entity due to a business validation problem. (https://tools.ietf.org/html/rfc4918#section-11.2)

Inherits from:

- Error

Name	Type	Description
code*	Error422Code	One of the following error codes: - missingProperty: The property that was expected is not present in the payload - invalidValue: The property has an incorrect value - invalidFormat: The property value does not comply with the expected value format - referenceNotFound: The object referenced by the property cannot be identified in the target system - unexpectedProperty: Additional, not expected property has been provided - tooManyRecords: the number of records to be provided in the response exceeds the threshold. - otherIssue: Other problem was identified (detailed information provided in a reason)
propertyPath	string	A pointer to a particular property of the payload that caused the validation issue. It is highly recommended that this property should be used. Defined using JavaScript Object Notation (JSON) Pointer (https://tools.ietf.org/html/rfc6901).

7.1.1.10. `enum` Error422Code

Description: One of the following error codes:

- missingProperty: The property that was expected is not present in the payload
- invalidValue: The property has an incorrect value
- invalidFormat: The property value does not comply with the expected value format
- referenceNotFound: The object referenced by the property cannot be identified in the target system
- unexpectedProperty: Additional, not expected property has been provided
- tooManyRecords: the number of records to be provided in the response exceeds the threshold.
- otherIssue: Other problem was identified (detailed information provided in a reason)

Value
missingProperty
invalidValue
invalidFormat
referenceNotFound
unexpectedProperty
tooManyRecords
otherIssue

7.1.1.11. Type Error500

Description: Internal Server Error. (https://tools.ietf.org/html/rfc7231#section-6.6.1)

Inherits from:

- Error

Name	Type	Description
code*	string	The following error code: - internalError: Internal server error - the server encountered an unexpected condition that prevented it from fulfilling the request.

7.2. Management API Data model

Figure 20 presents the whole Service Order Management data model. The data types are discussed later in this section.

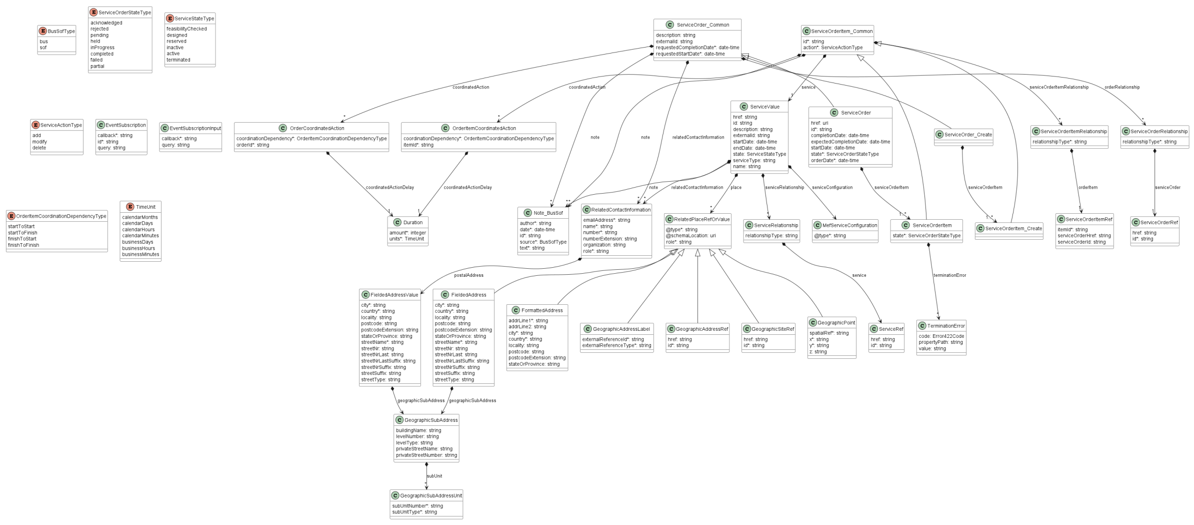


Figure 20. Service Order Management Data Model

7.2.1. ServiceOrder

7.2.1.1 Type ServiceOrder_Common

Description: A Service Order is used to request operations on a Service instance. A Service Order groups one or more one Service Order Items - one per specific action on a Service instance. The Action associated with the Service Order Item describes the operation (add, modify, delete) to be applied on the specified Service instance.The Service Order Item and its associated Action can operate on both existing (modify, delete) as well as future (add) Service instance.The Service Order is triggered from the Business Application (BA) system in charge of the Service Order management to the Service Orchestration Function (SOF) system that will orchestrate the Service fulfillment.

This type defines all attributes common to objects used in request and response.

