Open Gateway Technical Realisation Guidelines

Version 5.0

17th July 2026

Security Classification: Non-Confidential

Copyright Notice

Disclaimer

Compliance Notice

Table of Contents

1 Introduction

1.1 Overview

In the dynamic telecommunications industry, the GSMA Open Gateway initiative represents a significant step toward unified and standardised service delivery and management across Mobile Network Operators (MNOs). This initiative seeks to enhance interoperability, streamline service management, and foster innovation through standardised APIs, ensuring a seamless and consistent user experience across diverse networks. The GSMA Open Gateway Platform is a deployment option of the GSMA Operator Platform.

This GSMA Open Gateway Technical Realisation Guideline document serves as an essential resource for stakeholders—including MNOs, service aggregators, and technology partners—who are involved in deploying and utilising the GSMA Open Gateway Platform. This guideline outlines the required steps, best practices, and technical specifications necessary for successful implementation and utilisation of the GSMA Open Gateway Platform.

1.2 Purpose and Scope

The primary objective of this document is to provide a structured framework for the realisation of the GSMA Open Gateway Platform. It aims to facilitate a comprehensive understanding of the platform’s architecture, functionalities, and operational procedures. By adhering to these guidelines, stakeholders can ensure efficient deployment and integration of services, thereby enhancing interoperability and service delivery across multiple Operators and Channel Partners.

1.2.1 Audience

This guideline is intended for:

- **Mobile Network Operators:** technical and operational teams responsible for deploying and managing network services.
- **Channel partners:** entities that offer bundled services across multiple MNOs, requiring standardised and interoperable interfaces.
- **Technology Partners:** companies providing technology solutions and support for the implementation of the GSMA Open Gateway Platform.
- **Regulatory Bodies:** authorities overseeing compliance with industry standards and regulations.

1.3 Definitions

For the purposes of the present document, the term definitions given in OPG.02 [1] and the following apply.

Term	Description
API Consumer	An entity that makes an API request to, or within, the OGW Platform
API Initiator	A sub-class of API Consumer that originates the first message in the API sequence

Term	Description
Leading OGW Platform	The OGW Platform instance connected to the Application or Channel Platform and receiving the Application's requests, sharing them to the selected federated Partner OGW Platforms where required. Note: the use of this term relates to federation and is used in relation to the Partner OGW Platforms with which the Leading OGW Platform is federated.
Open Gateway (OGW) Platform	A realisation of a GSMA Operator Platform (defined in [1]), providing APIs for universal access to Operator networks for developers.
Originating OGW Platform	The OGW Platform initiating the federation creation request towards the Partner OGW Platform.
Partner OGW Platform	The OGW Platform that offers exposure of its network capabilities to other OGW Platform via E/WBI.
Synthetic Monitoring	Use of behavioural scripts to simulate actions that an End-User would take in interacting with an API and monitoring them to measure performance.
User / Resource Owner	The End-User or Subscriber which Personal Data processed by a CAMARA API relates to, the Resource Owner has the authority to authorise access to CAMARA APIs which process Personal Data.

Note: A term defined in the present document might need alignment GSMA OPG.02 [1]

1.4 Abbreviations

Term	Description
AAPrM	Authentication, Authorization and Privacy Management
API	Application Programming Interface
AuC	Authentication Centre
AUSF	AUthentication Server Function
BSS	Business Support System
CIBA	OpenID Connect Client-Initiated Backchannel Authentication
EWBI	East-West Bound Interface
GDPR	General Data Protection Regulation
HLR	Home Location Register
HSS	Home Subscriber Server
JSON	JavaScript Object Notation
JWKS	JSON Web Key Set
JWT	JSON Web Token
MCC	Mobile Country Code
MNC	Mobile Network Code
MNO	Mobile Network Operator
MVNE	Mobile Virtual Network Enabler
MVNO	Mobile Virtual Network Operator
NBI	North Bound Interface
NNI*	Network-to-Network interface (both Core and BSS)

Term	Description
OGW	Open Gateway
OIDC	OpenID Connect
OP	Operator Platform
OSS	Operations Support System
PII	Personally Identifiable Information
PLMN	Public Land Mobile Network
QoD	Quality On Demand
QoE	Quality of Experience
QoS	Quality of Service
REST	Representational State Transfer
SBI	South Bound Interface
SLA	Service Level Agreement
SOAP	Simple Object Access Protocol
UE	User Equipment
VPLMN	Visited Public Land Mobile Network
XML	Extensible Markup Language
XR	Extended Reality

1.5 References

Ref	Doc Number	Title
[1]	GSMA PRD OPG.02	Operator Platform: Requirements and Architecture
[2]	RFC 2119	"Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997. Available at http://www.ietf.org/rfc/rfc2119.txt
[3]	RFC 8174	Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words https://www.rfc-editor.org/info/rfc8174
[4]	GSMA PRD OPG.09	Open Gateway NBI APIs Realisation in the SBI
[5]	GSMA PRD WA.101	Open Gateway Channel Partner Onboarding Guide
[6]		CAMARA Project https://camaraproject.org/
[7]		CAMARA Identity And Consent Management (ICM) v0.5.0/ documentation / CAMARA-Security-Interoperability.md
[8]		CAMARA Identity And Consent Management (ICM) v0.5.0 / documentation / CAMARA-API-access-and-user-consent.md
[9]		W3C Data Privacy Vocabulary (DPV) https://w3c.github.io/dpv/2.0/dpv/
[10]		The OAuth 2.0 Authorization Framework https://datatracker.ietf.org/doc/html/rfc6749

Ref	Doc Number	Title
[11]		OpenID Connect Core 1.0 https://openid.net/specs/openid-connect-core-1_0.html
[12]	OpenID Connect Discovery 1.0	OpenID Provider Metadata https://openid.net/specs/openid-connect-discovery-1_0.html#ProviderMetadata
[13]	RFC 8414	OAuth 2.0 Authorization Server Metadata https://datatracker.ietf.org/doc/html/rfc8414#section-3
[14]	RFC 4632	Classless Inter-domain Routing (CIDR) https://datatracker.ietf.org/doc/html/rfc4632
[15]	TS 23.003	Numbering, addressing and identification https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=729
[16]	RFC 6749	The OAuth 2.0 Authorization Framework https://datatracker.ietf.org/doc/html/rfc6749
[17]	OpenID Connect Core 1.0	OIDC Client Authentication https://openid.net/specs/openid-connect-core-1_0.html#ClientAuthentication
[18]		CAMARA Commonalities v0.8.0 / documentation / API-design-guidelines.md and CAMARA-API-Event-Subscription-and-Notification-Guide.md
[19]	TMF 931	Open Gateway Onboarding and Ordering Component Suite https://www.tmforum.org/oda/open-apis/directory/open-gateway-onboarding-and-ordering-component-suite-TMF931/v5.2
[20]	TS 23.222	Common API Framework for 3GPP Northbound APIs https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3337
[21]	GSMA PRD OPG.07	Southbound Interface Charging Function APIs
[22]	TS 32.254	Telecommunication management; Charging management; Exposure function Northbound Application Program Interfaces (APIs) charging https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3275
[23]	TMF 767	Product Usage Management User Guide https://www.tmforum.org/oda/open-apis/directory/product-usage-management-api-TMF767/v5.0
[24]	TMF 727	Service Usage Management API REST Specification https://www.tmforum.org/oda/open-apis/directory/service-usage-management-api-TMF727/v5.0
[25]	TMF 771	Resource Usage Management User Guide https://www.tmforum.org/oda/open-apis/directory/resource-usage-api-TMF771/v5.0

Ref	Doc Number	Title
[26]	RFC 7239	Forwarded HTTP Extension https://datatracker.ietf.org/doc/html/rfc7239
[27]		OpenID Connect Client-Initiated Backchannel Authentication Flow - Core 1.0 https://openid.net/specs/openid-client-initiated-backchannel-authentication-core-1_0.html
[28]		Google study on page load time statistics https://thinkwithgoogle.com/marketing-strategies/app-and-mobile/page-load-time-statistics
[29]	RFC 7523	JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants https://datatracker.ietf.org/doc/html/rfc7523
[30]	TMF 936	Open Gateway Product Catalog API https://www.tmforum.org/oda/open-apis/directory/open-gateway-product-catalog-api-TMF936/v5.0
[31]		CAMARA Consent Info API v0.2.0 / code / API_definitions / consent-info.yaml
[32]	GSMA PRD OPG.04	East-Westbound Interface APIs
[33]	TMF S026	TMF document defining Use Case for Commercializing CAMARA APIs with Operate APIs https://www.tmforum.org/resources/technical-specification/tmfs026-use-case-commercializing-camara-apis-with-operate-apis-v1-0-0/
[34]		CAMARA Consent Management API v0.1.0 / code / API definitions / consent-management.yaml
[35]	RCC.07	Rich Communication Suite - Advanced Communications Services and Client Specification
[36]	NG.131	RCS Verification Authority API

1.6 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 [2] and clarified by RFC8174 [3], when, and only when, they appear in all capitals, as shown here.

2 High Level Architecture

2.1 General

GSMA PRD OPG.02 [1] defines the Operator Platform (OP) architecture framework and requirements. An Open Gateway (OGW) Platform is a specific realisation (or deployment option) of a subset of the OP functions. Therefore, the definitions, architecture and requirements provided in [1] apply.

An Open Gateway (OGW) Platform exposes Service APIs (defined by CAMARA), Operate APIs (defined by TM Forum) and possibly other APIs so third-party services can consume them in a secure, consistent and monetisable way.

2.2 Detailed Architecture / Components View

Figure 1 presents the high-level architecture and canonical functions used in an OGW Platform.

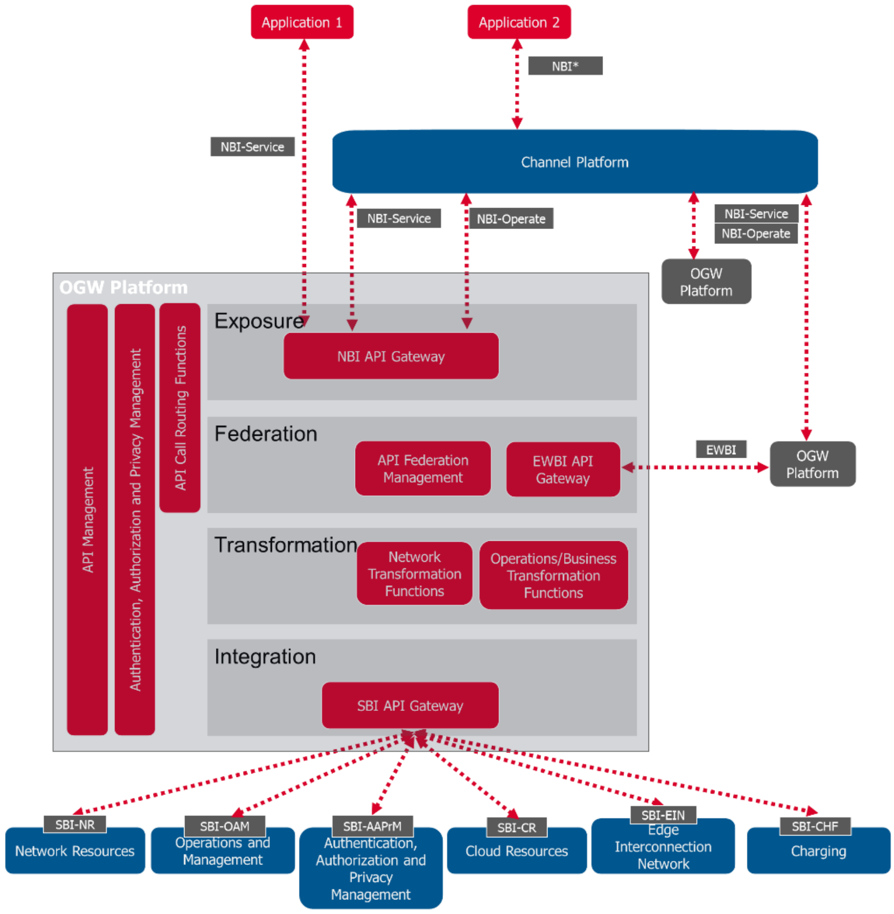


Figure 1: OGW Platform - High level architecture and functions

As shown in Figure 1, the functions can be grouped into four functional levels: a) Exposure, b) Federation, c) Transformation and d) Integration Functions. It is worth mentioning that some common functions can span multiple functional levels (see e.g., API Management in Figure 1).

The functional components in Figure 1 may be deployed in a distributed manner (as an architectural pattern that goes beyond monolithic realisations) enabling also flexible functional composition (for instance, if federation is not a scenario to be considered, the Federation-related functionalities do not need to be deployed).

Note: Alignment with the GSMA OPG on the harmonised architecture might be needed as some of it might have to be reflected in GSMA PRD OPG.02 [1] as well.

2.2.1 Common Functions

The following functions may be applicable to all APIs.

2.2.1.1 API Management Functions

Providing (among others) the following functions:

- API Catalogue
- Application Provider management
- Application Onboarding
- API Subscription management
- API Usage management
- API Monitoring
- API SLA management
- API Provider management
- API Lifecycle management
- API Access Policy management

2.2.1.2 API Gateway Functions

API Gateway Functions are available in all of the interfaces in the architecture. They include (among others) the following functions:

- API Registry
- API Access Control / Security enforcement
    - Authentication (see below clause 2.2.1.3)
    - Authorisation (see below clause 2.2.1.3)
    - Plan control
- API Usage Data Generation
- API Logging and Tracing
- API Metrics Generation
- API Audit Logging
- API Traffic Management
    - Spike arrest
    - Usage throttling / Rate limiting
    - Traffic prioritisation
- Interface translation
    - Format translation (e.g., from XML to JSON)
    - Protocol translation (e.g., from SOAP to REST)
- Caching

2.2.1.3 Authentication, Authorisation and Privacy Management

Providing (among others) the following:

- Authentication and Authorisation (server side).

- Identity Management (if applicable)
- Privacy Management (if applicable)
    - key and certificate management
    - whenever Consent is the applicable legal basis:
        - Consent enforcement point (for NBI or EWBI)
        - Caching relevant Consent configuration retrieved from the Privacy Management function in the Operator domain (if allowed by local regulations)
        - Triggering Consent capture by the Privacy Management in the Operator domain
        - In federated scenarios, triggering Consent capture by the Privacy Management function in the Operator domain of the federated partner

Note: OGW Platform may rely on procedures regarding Authentication (including Identity) / Authorisation / Privacy management through interfaces already in place in the Operator domain which are mapped to the SBI-AAPrM.

2.2.1.4 API Call Routing Functions

The API call routing functions provides (among others) the following:

- Load balancing
- Telco Finder service which is responsible for resolving the Operator associated with a target user identifier (e.g. based on a specific phone number) and returning information about the associated Operator

2.2.1.4.1 Telco Finder

This document describes the Telco Finder components within the Open Gateway Platform architecture. The Telco Finder is responsible for resolving the Operator associated with a target user identifier (e.g. the Operator that owns a specific phone number) and returns information about the associated Operator (i.e. Operator ID, API root URL, authorisation provider data). It is exposed as a RESTful API.

2.2.1.4.1.1 Service Overview

Telco Finder is an integral component of the Open Gateway Platform architecture designed to provide information about the Operator associated with a user, as well as the relevant endpoints required for performing operations related to that Operator.

Telco Finder can be implemented by any partner, such as an Aggregator, an Operator, or a third-party commercial service. The consumers of the Telco Finder, such as Aggregators or Operators, enter into contractual agreements with the Telco Finder to access and utilise its services. The Telco Finder internal functionality is also contingent on commercial agreements with partner Operators who agree to share routing data with the Telco Finder. This routing data serves as the foundational element of its internal logic.

Telco Finder has two main functions:

- Resolution of User identifier to Operator identifier: The primary function of Telco Finder is to map a user's identifier to the corresponding Operator identifier. This process is managed by an internal Resolution component that queries both internal and external lookup data to achieve the mapping.

- Retrieval of Operator URLs and endpoints: Upon obtaining the Operator identifier, Telco Finder has the capability to retrieve the associated data and the URLs exposed by that Operator. This can be achieved in two ways:
  - Internal Storage: Telco Finder may store the necessary information and provide it directly.
  - Delegation: Telco Finder can delegate the retrieval of information to another Telco Finder, which will return the required data. This approach is particularly beneficial in multi-brand scenarios.

2.2.1.4.1.2 Telco Finder API Interface

Telco Finder is exposed as a RESTful API in OAS format – the specification can be found in the Annex A.1.

The specification contains detailed usage information.

It provides a POST /search endpoint to retrieve information about the Operator associated with a given user identifier. At a fundamental level, it accepts a user identifier as an input and responds with an operatorId. Optionally, based on input control flags, it also returns the Operator's API root URL and the Operator's authorisation server discovery endpoint. For use in regions with mobile number portability, the interface also provides an input parameter that controls the internal search mode of Telco Finder.

2.2.1.4.1.2.1 Request

Consumers invoke the /search endpoint to discover the owning Operator of a particular user. The JSON request payload can contain the following fields:

- **target**: This is a mandatory object field whose purpose is to convey user information. This object comprises of multiple optional fields to identify a target user **(phoneNumber, ipv4Address, ipv6Address)**.
- **includeApiRoot**: This optional boolean field is used to control whether the response should contain the Operator’s API root URL. If the field is not included in the request, the default value is false.
- **includeAuthProviderConfiguration**: This optional boolean field is used to control whether the response should contain the Operator’s authorisation server discovery endpoint. If the field is not included in the request, the default value is false.
- **portabilitySearchMode**: This optional enum field is used to control the search behaviour of the Telco Finder in regions with mobile number portability. It supports 2 values: **SHALLOW** and **DEEP**. The shallow option instructs Telco Finder to search only its internal records (e.g. cache). This method can be preferred to avoid higher monetary costs associated with extended searches. The full search triggers a comprehensive search against all external systems, providing more thorough results at a potentially higher cost and ensuring up-to-date information by bypassing stale cached data. If the field is not included in the request, the default value is implementation specific.

Example payloads are available in sub-sections below.

2.2.1.4.1.2.2 Response

The data returned by Telco Finder:

- **Operator ID:** The Operator to which the target user belongs. This field will always be returned in the response.
- **API Root of the Operator:** The root URL of the API Gateway managed by the owning Operator. This field is false by default but can be included in the response by setting the request field **includeApiRoot** to true.
- **Authorisation server discovery endpoint:** The discovery endpoint of the Operator's authorisation server. This is a standardised URL in OpenID Connect [12] and OAuth 2.0 [13] that allows clients to dynamically retrieve configuration metadata about the authorisation server. This field is false by default but can be included in the response by setting the request field **includeAuthProviderConfiguration** to true.

2.2.1.4.1.2.3 Rationale for optional fields

The **includeApiRoot** and **includeAuthProviderConfiguration** request fields allow consumers to optimise the response based on their specific needs. By default, only minimal information is returned (Operator ID) to minimise computational costs. If a consumer is interested in further information, they can set the aforementioned field values to true.

2.2.1.4.1.2.4 Examples – Request and Response

2.2.1.4.1.2.4.1 *Example A*

Searching for a telephone number - using only mandatory input fields:

```
POST /telco-finder/v1/search HTTP/1.1
HOST: api.operator.com
Content-Type: application/json

{
  "target": {
    "phoneNumber": "+447709558432"
  }
}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "operatorId": "OPERATOR_ID"
}
```

2.2.1.4.1.2.4.2 *Example B*

Searching for an IP address - using optional boolean input fields to control response granularity. These booleans instruct the Telco Finder to also return the Operator's API root URL and the Operator's authorisation server discovery endpoint.

```
POST /telco-finder/v1/search HTTP/1.1
HOST: api.operator.com
Content-Type: application/json

{
  "target": {
    "ipv4Address": {
      "publicAddress": "84.125.93.10",
      "publicPort": 59765
    }
  },
  "includeApiRoot": true,
  "includeAuthProviderConfiguration": true
}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "operatorId": "OPERATOR_ID",
  "apiRoot": "https://example.operator.com",
  "authProviderConfiguration":            "https://auth.operator.com/.well-
known/openid-configuration"
}
```

2.2.1.4.1.2.4.3 *Example C*

Searching for a phone number and specifying a portability search mode.

```
POST /telco-finder/v1/search HTTP/1.1
HOST: api.operator.com
Content-Type: application/json

{
  "target": {
    "phoneNumber": "+447709558432"
  },
  "portabilitySearchMode": "SHALLOW"
}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "operatorId": "OPERATOR_ID",
}
```

2.2.1.4.1.2.4.4 *Example D*

Searching for a phone number and using all possible input parameters

```
POST /telco-finder/v1/search HTTP/1.1
HOST: api.operator.com
Content-Type: application/json
```

```
{
  "target": {
    "phoneNumber": "+447709558432"
  },
  "includeApiRoot": true,
  "includeAuthProviderConfiguration": true
  "portabilitySearchMode": "SHALLOW"
}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "operatorId": "OPERATOR_ID",
  "apiRoot": "https://example.operator.com",
  "authProviderConfiguration":            "https://auth.operator.com/.well-
known/openid-configuration"
}
```

2.2.1.4.1.3 Functional components

The following diagram demonstrates the core internal components of the Telco Finder:

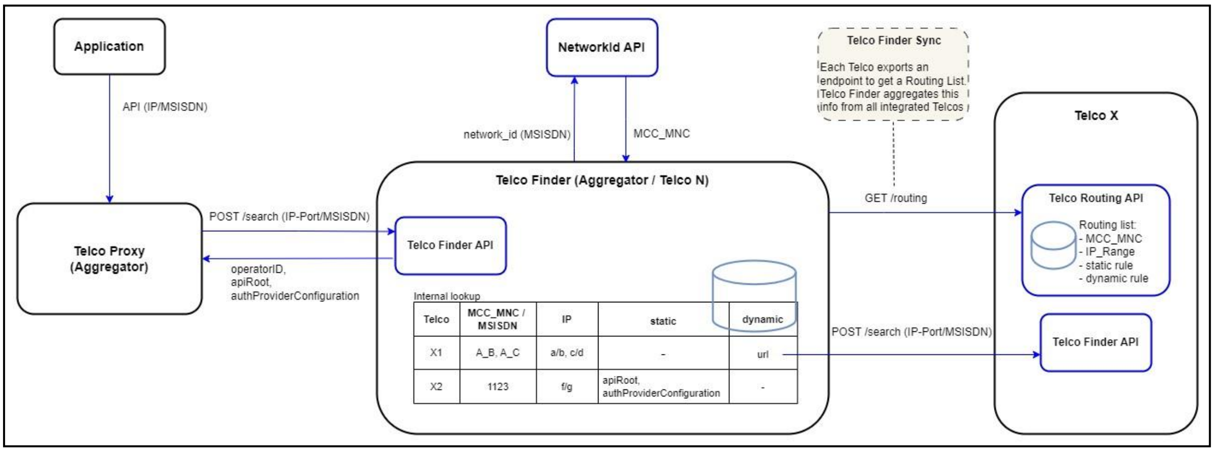


Figure 2: Telco Finder – main components

The main components are as follows:

- Telco Finder API: Telco Finder exposes an API that is consumed by Aggregators seeking to identify the owning Operator for a specific user identifier. This identifier is either a phone number or an IP address, and it is explicitly provided in the API request by the API Consumer.
- Internal lookup data storage: Internally, Telco Finder maintains data storage representing routing information. The format and structure of the data storage is implementation specific. However, the data contextually represents:
    - A list of Operators and their MCC/MNCs
    - Routing data including:

  - IP ranges in CIDR format
  - MSISDN prefixes owned by the Operator in regions without mobile number portability.