Name	Type	Multiplicity	Description
coordinatedAction	OrderCoordinatedAction[]	0..*	The interval after the completion of one or more related Service Order Items that this Service Order Item can be started or completed
description	string	0..1	A free-text description of the service order
externalId	string	0..1	ID given by the consumer to facilitate searches

Name	Type	Multiplicity	Description
note	Note_BusSof[]	0..*	Extra-information about the order; e.g. useful to add extra delivery information that could be useful for a human process
orderRelationship	ServiceOrderRelationship[]	0..*	A list of service orders related to this order (e.g. prerequisite, dependent on)
relatedContactInformation	RelatedContactInformation[]	0..*	Contact information of an individual or organization playing a role for this Service Order. For providing Notification Contact, `role=notificationContact` MUST be used.
requestedCompletionDate*	date-time	1	Requested delivery date from the requestors perspective
requestedStartDate*	date-time	1	Order start date wished by the requestor

7.2.1.2. Type ServiceOrder_Create

Description: A Service Order is used to request operations on a Service instance. A Service Order groups one or more one Service Order Items - one per specific action on a Service instance. The Action associated with the Service Order Item describes the operation (add, modify, delete) to be applied on the specified Service instance.The Service Order Item and its associated Action can operate on both existing (modify, delete) as well as future (add) Service instance.The Service Order is triggered from the Business Application (BA) system in charge of the Service Order management to the Service Orchestration Function (SOF) system that will orchestrate the Service fulfillment. This type extends `ServiceOrder_Common` and adds attributes specific to the request response.

Inherits from:

- ServiceOrder_Common

Name	Type	Multiplicity	Description

Name	Type	Multiplicity	Description
serviceOrderItem*	ServiceOrderItem_Create[]	1..*	A list of service order items to be processed by this order

7.2.1.3. Type ServiceOrder

Description: A Service Order is used to request operations on a Service instance. A Service Order groups one or more one Service Order Items - one per specific action on a Service instance. The Action associated with the Service Order Item describes the operation (add, modify, delete) to be applied on the specified Service instance.The Service Order Item and its associated Action can operate on both existing (modify, delete) as well as future (add) Service instance.The Service Order is triggered from the Business Application (BA) system in charge of the Service Order management to the Service Orchestration Function (SOF) system that will orchestrate the Service fulfillment.

Inherits from:

- ServiceOrder_Common

Name	Type	Multiplicity	Description
href	uri	0..1	Hyperlink reference
id*	string	1	unique identifier
completionDate	date-time	0..1	Effective delivery date amended by the provider
expectedCompletionDate	date-time	0..1	Expected delivery date amended by the provider
serviceOrderItem*	ServiceOrderItem[]	1..*	A list of service order items to be processed by this order
startDate	date-time	0..1	Date when the order was started for processing
state*	ServiceOrderStateType	1	The state of the Service Order

Name	Type	Multiplicity	Description
orderDate*	date-time	1	Date when the Service Order was created in the SOF's system and a Service Order Identifier was assigned

7.2.1.4. `enum` ServiceOrderStateType

Description: Possible values for the state of a Service Order

State	Description
acknowledged	A `ServiceOrder` request has been received and has passed message and basic validations and a *Success Response* has been sent.
rejected	This state indicates that: - Invalid information is provided through the `ServiceOrder` / `ServiceOrderItem` request - The request fails to meet validation rules for `Service` delivery (processing) If one `ServiceOrderItem` is rejected, then the entire `ServiceOrder` request is rejected and a *Error Response* is sent.
inProgress	This state indicates that all `ServiceOrderItems` have successfully passed the validations checks and the scheduled `Service` delivery/processing has started. The `ServiceOrder` will be in *`inProgress`* state if *at least one* `ServiceOrderItem` is in *`inProgress`* state
pending	This state indicates that a `ServiceOrderItem` is currently in a waiting stage for an action/activity to be completed before the order-processing can progress further, pending order amend or cancel assessment. A *`pending`* state can lead into auto cancellation of an `ServiceOrderItem`, if no action is taken within the agreed timeframe. The `ServiceOrder` will be in *`pending`* state if *at least one* `ServiceOrderItem` is in *`pending`* state
held	This state indicates that a `ServiceOrderItem` cannot be progressed due to an issue. The `Service` delivery (processing) has been temporarily delayed to resolve an infrastructure shortfall to facilitate supply of order. Upon resolution of the issue, the `ServiceOrderItem` will continue to progress. A *`held`* state can lead into auto cancellation of an `ServiceOrderItem`, if no action is taken within the agreed timeframe. The `ServiceOrder` will be in *`held`* state if at least one `ServiceOrderItem` is in *`held`* state