- The Telco Finder consumes a "**GET /routing**" endpoint that is exposed by each partner Operator. This endpoint returns routing data that belongs to the Operator. Telco Finder periodically retrieves data from these Operator endpoints and consolidates it within its internal lookup data storage.
- Each data element stored internally and received through the "GET /routing" endpoint is categorised as either "Static" or "Dynamic":

  - Static Data: Served directly by a single Operator's API gateway.
  - Dynamic Data: Requires a secondary request to a separate Telco Finder instance belonging to the specific Operator. This is particularly useful in scenarios involving multi-brand Operators where various data sets might be managed by different brands.

- The Telco Finder also consumes a "**NetworkID API**", which is responsible for returning the owning Operator of a MSISDN in regions with mobile number portability. The implementation of this API is region-specific and needs to be agreed by all parties within a federation/aggregation.

2.2.1.4.1.4 Telco Routing API

Each partner Operator is required to expose a standardised "GET /routing" API. Telco Finder periodically fetches data from this API. This data includes IP ranges, MSISDN prefixes, and MCC/MNCs associated with each Operator. The retrieved data is then aggregated to form a routing ruleset used to resolve ownership of user identifiers.

It is exposed as a RESTful API in OAS format – the specification can be found in the Annex A.2.

2.2.1.4.1.4.1 Routing Rules

The API returns an array of JSON objects where each object represents a rule. Each rule contains the following parameters:

- **ipv4:** array of strings in CIDR notation. List of IP V4 ranges (example: 23.124.1.200/20).
- **ipv6:** array of strings in CIDR notation. List of IP V6 ranges (example: ff22:0:0:ab:23:1a:346:7332/64).
- **msisdnPrefix:** array of strings representing an MSISDN prefix starting by the country code (example: +100234)
- **network:** array of strings representing an MCC_MNC code (example: 23401)
- **static:** A JSON Object that represents that the aforementioned data is static (served by a single gateway). It contains the following string fields:

  - operatorId: The ID of the Operator
  - apiRoot: The root URL of the Operator's API Gateway
  - authProviderConfiguration: The discovery endpoint of the Operator's authorisation server

- **dynamic**: A JSON Object that represents that the aforementioned data is dynamic and that it requires a call to a second level Telco Finder instance (e.g. to resolve multi-brand routing). It contains the following string fields:
    - telcoFinder: URL of the Operator’s Telco Finder
    - authProviderConfiguration: The discovery endpoint of the Operator's authorisation server

Each Rule must contain at least one of ipv4 / ipv6 / msisdnPrefix / network member. Each rule must contain either one of static or dynamic.

2.2.1.4.1.4.1.1 *Examples*

2.2.1.4.1.4.1.1.1 *Static routing rule*

```
  {
    "ipv4": [
      "23.124.1.200/20",
      "34.231.2.120/22"
    ],
    "ipv6": [
      "ff22:0:0:ab:23:1a:346:7332/64"
    ],
    "network": [
      "23405",
      "23411"
    ],
    "static": {
      "operatorId": "OPERATOR_ID",
      "authProviderConfiguration":        "https://auth.operator.com/.well-
known/openid-configuration",
      "apiRoot": "https://example.operator.com"
    }
  }
```

2.2.1.4.1.4.1.1.2 *Dynamic routing rule*

```
  {
    "network": ["23405","23411"],
    "dynamic": {
      "authProviderConfiguration":        "https://auth.operator.com/.well-
known/openid-configuration",
      "telcoFinder": "“https://apis.operator.com/telco-finder/v1”"
    }
```

2.2.1.4.1.4.1.1.3 *Dynamic routing rule with MSISDN prefixes*

```
  {
    “msisdnPrefix”: [“+100234", "+100333"],
    "dynamic": {
      "authProviderConfiguration":        "https://auth.operator.com/.well-
known/openid-configuration",
      "telcoFinder": "https://apis.operator.com/telco-finder/v1"
    }
  }
```

2.2.1.4.1.4.1.1.4 *Telco Routing API request and response with both static and dynamic rules*

Request

```
GET /routing HTTP/1.1
Host: apis.operator.com
```

```
Accept: application/json
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "ipv4": [
      "23.124.1.200/20",
      "34.231.2.120/22"
    ],
    "ipv6": [
      "ff22:0:0:ab:23:1a:346:7332/64"
    ],
    "static": {
      "operatorId": "OPERATOR_ID",
      "authProviderConfiguration":         "https://auth.operator.com/.well-
known/openid-configuration",
      "apiRoot": "https://example.operator.com"
    }
  },
  {
    "network": [
      "23405",
      "23411"
    ],
    "dynamic": {
      "authProviderConfiguration":         "https://auth.operator.com/.well-
known/openid-configuration",
      "telcoFinder": "“https://apis.operator.com/telco-finder/v1”"
    }
  }
]
```

2.2.1.4.1.5 Network Id API

When a country allows number portability, the Operator owning a MSISDN cannot be determined by its prefix alone. Instead, the network ID (MCC_MNC) must be resolved through an API that should be available for each region.

The implementation of this API must be determined on a region-by-region basis within a federation. For instance, it could be a shared implementation between Operators, or provided by a market champion, or procured as a commercial third-party service.

The specification for this API can be found in the Annex A.3. It defines an operation for requesting network IDs:

```
POST /resolve-network-id HTTP/1.1
Content-Type: application/json
Accept: application/json

{
    "phoneNumber": "+34666777888"
}
```