State	Description
failed	This state indicates that `Service` delivery (processing) associated with a `ServiceOrderItem` has failed. This indicates an irrecoverable error as opposed to *`held`* or *`pending`* issues. The `ServiceOrder` will be in *`failed`* state if at *ALL* `ServiceOrderItems` are in *`failed`* state
completed	This state indicates that `Service` delivery (processing) associated with a `ServiceOrderItem` has completed. The `ServiceOrder` will be in *`completed`* state if at *ALL* `ServiceOrderItems` are in *`completed`* state
partial	This state indicates that some `ServiceOrderItem` are in *`completed`* state while others are in *`cancelled`* and/or *`failed`* states, so the entire `ServiceOrder` is in a *`partial`* state.

7.2.1.5. Type ServiceOrderRef

Description: Reference to a Service Order instance.

Name	Type	Multiplicity	Description
href	string	0..1	A hyperlink to the related order
id*	string	1	The id of the related order

7.2.1.6. Type ServiceOrderRelationship

Description: Reference to a related Service Order and the type of that association.

Name	Type	Multiplicity	Description
serviceOrder*	ServiceOrderRef	1	A reference to a Service Order

Name	Type	Multiplicity	Description
relationshipType*	string	1	Specifies the type (nature) of the relationship to the related Service. The nature of required relationships varies for Services of different types. For example, a UNI or ENNI Service may not have any relationships, but an Access E-Line may have two mandatory relationships (related to the UNI on one end and the ENNI on the other). More complex Services such as multipoint IP or Firewall Services may have more complex relationships. As a result, the allowed and mandatory `relationshipType` values are defined in the Service Specification.

7.2.2. Service Order Item

7.2.2.1 Type ServiceOrderItem_Common

Description: An identified part of the order. A service order is decomposed into one or more order items. This type holds the attributes common to request and response representation of the Service Order Item.

Name	Type	Multiplicity	Description
id*	string	1	Identifier of the order item (generally it is a sequence number 01, 02, 03, ...)
action*	ServiceActionType	1	Action to be applied to the Service referred by this Service Order Item

Name	Type	Multiplicity	Description
coordinatedAction	OrderItemCoordinatedAction[]	0..*	The interval after the completion of one or more related Service Order Items that this Service Order Item can be started or completed
note	Note_BusSof[]	0..*	Extra-information about the order item; e.g. useful to add extra delivery information that could be useful for a human process
service*	ServiceValue	1	A description of the service that is the subject of this service order item.
serviceOrderItemRelationship	ServiceOrderItemRelationship[]	0..*	Specifies the type (nature) of the relationship to the related Service. The nature of required relationships varies for Services of different types. For example, a UNI or ENNI Service may not have any relationships, but an E-Line may have two

Name	Type	Multiplicity	Description
			mandatory relationships (related to the UNI on one end and the ENNI on the other). More complex Services such as multipoint IP or Firewall Services may have more complex relationships. As a result, the allowed and mandatory `relationshipType` values are defined in the Service Specification. Related items can be both from within the same Service Order or from other one. When referencing item within the same Service Order,

7.2.2.2. Type ServiceOrderItem_Create

Description: An identified part of the order. A service order is decomposed into one or more order items. This type is used in the request.

Inherits from:

- ServiceOrderItem_Common

7.2.2.3. Type ServiceOrderItem

Description: An identified part of the order. A service order is decomposed into one or more order items. The modelling pattern introduces the `Common` supertype to aggregate attributes that are common to both `ServiceOrderItem` and `ServiceOrderItem_Create`. The `Create` type has a subset of attributes of the response type and does not introduce any new, thus the `Create` type has an empty definition

Inherits from:

- ServiceOrderItem_Common

Name	Type	Multiplicity	Description
state*	ServiceOrderStateType	1	State of the Service Order Item
terminationError	TerminationError[]	0..*	When the SOF cannot process the request, the SOF returns a text-based list of reasons here.

7.2.2.4. `enum` ServiceActionType

Description: Action to be applied to the Service referred by this Service Order Item

Value
add
modify
delete

7.2.2.5. Type ServiceOrderItemRef

Description: A reference to a Service Order Item. When referencing item from within the same Service Order, the `serviceOrderId` and `serviceOrderHref` MUST be empty.