The response is an MCC_MNC:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "networkId": "21407"
}
```

2.2.1.4.1.5.1 Commercial MSISDN lookup services

There are a number of commercial services that maintain extensive databases of MSISDNs and can be used to retrieve the home Operator. This is particularly useful where there is mobile number portability but there is no national MNP database. Coverage can be very large. Selection of any particular service provider is the decision of the aggregator. A commercial service or services may be used as the final choice when other methods have failed or as an initial lookup service for speed and convenience.

2.2.1.4.1.6 Operator resolution

Identifying the owning OGW Platform for any Subscriber and device is performed through a routing mechanism that involves all of the aforementioned components. The routing mechanism is reliant on core routing data - this data in turn is dependent on contractual agreements with Operators to share their routing data via a Telco Routing API. Note that in addition to these "supplier" agreements, there are also consumer agreements in place, where a consumer (such as an Aggregator) agrees to commercial terms to access the Telco Finder API.

The Routing API of each Operator is polled and aggregated to form an internal routing table. Per country, the Telco Finder aggregates the Operator routing tables to resolve user identifiers into the Operator brand and API endpoints.

A routing rule is composed by a condition and a resolution action. The condition is based on an ID range. The condition is satisfied if the actual user identifier belongs to one of the ranges:

- **IP Ranges** represented in the CIDR notation as defined in RFC 4632 [14]. For example, 80.23.124.200/22 for IPv4 or ffff:0:0:89fa:cdea:2341:2ds1f:ffff/20 for IPv6.
- **MSISDN prefixes**, for countries without phone number portability. For example +100234.
- **Network identifier**: MCC and MNC components of the IMSI as defined in TS 23.003 [15], for countries with phone number portability. For example, 22401. An MSISDN is resolved into the owning network by the Telco Finder using per-country specific Network Id API.

Mobile Country Code (MCC)	Mobile Network Code (MNC)	Mobile Subscriber Identification Number (MSIN)
3 digits	2 or 3 digits	up to 9 or 10 digits (max IMSI length 15 digits)

Table 1: IMSI (International Mobile Subscription Identity) structure

As alluded to in earlier sections, there are two types of routing resolution actions:

- **Static Routing**: In this simplest case, the routing rule directly maps user identifiers to endpoint URLs. All user identifiers within a specified range belong to the same brand and are served by the same endpoint.
- **Dynamic Routing**: When a user identifier range is shared among different brands, each brand exposes its own API endpoints. In this scenario, the routing rule maps the user identifier range to a second-level Telco Finder instance provided by the Telco Operator. The initial Telco Finder calls this interface to resolve the appropriate endpoint.

2.2.1.4.1.6.1 IP address lookup sequence diagram

For IP routing, the routing rule conditions utilised are the ipv4 and ipv6 ranges.

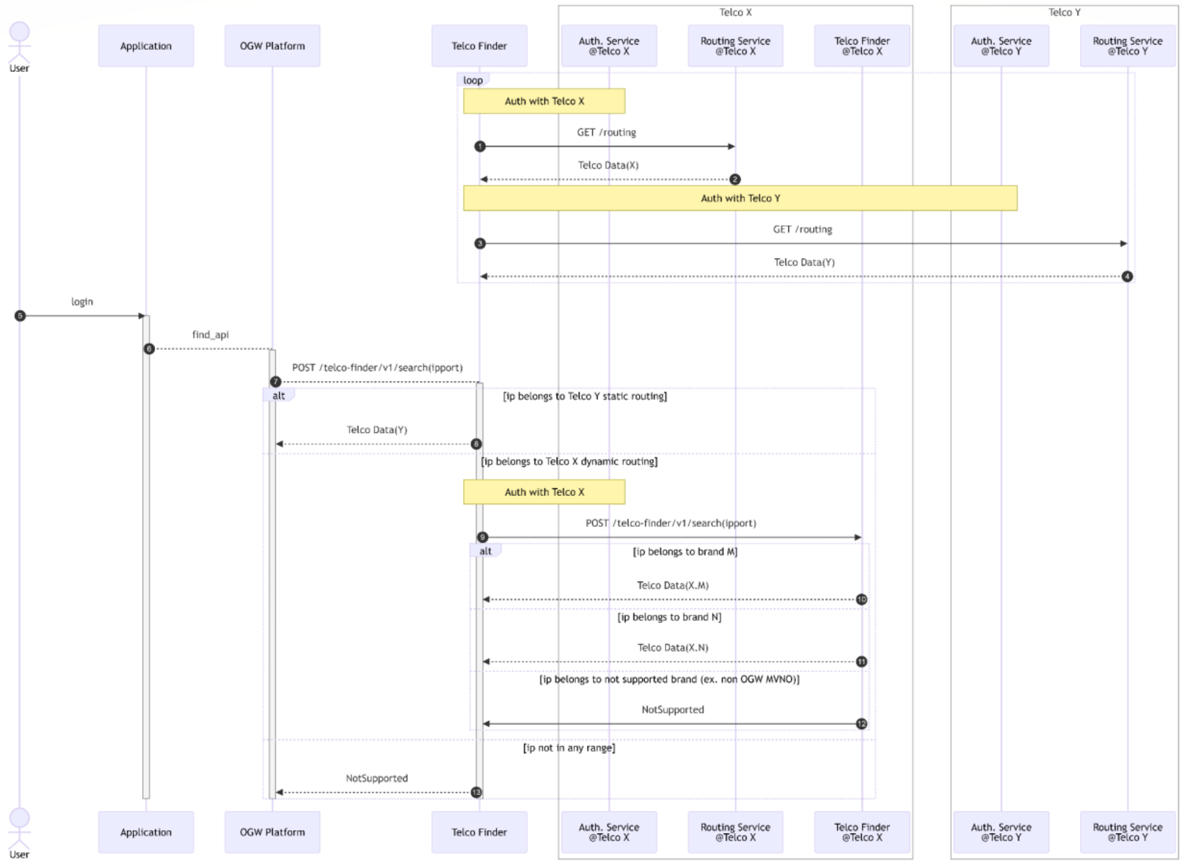


Figure 3: IP address lookup sequence diagram

1. Periodically (every x minutes), the Telco Finder consumes the Routing API of each Operator and aggregates the data into its internal lookup datastore (steps 1-4)
2. Each time a user logs in an Application (step 5), the Application requests that the OGW Platform returns the API endpoints for that user (where the user is identified by the calling ip-port of the device where the Application is running; this is observed by the OGW Platform) – steps (6-7).
3. The Telco Finder looks for the IP Address Range of the IP address of the device and determines:

a) The IP address belongs to Telco Y and the routing is static (Telco Y provided directly the API links). The Telco Finder then returns the Telco Y API links to the OGW Platform (step 8).
b) The IP address belongs to Telco X and the routing is dynamic through a second level Telco Finder. The initial Telco Finder then contacts the second level Telco Finder to resolve the ip-port (step 9). The Telco Finder of Telco X may return:

    i. If ip-port belongs to one of the Telco X brands, returns the brand api links (step 11)
    ii. If ip-port belongs to a brand which does not support CAMARA APIs returns a NotSupported error (step 12).

c) The IP address does not belong to any of the registered telcos and returns a NotSupported error (step 13).

2.2.1.4.1.6.2 MSISDN lookup sequence diagram

For MSISDN routing, the routing rule conditions utilised are:

- msisdnPrefix – for countries without number portability.
- network – a list of MCC_MNC identifiers for countries with number portability.

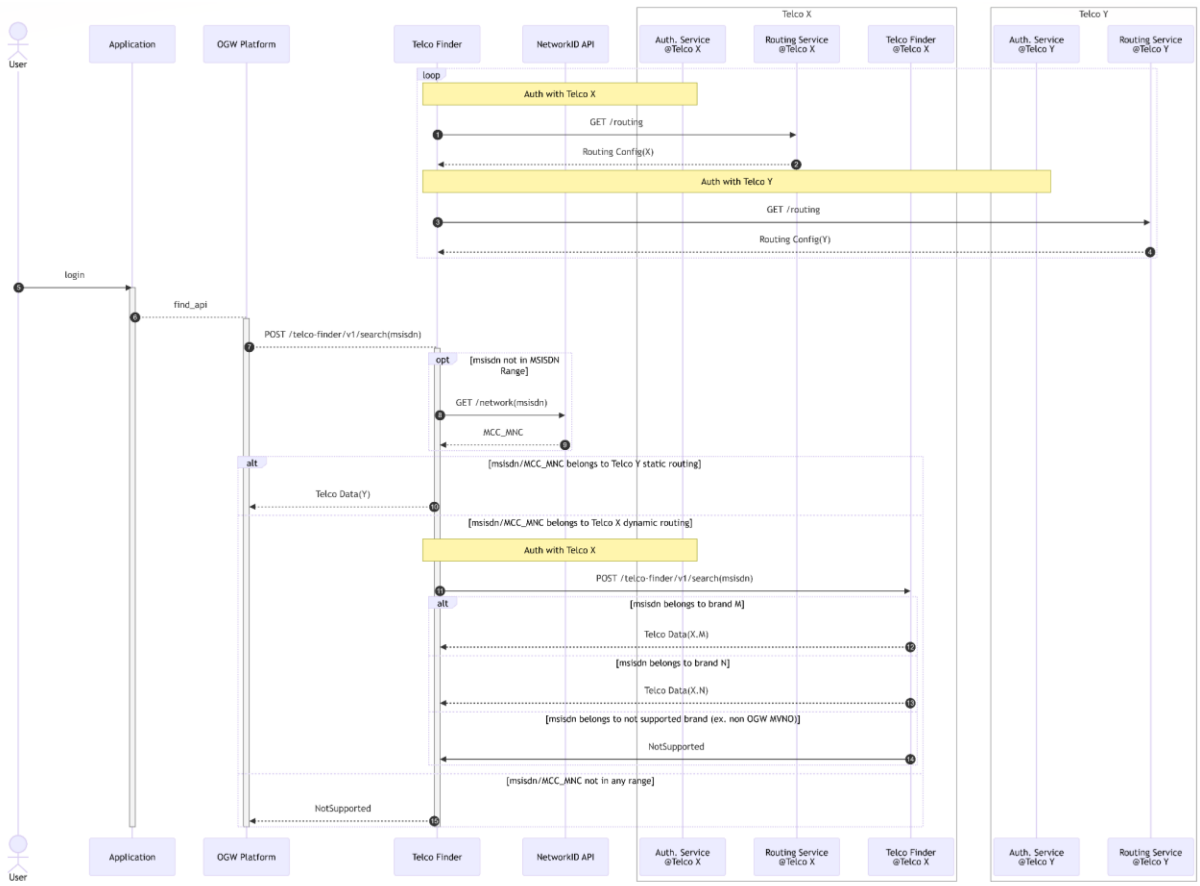


Figure 4: MSISDN lookup sequence diagram

1. Periodically (every x minutes), the Telco Finder consumes the Routing API of each Operator and aggregates the data into its internal lookup datastore (steps 1-4).
2. Each time a user logs in an Application (step 5), the Application requests that the OGW Platform returns the API endpoints for that user (identified by its msisdn) (steps 6-7)
3. The Telco Finder looks within the routing table for the Telco routing data based on:

    a) Whether the MSISDN belongs to a MSISDN prefix within its lookup data. If not:
    b) The Telco Finder contacts the NetworkId API and requests the MCC_MNC of the network belonging to the msisdn (steps 8-9).
    c) No routing record is found so a NotSupported error is returned.

4. The Telco Finder gets the resolved routing data:

    a) If MSISDN or MCC_MNC belongs to Telco Y and the routing is static (Telco Y directly provided the API links), then the Telco Y API links are returned to OGW Platform (step 10).

b) If MSISDN or MCC_MNC belongs to Telco X and the routing is dynamic, then a request is made to the provided second level Telco Finder URL to resolve the MSISDN (step 11). The Telco Finder of Telco X may return:

   i. If MSISDN belongs to one of the Telco X brands, the brand API links are returned (steps 12-13).
   ii. If MSISDN belongs to a brand which does not support CAMARA APIs then a NotSupported error is returned (steps 14-15).

2.2.1.4.1.6.3 Multi-brand lookup

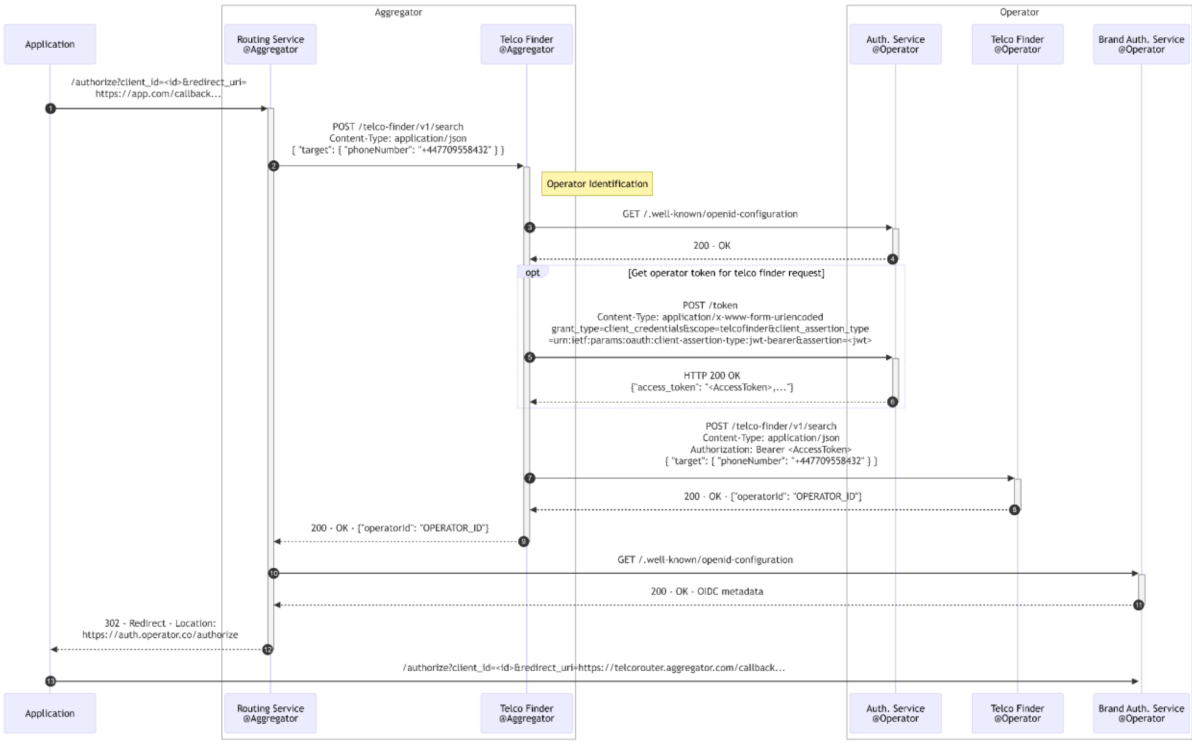


Figure 5: Multi-brand lookup

Figure 5 demonstrates how the component delegates obtaining information related to the user to another Telco Finder in a multi-brand scenario.

2.2.1.4.2 Security

The following APIs shall be secured by the client credentials flow of OAuth 2.0 [16]:

- Telco Finder API interface
- Telco Routing API
- Network Id API

The client authentication method for both Telco Routing and Telco Finder is based on private_key_jwt, as defined in OIDC Client Authentication [17].

2.2.1.4.3 Path definition

Following CAMARA API Design Guidelines [18], the API paths shall take the following format: https://host:port/<api>/<version>/<resource>

For example:

- Telco Finder: https://apis.router.com/telco-finder/v0/search
- Telco Routing: https://apis.telco.com/telco-routing/v1/routing
- Network Id: https://apis.network.com/network-id/v0/resolve-network-id

2.2.1.4.4 API call routing when Subscriber information is not applicable

In multi-network targeting scenarios in which an OGW Platform aggregates other OGW Platforms or federates with other OGW Platforms, for API calls that are not associated to a specific Subscriber (e.g., Population Density Data, Region Device Count, Dedicated Networks, Application Profiles, etc), mechanisms have to be in place to enable the routing of the incoming API calls. In such scenarios, the Telco Finder is not a suitable solution to find the right target OGW Platform.

To route an API call in such scenarios, the following needs to be obtained:

- Information about the authorization endpoints for the possible target OGW Platforms
- A candidate set of target Operator(s) (depending on the Application and use cases), and
- Operator selection mechanisms (depending on the Application and use cases)

2.2.1.4.4.1 Obtaining authorization endpoints information

During the interconnection / integration of an OGW Platform with other platforms, the Operator is expected to exchange information regarding certificates, API root(s) and authorization endpoints.