Name	Type	Multiplicity	Description
itemId*	string	1	Identifier of referenced item within the referenced Service Order
serviceOrderHref	string	0..1	Link to the order to which the referenced item belongs to
serviceOrderId	string	0..1	Identifier of the order to which the referenced item belongs to

7.2.2.6. Type ServiceOrderItemRelationship

Description: Specifies the type (nature) of the relationship to the related Service. The nature of required relationships varies for Services of different types. For example, a UNI or

ENNI Service may not have any relationships, but an E-Line may have two mandatory relationships (related to the UNI on one end and the ENNI on the other). More complex Services such as multipoint IP or Firewall Services may have more complex relationships. As a result, the allowed and mandatory `relationshipType` values are defined in the Service Specification. Related item can be both from within the same Service Order or from other one. When referencing item from within the same Service Order, the `orderItem.serviceOrderId` and `orderItem.serviceOrderHref` MUST be empty.

Name	Type	Multiplicity	Description
orderItem*	ServiceOrderItemRef	1	A reference to a Service Order Item
relationshipType*	string	1	Specifies the nature of the relationship to the related Service Order Item. A string that is one of the relationship types specified in the Service Specification.

7.2.3. Service representation

7.2.3.1. Type ServiceValue

Description: ServiceValue is a base class for defining the Service.

Name	Type	Multiplicity	Description
href	string	0..1	Hyperlink reference to a Service
id	string	0..1	unique identifier of a Service
description	string	0..1	Free-text description of the service
externalId	string	0..1	ID given by the consumer to facilitate searches
startDate	date-time	0..1	Date when the service starts
endDate	date-time	0..1	Date when the service ends
state	ServiceStateType	0..1	Represent the state of lifecycle of the Service Order.

Name	Type	Multiplicity	Description
note	Note_BusSof[]	0..*	A list of notes made on this service
serviceType	string	0..1	Business type of the service
name	string	0..1	Name of the service
serviceRelationship	ServiceRelationship[]	0..*	Specifies the type (nature) of the relationship to the related Service. The nature of required relationships varies for Services of different types. For example, a UNI or ENNI Service may not have any relationships, but an Access E-Line may have two mandatory relationships (related to the UNI on one end and the ENNI on the other). More complex Services such as multipoint IP or Firewall Services may have more complex relationships. As a result, the allowed and mandatory `relationshipType` values are defined in the Service Specification.
relatedContactInformation	RelatedContactInformation[]	0..*	Contact information of an individual or organization playing a role for this Service
place	RelatedPlaceRefOrValue[]	0..*	The relationships between this Service Order Item and one or more Places as defined in the Service Specification.

Name	Type	Multiplicity	Description
serviceConfiguration	MefServiceConfiguration	0..1	MEFServiceConfiguration is used to specify the MEF specific service payload. This field MUST be populated for all item 'actions' other than 'delete'. It MUST NOT be populated when an item `action` is `delete`. The @type is used as a discriminator.

7.2.3.2. Type MefServiceConfiguration

Description: MEFServiceConfiguration is used as an extension point for MEF specific service payload. The `@type` attribute is used as a discriminator

Name	Type	Multiplicity	Description
@type*	string	1	The value of the "$id" as defined in the JSON schema of the service.

7.2.3.3. Type ServiceRelationship

Description: A relationship to an existing Service. The requirements for usage for given Service are described in the Service Specification.

Name	Type	Multiplicity	Description
relationshipType*	string	1	Specifies the type (nature) of the relationship to the related Service. The nature of required relationships varies for Services of different types. For example, a UNI or ENNI Service may not have any relationships, but an Access E-Line may have two mandatory relationships (related to the UNI on one end and the ENNI on the other). More complex Services such as multipoint IP or Firewall Services may have more complex relationships. As a result, the allowed and mandatory `relationshipType` values are defined in the Service Specification.

Name	Type	Multiplicity	Description
service*	ServiceRef	1	A reference to a Service

7.2.3.4. enum ServiceStateType

Description: Valid values for the lifecycle state of the Service.

State	Description
feasibilityChecked	Initial check whether the necessary resources are available and sufficient for the installation of a given service.
designed	The Service is designed. The resources are identified and/or allocated, but not reserved.
reserved	All required resources for given service are reserved and ready.
inactive	The service is deactivated and is no longer available.
active	The service is fully available and active
terminated	The service is 'logically deleted'. All associated resources are freed and made available for service to other users.

7.2.3.5. Type ServiceRef

Description: Reference to a Service instance.

Name	Type	Multiplicity	Description
href	string	0..1	Hyperlink reference to Service
id*	string	1	unique identifier of Service

7.2.4. Place representation

There are several formats in which place information can be introduced to the Service Order request. They are described in Section 6.1.8.

7.2.4.1. Type RelatedPlaceRefOrValue

Description: A Place provided either by value or by reference

Name	Type	Multiplicity	Description
@type*	string	1	This field is used as a discriminator and is used between different place representations. This type might discriminate for additional related place as defined in '@schemaLocation'.

Name	Type	Multiplicity	Description
@schemaLocation	uri	0..1	A URI to a JSON-Schema file that defines additional attributes and relationships. May be used to define additional related place types.
role*	string	1	Role of this place

7.2.4.2. Type FieldedAddress

Description: A type of Address that has a discrete field and value for each type of boundary or identifier down to the lowest level of detail. For example "street number" is one field, "street name" is another field, etc. Reference: MEF 79 (Sn 8.9.2)

Inherits from:

- RelatedPlaceRefOrValue

Name	Type	Multiplicity	Description
city*	string	1	The city that the address is in
country*	string	1	Country that the address is in
geographicSubAddress	GeographicSubAddress	0..1	Additional fields used to specify an address, as detailed as possible.
locality	string	0..1	The locality that the address is in
postcode	string	0..1	Descriptor for a postal delivery area, used to speed and simplify the delivery of mail (also known as zip code)
postcodeExtension	string	0..1	An extension of a postal code. E.g. the part following the dash in a US urban property address
stateOrProvince	string	0..1	The State or Province that the address is in
streetName*	string	1	Name of the street or other street type

Name	Type	Multiplicity	Description
streetNr	string	0..1	Number identifying a specific property on a public street. It may be combined with streetNrLast for ranged addresses. MEF 79 defines it as required however as in certain countries it is not used we make it optional in API.
streetNrLast	string	0..1	Last number in a range of street numbers allocated to a property
streetNrLastSuffix	string	0..1	Last street number suffix for a ranged address
streetNrSuffix	string	0..1	The first street number suffix
streetSuffix	string	0..1	A modifier denoting a relative direction
streetType	string	0..1	The type of street (e.g., alley, avenue, boulevard, brae, crescent, drive, highway, lane, terrace, parade, place, tarn, way, wharf)

7.2.4.3. Type FieldedAddressValue

Description: A type of Address that has a discrete field and value for each type of boundary or identifier down to the lowest level of detail. For example "street number" is one field, "street name" is another field, etc. Reference: MEF 79 (Sn 8.9.2)

Name	Type	Multiplicity	Description
city*	string	1	The city that the address is in

Name	Type	Multiplicity	Description
country*	string	1	Country that the address is in
geographicSubAddress	GeographicSubAddress	0..1	Additional fields used to specify an address, as detailed as possible.
locality	string	0..1	The locality that the address is in
postcode	string	0..1	Descriptor for a postal delivery area, used to speed and simplify the delivery of mail (also known as zip code)
postcodeExtension	string	0..1	An extension of a postal code. E.g. the part following the dash in a US urban property address
stateOrProvince	string	0..1	The State or Province that the address is in
streetName*	string	1	Name of the street or other street type
streetNr	string	0..1	Number identifying a specific property on a public street. It may be combined with streetNrLast for ranged addresses. MEF 79 defines it as required however as in certain countries it is not used we make it optional in API.
streetNrLast	string	0..1	Last number in a range of street numbers allocated to a property
streetNrLastSuffix	string	0..1	Last street number suffix for a ranged address

Name	Type	Multiplicity	Description
streetNrSuffix	string	0..1	The first street number suffix
streetSuffix	string	0..1	A modifier denoting a relative direction
streetType	string	0..1	The type of street (e.g., alley, avenue, boulevard, brae, crescent, drive, highway, lane, terrace, parade, place, tarn, way, wharf)

7.2.4.4. Type FormattedAddress

Description: A type of Address that has discrete fields for each type of boundary or identifier with the exception of street and more specific location details, which are combined into a maximum of two strings based on local postal addressing conventions. Reference: MEF 79 (Sn 8.9.3)

Inherits from:

- RelatedPlaceRefOrValue

Name	Type	Multiplicity	Description
addrLine1*	string	1	The first address line in a formatted address
addrLine2	string	0..1	The second address line in a formatted address
city*	string	1	The city that the address is in
country*	string	1	Country that the address is in
locality	string	0..1	An area of defined or undefined boundaries within a local authority or other legislatively defined area, usually rural or semi-rural in nature
postcode	string	0..1	Descriptor for a postal delivery area, used to speed and simplify the delivery of mail (also known as ZIP code)
postcodeExtension	string	0..1	An extension of a postal code. E.g. the part following the dash in an US urban property address
stateOrProvince	string	0..1	The State or Province that the address is in

7.2.4.5. Type GeographicPoint

Description: A GeographicPoint defines a geographic point through coordinates. Reference: MEF 79 (Sn 8.9.5)