2.2.1.4.4.2 Candidate set of target OGW Platforms (via operational interactions)

2.2.1.4.4.2.1 Product Catalogue

Information regarding the geographical availability of an API can be leveraged for routing API calls which are associated to a geographical service area. Currently, the description field of `productOfferingPrice` [30] allows including a list of countries in ISO-3166 Alpha-2 format to indicate where the API is available and the associated price.

Note: further alignment on supporting mechanism and terminology within TMF936 is FFS.

2.2.1.4.4.2.2 Application Provider onboarding

As stated in [19], the Application Provider may select a set of Operators at onboarding time. If no explicit selection is made, by default, the set of all possible Operators is used. Within the context of the application provided by the Application Provider, the OGW Platform (acting as an Aggregator or Leading OGW Platform) should route API calls respecting the selection performed by the Application Provider. The authorization endpoint information was obtained during interconnection / integration time of each Operator’s OGW Platform.

2.2.1.4.4.3 Operator selection mechanisms

For a specific Application region, and for a set of possible target Operators, whenever no single Operator is found to be targeted, the OGW Platform (acting as an Aggregator or Leading OGW Platform) could:

- Target one Operator (out of the candidate set)
  - Whenever possible, the target Operator shall be selected based on the Application logic or use case context. For instance, if the Application was developed for an event that is sponsored by one Operator (and no administrative selection of Operator was performed while onboarding), then the OGW Platform should route API calls to the OGW Platform of the sponsor Operator
  - When no selection can be made based on the Application logic or use case context, the OGW Platform can select the target Operator based on criteria like:
    - Load balancing
    - Fairness
    - Accuracy
    - Availability
- Target all the Operators (in the candidate set)
  - Whenever the use case allows / mandates it, the OGW Platform (acting as an Aggregator or Leading OGW Platform) shall forward an API call to all the candidate Operator OGW Platforms. This may be done for instance when getting aggregated information from all the Operators will improve the response accuracy towards the Application.

Note: methods for aggregating responses from other OGW Platforms are out-of-scope of this document.

2.2.1.4.4.4 Example decision flow for routing API calls when Subscriber information is not applicable

Figure 6 presents an example on how the concepts described in the previous sections enable making decisions about where to route an API call when Subscriber information is not applicable.

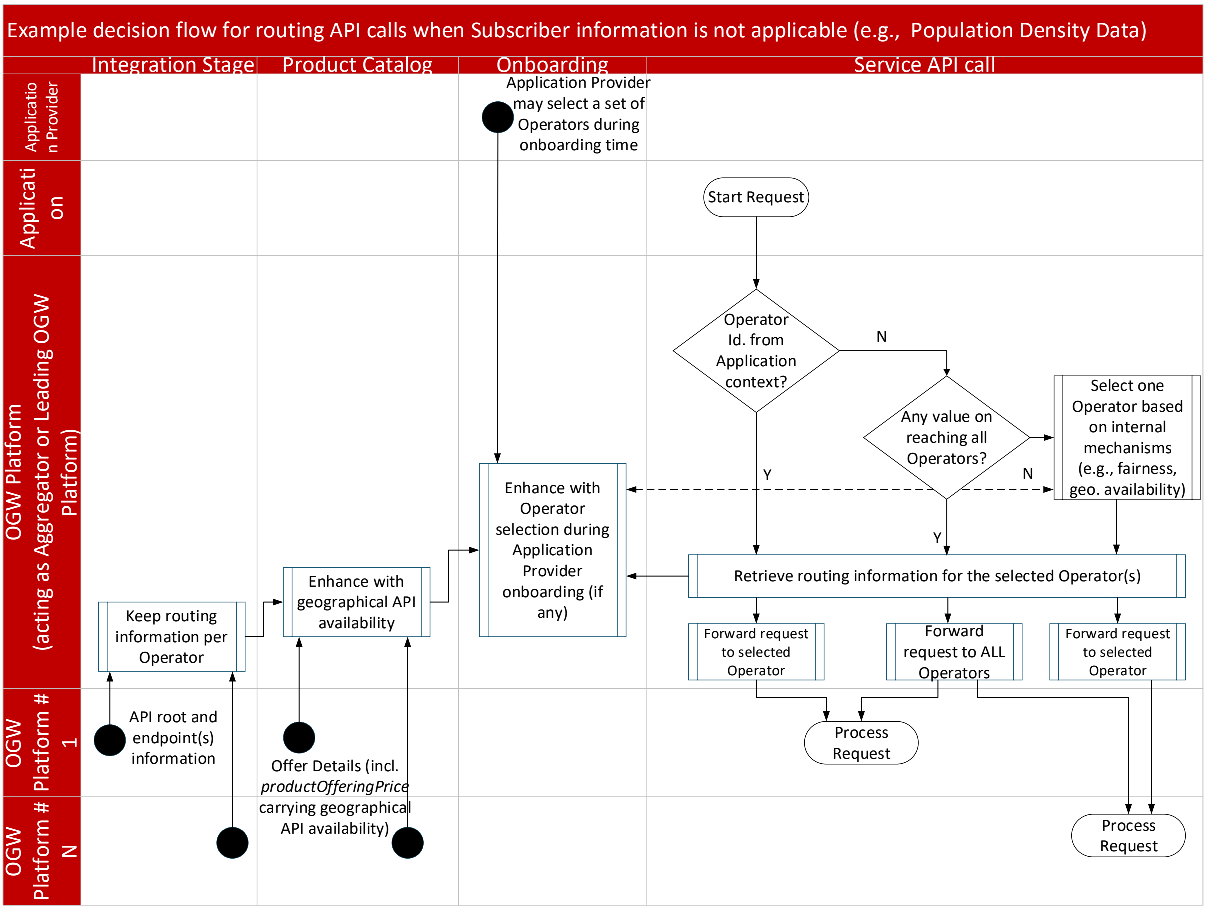


Figure 6: Example decision flow for routing API calls when Subscriber information is not applicable

2.2.1.4.5 API call routing when Subscriber interaction is not needed

There may be cases in which the functionality provided by an API is associated with End-User information, but the authorization request does not require the End-User interaction at the time of the request, for instance:

- A legal basis that does not require explicit End-User interaction to allow personal data processing is applicable (e.g., Legitimate Interest).
- The End-User has previously granted Consent, or
- Even though personal data could be handled, End-Users cannot exercise their rights through mechanisms such as opt-in and/or opt-out.

For the above scenarios an access token needs to be bound to a user identifier to enable routing procedures performed by the Channel Platform or by an OGW Platform aggregating other OGW Platforms. Therefore, it is recommended to leverage the JWT Bearer flow, as captured in clause 3.1.1.3, and especially the `sub` claim in the JWT assertion, which must uniquely identify the End-User in the Operator's domain.

For cases when no End-User information is applicable, the routing considerations are captured in clause 2.2.1.4.4.

2.2.2 Exposure Functions

The Exposure functions enable exposing Service APIs (to Applications or Channel Platforms) via the NBI-Service interface and Operate APIs (to Channel Platforms) via the NBI-Operate interface. The termination points for the NBI-* API calls are provided by the corresponding API Gateway function as described below.

Note: The names NBI-Service and NBI-Operate may change in the future based on further discussions taking place across several groups.

2.2.2.1 NBI API Gateway

In addition to the common API Gateway functions provided in above clause 2.2.1.2, the NBI API Gateway supports (among others) the following functions:

- Providing termination points for Service API calls from Applications (owned by Application Providers) or Channel Platforms (owned by an Aggregator or a 3rd party)
- Mapping to Transformation functions / SBI Gateway
- Routing to API Federation Management function / EWBI Gateway in case of API call Federation

Additionally, the NBI API Gateway supports:

- Providing termination points for Operate API calls from Channel Platforms
- Mapping to Operations and Business Transformation Functions / SBI Gateway

2.2.3 Federation Functions

2.2.3.1 EWBI API Gateway

In addition to the common API Gateway functions provided in above clause 2.2.1.2, the EWBI API Gateway supports (among others) the following functions:

- Providing termination of EWBI API calls to/from other Operator exposure platforms
- Routing to API Federation Management function to reach Network Transformation function to SBI API Gateway for API calls forwarded from another OP/OGW Platform.
- Routing to API Federation Management function to reach NBI API Gateway in case of federated API responses.

2.2.3.2 API Federation Management

Providing (among others) the following services:

- Handling connectivity aspects among Operators in federated environments
  - For instance, providing Heartbeat/Keep-Alive mechanisms over the EWBI

2.2.4 Transformation Functions

2.2.4.1 Network Transformation Functions

Providing (among others) the following services:

- Transformation Functions for the realisation of the Service APIs in the lower levels of the architecture (e.g., as in GSMA PRD OPG.09 [4])

2.2.4.2 Operations and Business Transformation Functions

Providing (among others) the following services:

- Transformation Functions for the realisation of the TM Forum Operate APIs in the lower levels of the architecture (e.g., on the SBI-OAM interface)

2.2.5 Integration Functions

2.2.5.1 SBI API Gateway

In addition to the common API Gateway functions provided in above clause 2.2.1.2, the SBI API Gateway provides (among others) the following functions:

- Termination of the SBI towards:
  - Network Resources (SBI-NR)
  - Operations and Management systems (SBI-OAM)
  - Authentication, Authorisation and Privacy Management in Operator domain (SBI-AAPrM)
  - Cloud Resources (SBI-CR)
  - Edge Interconnection Network (SBI-EIN)
  - Charging (SBI-CHF)

2.2.6 Other considerations

- An OGW Platform requires the integration (i.e., connectivity) with southbound services over the SBI, with federated partner’s OGW Platforms (over the EWBI), with Application and Channel Platforms (over the NBI) and with OSS/BSS (over the SBI).
- Realisation guidelines on developer portal (and associated developer services) is considered out-of-the-scope
- API billing is considered an external functionality

Whether one instance of API Gateway per interface (or only one per platform) is needed is left as a realisation option.

3 Deployment Scenarios

3.1 Aggregation model

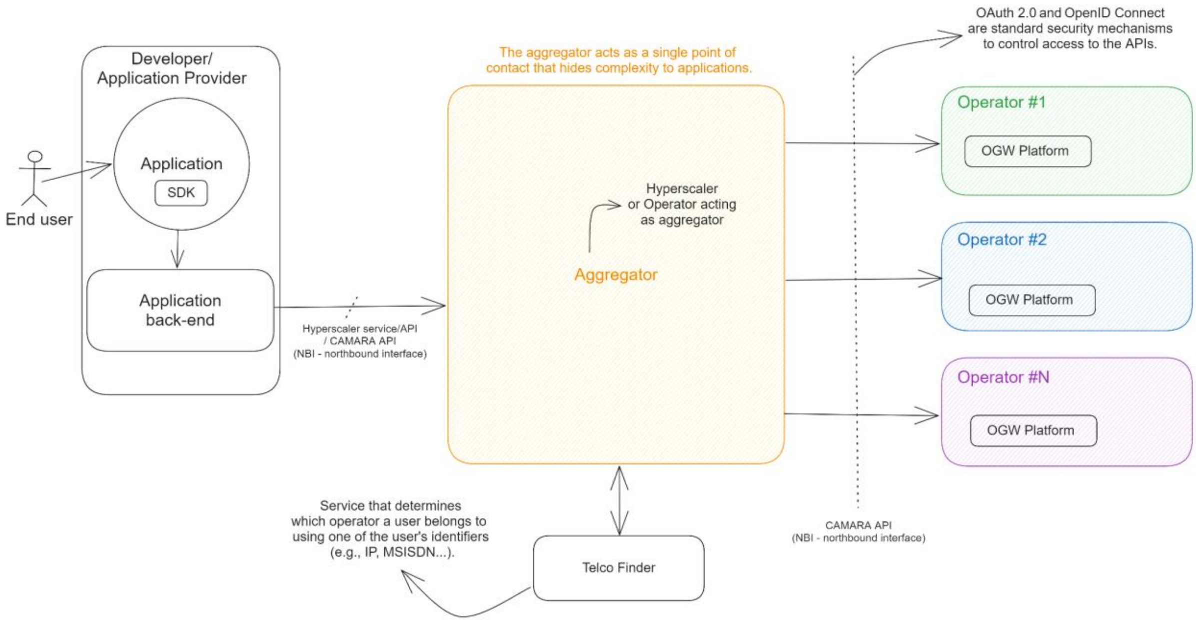


Figure 7: Aggregation model

The Open Gateway (OGW) Aggregator model consists of five different players (as defined in section 1.3):

- **End-User**: the Operator's Subscriber is usually also the End-User, but this is not always the case. For example, a parent may be the Subscriber of a mobile subscription for their child, the End-User.
- **User / Resource Owner**.
- **Developer / Application Provider**, who builds an Application that consumes Open Gateway-based services to deliver enhanced functionality to End-Users or enable new use cases.
- **Aggregator**: it aggregates the Operator's CAMARA APIs to build Open Gateway-based services and implement Operator endpoint routing based on Subscriber identification in the network.