Inherits from:

- RelatedPlaceRefOrValue

Name	Type	Multiplicity	Description
spatialRef*	string	1	The spatial reference system used to determine the coordinates (e.g. "WGS84"). The system used and the value of this field are to be agreed during the onboarding process.
x*	string	1	The latitude expressed in the format specified by the `spacialRef`
y*	string	1	The longitude expressed in the format specified by the `spacialRef`
z	string	0..1	The elevation expressed in the format specified by the `spacialRef`

7.2.4.6. Type GeographicAddressLabel

Description: A unique identifier controlled by a generally accepted independent administrative authority that specifies a fixed geographical location. Reference: MEF 79 (Sn 8.9.4)

Inherits from:

- RelatedPlaceRefOrValue

Name	Type	Multiplicity	Description
externalReferenceId*	string	1	A reference to an address by id
externalReferenceType*	string	1	Uniquely identifies the authority that specifies the addresses reference and/or its type (if the authority specifies more than one type of address). The value(s) to be used are to be agreed during the onboarding. For North American providers this would normally be CLLI (Common Language Location Identifier) code.

7.2.4.7. Type GeographicSubAddress

Description: Additional fields used to specify an address, as detailed as possible.

Name	Type	Multiplicity	Description
buildingName	string	0..1	Allows for identification of places that require building name as part of addressing information
levelNumber	string	0..1	Used where a level type may be repeated e.g. BASEMENT 1, BASEMENT 2
levelType	string	0..1	Describes level types within a building
privateStreetName	string	0..1	"Private streets internal to a property (e.g. a university) may have internal names that are not recorded by the land title office
privateStreetNumber	string	0..1	Private streets numbers internal to a private street
subUnit	GeographicSubAddressUnit[]	0..*	Representation of a MEFSubUnit It is used for describing subunit within a subAddress e.g. BERTH, FLAT, PIER, SUITE, SHOP, TOWER, UNIT, WHARF.

7.2.4.8. Type GeographicSubAddressUnit

Description: Allows for sub unit identification

Name	Type	Multiplicity	Description
subUnitNumber*	string	1	The discriminator used for the subunit, often just a simple number but may also be a range.
subUnitType*	string	1	The type of subunit e.g.BERTH, FLAT, PIER, SUITE, SHOP, TOWER, UNIT, WHARF.

7.2.4.9. Type GeographicAddressRef

Description: A reference to a Geographic Address resource available through Address Validation API.

Inherits from:

- RelatedPlaceRefOrValue

Name	Type	Multiplicity	Description
href	string	0..1	Hyperlink to the referenced GeographicAddress. Hyperlink MAY be provided by the SOF in responses. Hyperlink MUST be ignored by the SOF in case it is provided by the BA in a request
id*	string	1	Identifier of the referenced Geographic Address. This identifier is assigned during a successful address validation request (Geographic Address Validation API)

7.2.4.10. Type GeographicSiteRef

Description: A reference to a Geographic Site resource available through Service Site API

Inherits from:

- RelatedPlaceRefOrValue

Name	Type	Multiplicity	Description
href	string	0..1	Hyperlink to the referenced GeographicSite. Hyperlink MAY be provided by the SOF in responses. Hyperlink MUST be ignored by the SOF in case it is provided by the BA in a request
id*	string	1	Identifier of the referenced Geographic Site.

7.2.5. Notification registration

Notification registration and management are done through `/hub` API endpoint. The below sections describe data models related to this endpoint.

7.2.5.1. Type EventSubscriptionInput

Description: This class is used to register for Notifications.

Name	Type	Multiplicity	Description
callback*	string	1	This callback value must be set to *host* property from Service Ord (serviceOrderNotification.api.yaml). This property is appended with in that API to construct an URL to which notification is sent. E.g. fo service order state change event notification will be sent to: `https://bus.com/listenerEndpoint/mefApi/legato/serviceOrderingMa
query	string	0..1	This attribute is used to define to which type of events to register to. serviceOrderStateChangeEvent". To subscribe for more than one eve `eventType=serviceOrderStateChangeEvent,serviceOrderItemStateC 'serviceOrderEventType' in serviceOrderNotification.api.yaml. An e in subscription for all event types.

7.2.5.2. Type EventSubscription

Description: This resource is used to respond to notification subscriptions.

Name	Type	Multiplicity	Description
callback*	string	1	The value provided by in `EventSubscriptionInput` during notification registration
id*	string	1	An identifier of this Event Subscription assigned when a resource is created.
query	string	0..1	The value provided by the `EventSubscriptionInput` during notification registration

7.2.6. Common

Types described in this subsection are shared among two or more Cantata and Sonata APIs.