- **Operator**: it exposes network capabilities and/or network resources through CAMARA standardised APIs and partners with Aggregators to enable the Open Gateway-based services that they offer to Application Providers.

The Aggregator acts as a single point of contact that hides complexity to Applications. It allows developers to avoid being aware of multiple Operators when building and running their Applications and eliminates the need to dispatch or orchestrate calls to them. This is important from a developer experience perspective.

The Aggregator role can be played by:

- A hyperscaler offering its own services and APIs that make use of CAMARA APIs exposed by aggregated Operators (see section 3.1.1.4) or directly exposing CAMARA APIs available at these Operators.
- An Operator acting as an Aggregator, i.e., aggregating other Operators and exposing CAMARA APIs available at those Operators.

An Aggregator needs to interact with Operators with two different roles:

- As a service consumer to call Service APIs (as described in this document section).
- As an administrator to register/unregister Applications using Operate APIs.

In Figure 7, Telco Finder is depicted as a separate entity. It may be a component of the Aggregator or offered by a different party.

3.1.1 How to consume Operator Service APIs

For an existing Application, the Aggregator can start consuming Operator Service APIs on behalf of the Application. In the Open Gateway context the Operator Service APIs are defined by CAMARA [6].

This process follows the CAMARA standard mechanisms as described in the “CAMARA Security and Interoperability Profile” [7] and “CAMARA APIs access and user consent management” [8] technical specifications.

Some Service APIs process personal data and require a “legal basis” to do so (e.g., “legitimate interest”, “contract”, “Consent”, etc). Operators must follow a privacy-by-default approach to fully comply with the spirit and letter of the different privacy regulations (e.g., GDPR), to protect user privacy. This means that an API that processes personal data may require user Consent, depending on the “legal basis” for processing that data. This Consent is given by users to legal entities to process personal data under a specific purpose [9].

An example of a Service API that requires Consent in most scenarios is the CAMARA Device Location API. This API can verify whether a mobile connection is within certain coordinates of a geographical location. The mobile connection is associated with the Subscriber’s phone number, so processing the phone number network location may require user Consent depending on the legal basis under which the information is processed (and ultimately the purpose of the use of that data). And in that case, it would be necessary to obtain the user Consent for the Application to access the “Device Location API” for a specific purpose such as “fraud prevention and detection”.

In some cases, it is not necessary to capture the Consent in the authorisation flow because the data processing is carried out under a different “legal basis” and the access is granted according to that “legal basis” (e.g. through “legitimate interest”, “contract”, etc.). For example, improving the quality of service (via CAMARA Quality-On-Demand (QoD) API) of a mobile connection under the e.g., “service provision” and “service optimisation” purposes included in the Subscriber’s contract with Terms and Conditions (T&Cs).

Figure 8 shows a very high-level summary flow of how to consume a Service API. To simplify the high-level view of the flow, some parts (e.g., Consent capture) are not shown. Take this diagram as a simplified introduction to how the flow works.

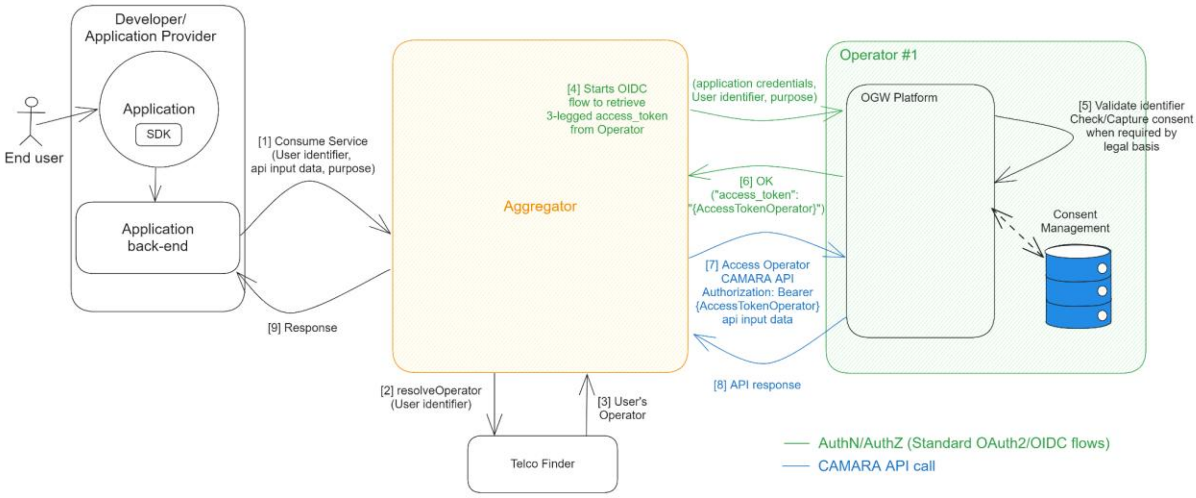


Figure 8: High level flow of the consumption of a Service API

First, the Application uses an Aggregator service which requires a specific network capability provided by an Operator. The Aggregator receives a user identifier from the Application (Step 1).

Then, the Aggregator resolves the Operator to which the user belongs, using the Telco Finder (Steps 2-3). After that, the Aggregator will know the Operator Platform it has to call, using the CAMARA API.

Before calling any Service API, it is necessary to authenticate the user with the Operator following the CAMARA standard mechanisms as described in the CAMARA Security and Interoperability Profile [7]. With this process it is possible to identify the Operator's Subscriber based on the given user identifier. And, if necessary, check for Consent and, if not yet granted, obtain it from the user. If all goes well, the Aggregator receives an OAuth 2.0 access token (Steps 4-6).

Note: It is important to remark that in cases where personal user data is processed by the API, and users can exercise their rights through mechanisms such as opt-in and/or opt-out, the use of 3-legged access tokens becomes mandatory as described in "CAMARA APIs access and user consent management" [8].

Once the Aggregator has a valid access token, this must be used to invoke the Operator Service API, which is depicted at the end of the flow (Steps 7-8).

Now, the Aggregator can confirm the Application and provide the corresponding information as per use case (Step 9).

3.1.1.1 Detailed CIBA flow (backend-based)

To get into the details of consuming Service APIs, the Figure 9 describes the entire CIBA flow.

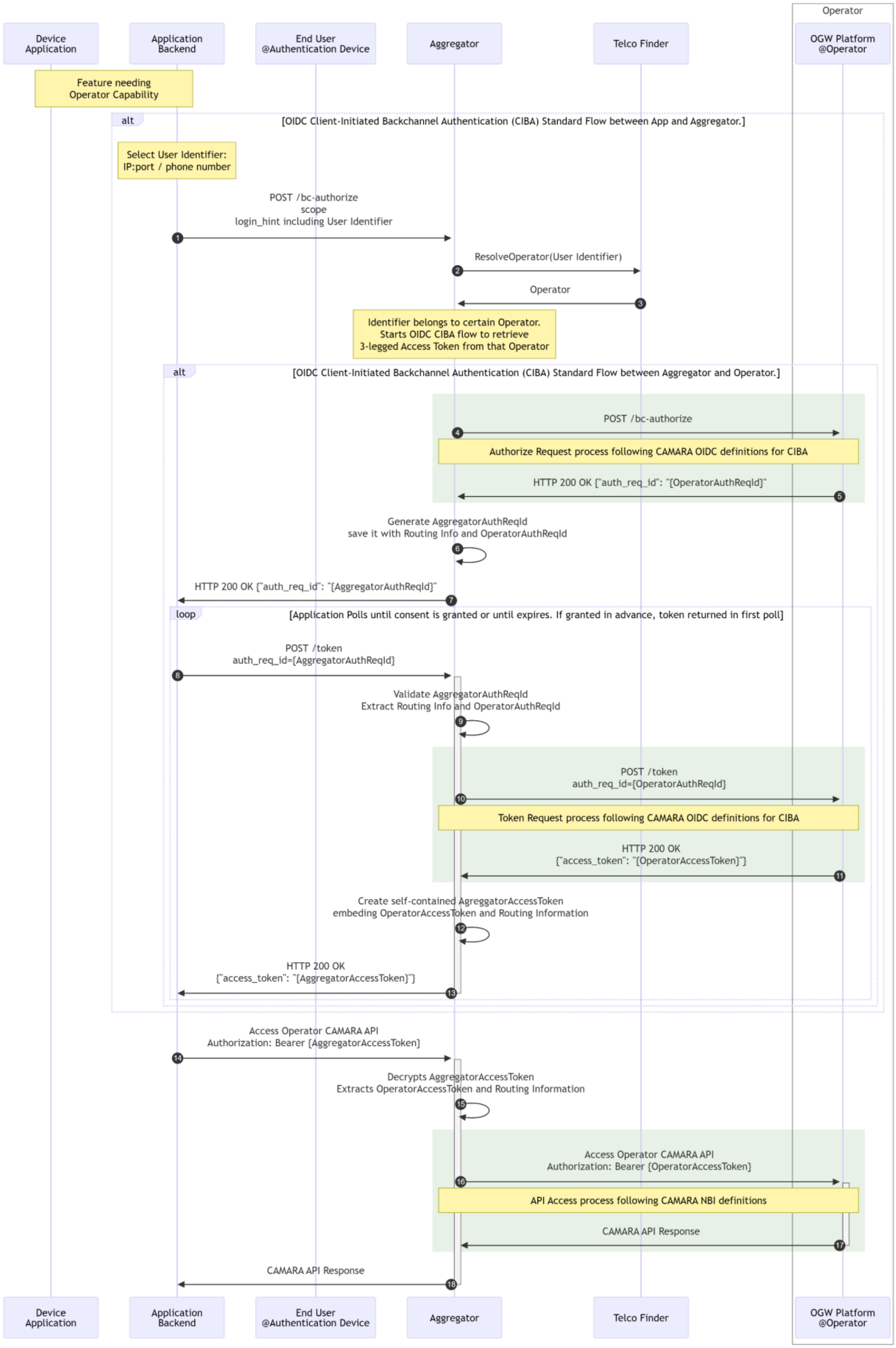


Figure 9: Detailed CIBA flow

Scenario description:

- The Aggregator exposes CAMARA APIs to the Application, access is protected by 3-legged OIDC access tokens (as defined in [8]).
- The same CAMARA APIs are exposed by Open Gateway (OWG) Platform in the Operator to the Aggregator as the Aggregator to the Application i.e.: same northbound interface (NBI).

Note: If the Aggregator is a hyperscaler, the hyperscaler may expose its own services and APIs to Applications, depending on the aggregation model. See section 3.1.1.4 for more details.

- The Aggregator generated access tokens are always based on an Operator access token. As shown above, the technical solution chosen is to encapsulate the Operator access token within the Aggregator access token.

Note: Other implementation options are possible for the same concept. For example, instead of using self-contained tokens, the Aggregator could store the Operator access token (as well as the required routing information) in a database and use a reference token to access it.

- Operator has (at least):
    - Open Gateway (OGW) platform with authentication server providing CAMARA authorisation/authentication mechanisms [7] to issue 3-legged access tokens (as defined in [8]) to the Aggregator (in this case CIBA grant type); and an API gateway to handle API requests with issued access tokens.
    - Privacy Management capability to check if user Consent exists so access tokens can be issued, when applicable.
    - One or several Consent Capture channels.

Flow description:

First, the Application requests an access token from the Aggregator. The process follows the OpenID Connect Client-Initiated Backchannel Authentication (CIBA) flow according to the CAMARA-defined specifics [7] for using the CIBA flow.

The Application has to provide in the authentication request (`/bc_authorize`) a `login_hint` with a valid user identifier together with the Application credentials (as stated in clause 7.1 in [27]) and indicate the purpose for accessing the data (step 1):

- One option for the identifier is the public IP and (optionally – when applicable) port of the Application. The other option is the phone number.

Note: In IoT scenarios or, in general, in those cases where the consumption device is different than the authorisation device, the IP and port is the one of the consumption device for which the network capabilities will be requested/applied.

- The `login_hint` is a hint regarding the user for whom authentication is being requested. It follows CAMARA defined format for `login_hint` as described in the CAMARA Security and Interoperability Profile [7].
- Purpose under which the personal data associated to API consumption will be processed.

Note: The way to declare a purpose when accessing the CAMARA APIs is also defined in the CAMARA Security and Interoperability Profile [7].

Using the information provided by the Application, the Aggregator will first determine the Operator for the given user identifier by querying the Telco Finder (steps 2-3). If the Telco Finder is unable to determine which Operator the user belongs to, the Aggregator will return an error.