7.2.6.1. Type OrderCoordinatedAction

Description: The interval after the completion of one or more related Order that this Order can be started or completed

Name	Type	Multiplicity	Description

Name	Type	Multiplicity	Description
coordinatedActionDelay*	Duration	1	The period of time for which the coordinated action is delayed.
coordinationDependency*	OrderItemCoordinationDependencyType	1	A dependency between the Order and a related Order
orderId*	string	1	Specifies Order that is to be coordinated with this Order.

7.2.6.2. Type OrderItemCoordinatedAction

Description: The interval after the completion of one or more related Order Items that this Order Item can be started or completed

Name	Type	Multiplicity	Description
coordinatedActionDelay*	Duration	1	The period of time for which the coordinated action is delayed.
coordinationDependency*	OrderItemCoordinationDependencyType	1	A dependency between the Order Item and a related Order Item

Name	Type	Multiplicity	Description
itemId*	string	1	Specifies Order Item that is to be coordinated with this Order Item.

7.2.6.3. enum OrderItemCoordinationDependencyType

Description: Possible values of the Order Item Coordination Dependency

OrderItemCoordinationDependencyType	Description
startToStart	Work on the Specified Order Item can only be started after the Coordinated Order Items are started
startToFinish	The Coordinated Order Items must complete before work on the Specified Order Item begins
finishToStart	Work on the Related Order Items begins after the completion of the Specified Order Item
finishToFinish	Work on the Related Order Items completes at the same time as the Specified Order Item

7.2.6.4. Type Note_BusSof

Description: Extra information about a given entity. Only useful in processes involving human interaction. Not applicable for an automated process.

Name	Type	Multiplicity	Description
author*	string	1	Author of the note
date*	date-time	1	Date of the note
id*	string	1	Identifier of the note within its containing entity (may or may not be globally unique, depending on provider implementation)
source*	BusSofType	1	Indicates if this Note was added by BUS or SOF.
text*	string	1	Text of the note

7.2.6.5. Type RelatedContactInformation

Description: Contact information of an individual or organization playing a role for this Order Item. The rule for mapping a represented attribute value to a `role` is to use the *lowerCamelCase* pattern

Name	Type	Multiplicity	Description
emailAddress*	string	1	Email address
name*	string	1	Name of the contact
number*	string	1	Phone number
numberExtension	string	0..1	Phone number extension
organization	string	0..1	The organization or company that the contact belongs to
postalAddress	FieldedAddressValue	0..1	Identifies the postal address of the person or office to be contacted.
role*	string	1	A role the party plays in a given context.

The `role` attribute is used to provide a reason the particular party information is used. It can result from business requirements (e.g. SOF Contact Information) or from the Service Specification requirements.

The rule for mapping a represented attribute value to a `role` is to use the *lowerCamelCase* pattern e.g.

- BUS Contact: `role` equal to `busInformation`
- SOF Contact: `role` equal to `sofContact`

7.2.6.6. Type TerminationError

Description: This indicates an error that caused an Item to be terminated. The code and propertyPath should be used like in Error422.

Name	Type	Description

Name	Type	Description
code	Error422Code	One of the following error codes: - missingProperty: The property the SOF has expected is not present in the payload - invalidValue: The property has an incorrect value - invalidFormat: The property value does not comply with the expected value format - referenceNotFound: The object referenced by the property cannot be identified in the SOF system - unexpectedProperty: Additional property, not expected by the SOF has been provided - tooManyRecords: the number of records to be provided in the response exceeds the SOF's threshold. - otherIssue: Other problem was identified (detailed information provided in a reason)
propertyPath	string	A pointer to a particular property of the payload that caused the validation issue. It is highly recommended that this property should be used. Defined using JavaScript Object Notation (JSON) Pointer (https://tools.ietf.org/html/rfc6901).
value	string	Text to describe the reason of the termination.

7.2.6.7. `enum` TimeUnit

Description: Represents a unit of time.

Value
calendarMonths
calendarDays
calendarHours
calendarMinutes
businessDays
businessHours
businessMinutes

7.3. Notification API Data model

Figure 21 presents the Service Order Management Notification data model.

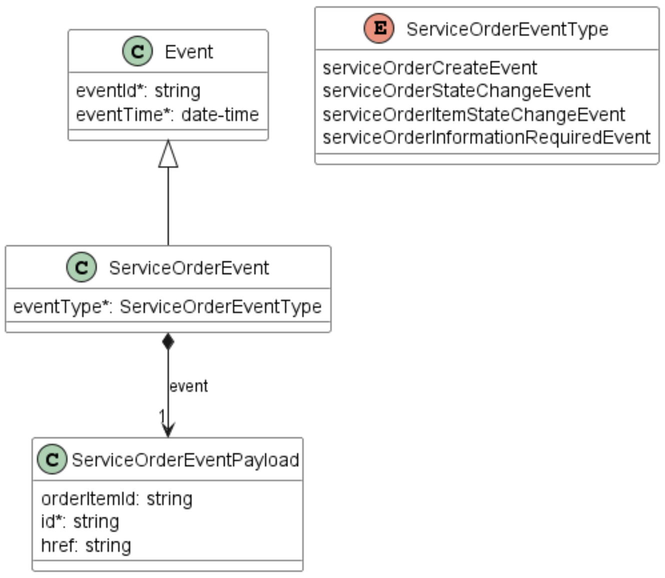


Figure 21. Service Order Management Notification Data Model

This data model is used to construct requests and responses of the API endpoints described in Section 5.2.2.

7.3.1. Type Event

Description: Event class is used to describe information structure used for notification.

Name	Type	Multiplicity	Description
eventId*	string	1	Id of the event
eventTime*	date-time	1	Date-time when the event occurred

7.3.2. Type ServiceOrderEvent

Description:

Inherits from:

- Event

Name	Type	Multiplicity	Description
eventType*	ServiceOrderEventType	1	Indicates the type of the event.
event*	ServiceOrderEventPayload	1	A reference to the Service Order that is source of the notification.

7.3.3. Type ServiceOrderEventPayload

Description: The identifier of the Service Order and Order Item being subject of this event.

Name	Type	Multiplicity	Description
orderItemId	string	0..1	ID of the Service Order Item (within the Service Order) which state change triggered the event. Mandatory for `serviceOrderItemStateChangeEvent`.
id*	string	1	ID of the Service Order
href	string	0..1	Hyperlink to access the Service Order

7.3.4. `enum` ServiceOrderEventType

Description: Indicates the type of Service Order event.

Value
serviceOrderCreateEvent
serviceOrderStateChangeEvent
serviceOrderItemStateChangeEvent
serviceOrderInformationRequiredEvent

8. References

- JSON Schema draft 7, JSON Schema: A Media Type for Describing JSON Documents and associated documents, by Austin Wright and Henry Andrews, March 2018. Copyright © 2018 IETF Trust and the persons identified as the document authors. All rights reserved.
- MEF 10.4, Subscriber Ethernet Services Attributes, December 2018
- MEF 26.2, External Network Network Interface (ENNI) and Operator Service Attributes, August 2016
- MEF 55.1 Lifecycle Service Orchestration (LSO): Reference Architecture and Framework, February 2021
- MEF 61.1, IP Service Attributes, May 2019
- MEF 61.1.1, Amendment to MEF 61.1: UNI Access Link Trunks, IP Addresses, and Mean Time to Repair Performance Metric, July 2022
- MEF 70, SD-WAN Service Attributes and Services, July 2019
- MEF 79, Address, Service Site, and Product Offering Qualification Management, Requirements and Use Cases, November 2019
- MEF 79.0.1, Amendment to MEF 79: Address, Service Site, and Product Offering Qualification Management, Requirements, and Use Cases, December 2020
- MEF 79.0.2, Amendment to MEF 79: Address Validation, July 2021
- [MEF W100], LSO Legato Service Specification - SD-WAN Schema Guide
- [MEF W101], LSO Legato Service Specification - Carrier Ethernet Schema Guide
- [MEF W102], LSO Legato Service Specification - IP/IP-VPN Schema Guide
- MEF 121, LSO Cantata and LSO Sonata Address Management API - Developer Guide, May 2022
- MEF 122, LSO Cantata and LSO Sonata Site Management API - Developer Guide, May 2022
- MEF 128, LSO API Security Profile, July 2022
- RFC2119, Key words for use in RFCs to Indicate Requirement Levels, by S. Bradner, March 1997
- RFC3986 Uniform Resource Identifier (URI): Generic Syntax, January 2005
- RFC8174, Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words, by B. Leiba, May 2017, Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.
- RFC7231, Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, June 2014 https://tools.ietf.org/html/rfc7231
- TMF630 TMF630 API Design Guidelines 4.2.0
- TMF641 TMF641 Service Order Management API REST Specification v4.1.0

Appendix A Acknowledgments

The following contributors participated in the development of this document and have requested to be included in this list.

Mike **BENCHECK**

Michał **ŁĄCZYŃSKI**

Jack **PUGACZEWSKI**

Karthik **SETHURAMAN**

Mehmet **TOY**