Once the Operator is known, the Aggregator should obtain a valid access token to consume the Service API exposed by the OGW Platform in that Operator. The same standard OpenID Connect CIBA flow is also used by the Aggregator to obtain a 3-legged access token (as defined in [8]) from the Operator following CAMARA OIDC definitions for CIBA as described in “CAMARA APIs access and user consent management” [8].

So, the Aggregator sends the authentication request (`/bc_authorize`) to the OGW Platform at the Operator, sending a `login_hint` with the same user identifier provided by the Application, which has already been set as owned by this Operator (according to the Telco Finder response) (Step 4). As part of the CAMARA standard flow, the OGW Platform will:

- Validate the user identifier, map it to an Operator subscription identifier when applicable, e.g.: map IP to phone number.
- Check whether user Consent is required, depending on the legal basis (“legitimate interest”, “contract”, “Consent”, etc.) associated with the declared purpose. If needed, it checks through the Operator Privacy Management if Consent has already been given for exposing the requested data scope for this purpose to the Application.

When the authentication request process is complete on the Operator side (it may require the Operator to trigger an out-of-band Consent capture mechanism to interact with the user), the OGW Platform returns a 200 OK response with the CIBA authentication request identifier (auth_req_id=OperatorAuthReqId) to the Aggregator to indicate that the authentication request has been accepted and will be processed (step 5).

The Aggregator then generates an AggregatorAuthReqId and stores the mapping between the OperatorAuthReqId and the AggregatorAuthReqId along with the Operator routing information (step 6). Finally, the Aggregator returns a 200 OK response to the Application with the CIBA authentication request identifier (auth_req_id=AggregatorAuthReqId) (step 7).

The Application then polls the token endpoint by making an HTTP POST request by sending the grant_type (`urn:openid:params:grant-type:ciba`) and `auth_req_id` (AggregatorAuthReqId) parameters (step 8) according to the CAMARA definitions [7].

- The Aggregator validates the `auth_req_id` and retrieves the OperatorAuthReqId and the Operator routing information.

- The Aggregator routes the polling request to the Operator using OperatorAuthReqId obtained before (step 10). This token request process follows CAMARA OIDC definitions for CIBA as described in “CAMARA APIs access and user consent management” [8].
- Finally, and after the user has given Consent (if required), the OGW Platform will provide the access token (OperatorAccessToken) to the Aggregator (step 11).

Once the Aggregator has the Operator access token, it will create a new access token, AggregatorAccessToken, by creating a JWT extended with additional claims that will carry (step 12):

- The access token issued by the Operator (OperatorAccessToken).
- Routing information to know where to route later API Calls using AggregatorAccessToken.

Note: As mentioned above, there are other ways to implement the same concept. For example, the Aggregator could store the OperatorAccessToken (as well as the necessary routing information) in a database and use a reference token to access it.

The created AggregatorAccessToken will be encrypted so no relevant information is disclosed.

The AggregatorAccessToken will be provided to the Application (step 13), completing the OIDC flow.

At this point, the Application has a valid access token that can be used to invoke the CAMARA API provided by the Aggregator (step 14).

The Aggregator will decrypt the access token, check its validity, find the routing information in it and extract the OperatorAccessToken (step 15). It will then forward the CAMARA API request unchanged to the Operator, including the OperatorAccessToken in the request (step 16). Finally, the Operator will validate the OperatorAccessToken, grant access to the API based on the scopes bound to the access token, forward the request to the corresponding API backend and retrieve the API response. The CAMARA API access process follows CAMARA NBI definitions as described in “CAMARA APIs access and user consent management” [8].

The Operator will provide the API response to the Aggregator (step 17), which will then be sent to the Application (step 18).

3.1.1.2 Detailed Authorization Code flow (frontend-based)

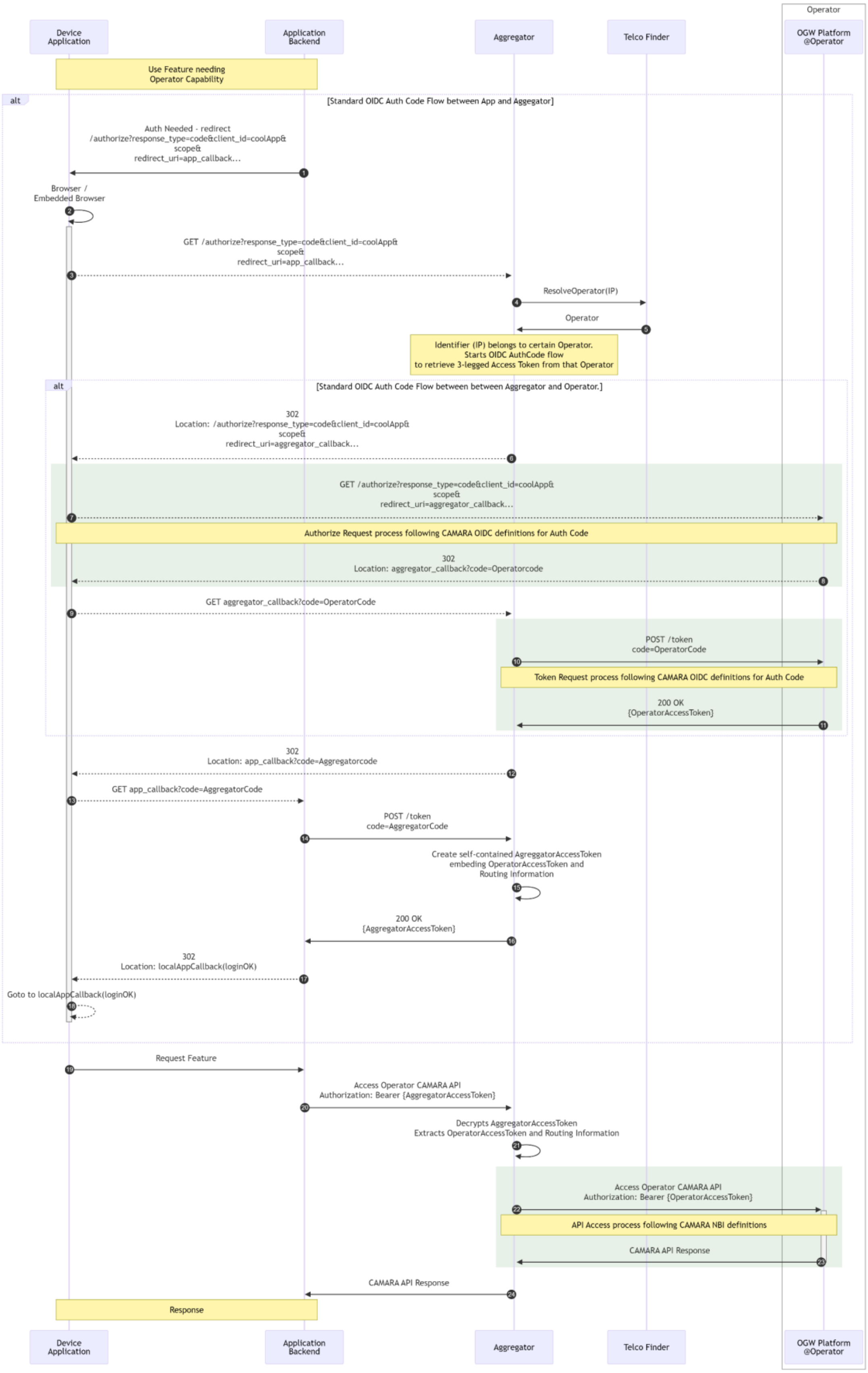


Figure 10: Detailed Authorization Code flow

Flow description:

First, the Application Backend instructs the Application Frontend client in the device to initiate the OIDC Authorization code flow with the Aggregator (steps 1-18) according to the CAMARA-defined specifics [7] regarding the use of the Authorization code flow.

The device Application is redirected to the Aggregator's authorisation endpoint (steps 1-3), providing a redirect_uri (app_callback) pointing to the Application Backend (where the authorisation code will be eventually sent), as well as the purpose for accessing the data.

Note: The way to declare a purpose when accessing the CAMARA APIs is defined by CAMARA as described in the CAMARA Security and Interoperability Profile [7].

The Aggregator receives the request from the device Application and collects the IP from which the device Application is accessing. The Aggregator will then discover the Operator for the connection in the device where the Application is running by querying the Telco Finder (steps 4-5). If the Telco Finder is unable to determine which Operator the user belongs to, the Aggregator will return an error to the Application Backend via the redirect_uri (app_callback).

Once the Operator is known, the Aggregator itself initiates the OIDC Authorization code flow with the Operator (steps 6-11) following CAMARA OIDC definitions for Authorization code flow as described in "CAMARA APIs access and user consent management" [8].

According to the CAMARA Authorization code flow, the device Application is redirected to the authorisation endpoint of the OGW Platform (step 7), providing a redirect_uri (aggregator_callback) pointing to the Aggregator where the auth code will be sent, as well as the purpose for accessing the data originally sent by the device Application to the Aggregator.

Note: The way to declare a purpose when accessing the CAMARA APIs is defined by CAMARA as described in the CAMARA Security and Interoperability Profile [7].

As part of the CAMARA standard flow, the OGW Platform will:

- Use network-based authentication mechanism to obtain an Operator subscription identifier, e.g.: phone number.
- Check whether user Consent is required, depending on the legal basis ("legitimate interest", "contract", "Consent", etc.) associated with the declared purpose. If needed, it checks through the Operator Privacy Management if Consent has already been given for exposing the requested data scope for this purpose to the Application.

When the authorisation request process is complete on the Operator side (it may require Consent capture from user), the Authorization code flow continues by redirecting to the Aggregator's redirect_uri (aggregator_callback) and including the authorisation code (OperatorCode) (step 8).

Once the Aggregator receives the redirect with the authorisation code (OperatorCode - step 9), it will retrieve the access token from the OGW Platform (OperatorAccessToken) (steps

10-11). This token request process follows CAMARA OIDC definitions for the Authorization code flow as described in the CAMARA Security and Interoperability Profile [7].

The Aggregator will then continue the Authorization code flow by redirecting to the Application's redirect_uri (app_callback) and including the authorisation code (AggregatorCode - steps 12-13).

The Application will request an access token from Aggregator (step 14). Aggregator creates a new access token, AggregatorAccessToken, by creating a JWT extended with additional claims that will carry (step 15):

- The access token issued by the Operator (OperatorAccessToken).
- Routing information to know where to route later API calls using the AggregatorAccessToken.

Note: As mentioned above, there are other ways to implement the same concept. For example, the Aggregator could store the OperatorAccessToken (as well as the necessary routing information) in a database and use a reference token to access it.

The AggregatorAccessToken created is encrypted so that no sensitive information is exposed.

The AggregatorAccessToken is made available to the Application (step 16). The Application completes the OIDC flow by interacting backend with frontend (steps 17-18).

Note: The point at which the /token call from the Aggregator to the OGW Platform (OperatorAccessToken) is made is an implementation decision. The flowchart currently shows this happening at steps 10-11. Alternatively, it could also take place directly after the Application Backend makes a /token call to the Aggregator, which would be immediately after step 14 in the current flowchart.

Now the Application has a valid access token that can be used to invoke the CAMARA API provided by the Aggregator (step 20).

The Aggregator will decrypt the access token, check its validity, find the routing information inside and extract the OperatorAccessToken (step 21). Then it will forward the CAMARA API request unchanged to the Operator, including the OperatorAccessToken in the request (step 22). Finally, the Operator will validate OperatorAccessToken, grant the access to the API based on the scopes bound to the access token, progress request to the corresponding API backend and retrieve the API response. The CAMARA API access process follows CAMARA NBI definitions as described in "CAMARA APIs access and user consent management" [8].

Finally, the Operator will provide API response to the Aggregator (step 23) which is then sent to the Application (steps 24-25).

3.1.1.3 Detailed JWT Bearer flow (backend-based)

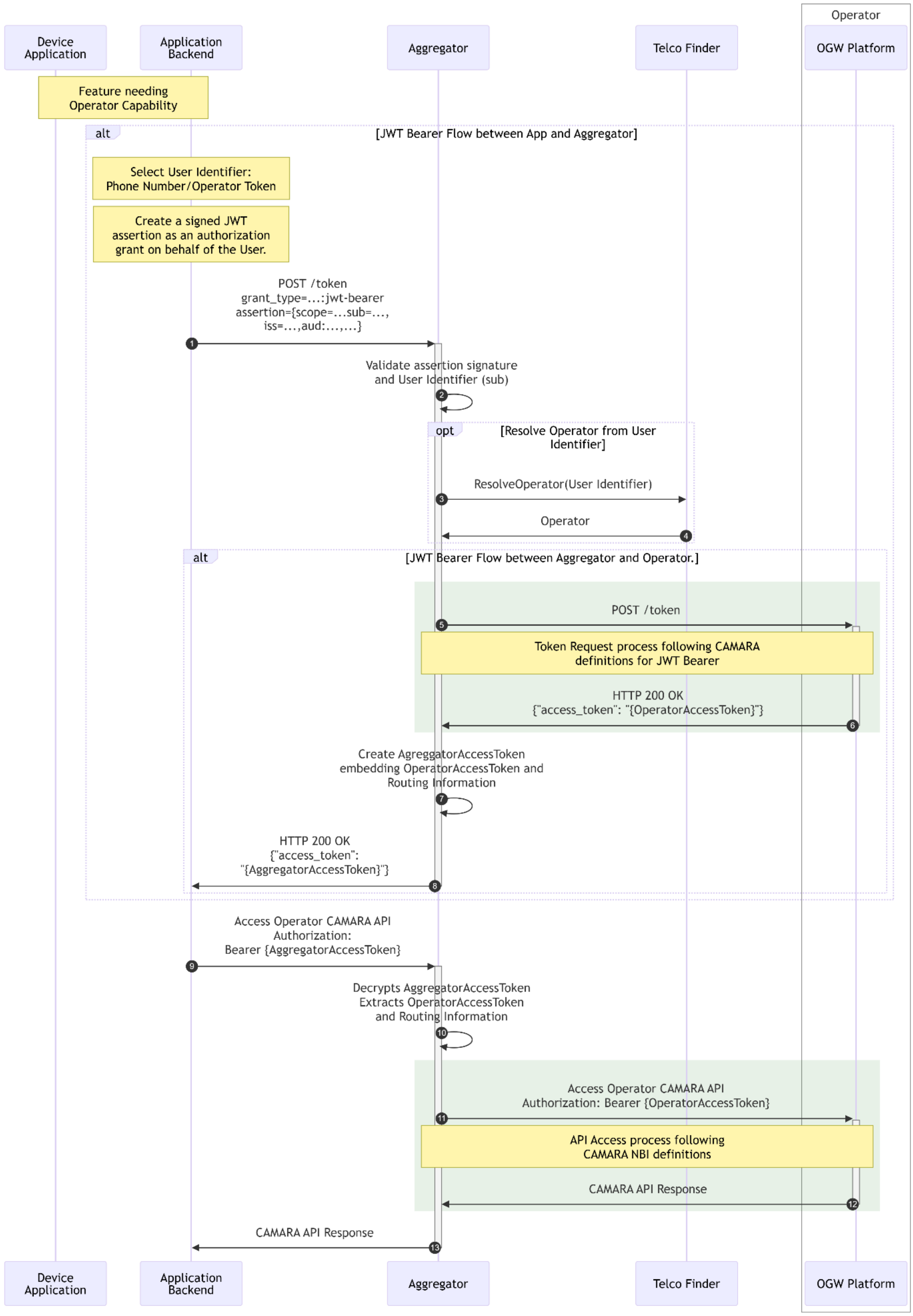


Figure 11 Detailed JWT Bearer flow

Flow description:

Before the flow begins, the Application must share its public key (e.g. JWKS URI) with the Aggregator (e.g., during Application Provider onboarding). This enables the Aggregator to verify the digital signature of JWT assertions issued by the Application in the JWT Bearer flow [29].

First, the Application requests an access token from the Aggregator, which will subsequently be used to access CAMARA APIs. The process follows the JWT Bearer flow [29] according to the CAMARA-defined specifics for using the JWT Bearer flow [8]. This flow enables an Application Backend to obtain a 3-legged access token (as defined in [8]) from the Aggregator by presenting a signed JWT assertion on behalf of the User.

To start, the Application selects the User identifier for which the access token is requested, which can either be a phone number or a TS.43 token. The User is identified by the `sub` claim in the JWT assertion, which must uniquely identify the User in the Operator's system. As per the CAMARA Security and Interoperability Profile [7], the `sub` claim MUST be either a phone number prefixed by "`tel:`" or a TS.43 token prefixed by "`operatortoken:`".

Following this, the Application will create a signed JWT assertion. This assertion functions as an authorization grant on behalf of the User, packed with essential claims: the `grant_type` is set to “`urn:ietf:params:oauth:grant-type:jwt-bearer`”, and the assertion itself includes the requested scopes, the `sub` claim representing the User identifier, the `iss` claim identifying the Application client id, etc. The specific content and claims of the JWT assertion are also specified by the CAMARA Security and Interoperability Profile [7].

Note: The way to declare a purpose when accessing the CAMARA APIs is defined by CAMARA as described in the CAMARA Security and Interoperability Profile [7].

The Application Backend then sends this assertion within a token request (POST /token) to the Aggregator (step 1).

Upon receiving the request, the Aggregator validates the JWT using the Application’s public key and confirms the validity of the User identifier specified in the `sub` claim (step 2). If the User identifier is a phone number, the Aggregator might, when necessary, interact with the Telco Finder to pinpoint the exact Operator associated with that identifier (steps 3-4). If the Telco Finder is unable to determine which Operator the User belongs to, the Aggregator will return an error.

Once the relevant Operator is determined, the Aggregator initiates its own JWT Bearer flow with the Operator's OGW Platform to fetch a 3-legged access token (as defined in [8]) (steps 5-6). This sub-flow strictly adheres to CAMARA definitions for JWT Bearer as described in “CAMARA APIs access and user consent management” [8]. The Aggregator sends a token request to the Operator's OGW Platform, which validates the JWT and the legal basis for the requested access, and if positive, responds with an Operator access token (OperatorAccessToken).

With this Operator access token in hand, the Aggregator proceeds to create a self-contained Aggregator access token (AggregatorAccessToken). This unique token embeds both the

Operator access token and any crucial routing information needed for subsequent API calls (step 7).

Note: As mentioned above, there are other ways to implement the same concept. For example, the Aggregator could store the OperatorAccessToken (as well as the necessary routing information) in a database and use a reference token to access it.

Finally, the Aggregator completes this phase by responding to the Application Backend with an HTTP 200 OK, providing the newly minted AggregatorAccessToken (step 8).

With the AggregatorAccessToken successfully acquired, the Application Backend is now ready to access the desired CAMARA API. It sends its API request to the Aggregator's endpoint, including the AggregatorAccessToken securely in the Authorization header (step 9).

The Aggregator will decrypt the access token, check its validity, find the routing information inside and extract the OperatorAccessToken (step 10). Then it will forward the CAMARA API request unchanged to the Operator, including the OperatorAccessToken in the request (step 11). Finally, the Operator will validate OperatorAccessToken, grant the access to the API based on the scopes bound to the access token, progress the request to the corresponding API backend and retrieve the API response. The CAMARA API access process follows CAMARA NBI definitions as described in “CAMARA APIs access and user consent management” [8].

Finally, the Operator will provide an API response to the Aggregator (step 12) which is then sent to the Application (step 13).

3.1.1.4 Enhanced Service API flows

The Aggregator may expose its own services and APIs to Applications depending on the aggregation model (instead of exposing CAMARA APIs available at the aggregated Operators). In this case, the Aggregator uses the Operator’s CAMARA APIs to implement either explicit or implicit functions, providing developers with enhanced services using the Operator’s network capabilities and /or network resources.

When exposing their own services, the northbound authentication mechanism of Aggregators is up to them, as long as their flows contain the necessary information for each call to the Operator. The Aggregator is responsible for obtaining the Operator access token and including it in the CAMARA API calls to the Operator, following the CAMARA standard mechanisms as described in the “CAMARA Security and Interoperability Profile” [7] and “CAMARA APIs access and user consent management” [8] technical specifications.

The flows provided in the previous sections represent the scenario where the Aggregator exposes the same NBI as the Operator, i.e., the Aggregator exposes the same CAMARA APIs as the aggregated Operators. The flows below provide a generic view of the flows when the Aggregator exposes its own enhanced Service APIs to the Application.

Note: The parts of the flow in blue corresponds to the interaction between the Aggregator and the Application. When the Aggregator exposes its own services, these parts would actually be outside the scope of Open Gateway

Technical Group since each Aggregator would have its own mechanisms and services.

3.1.1.4.1 Enhanced Service API - CIBA flow (backend-based)

The Aggregator will be responsible for triggering the OIDC CIBA standard flow to the Operator when required according to its service design. Finally, it will be responsible for using Operator's CAMARA API(s) response as required by its enhanced service. The CIBA flow towards the Operator follows the CAMARA OIDC definitions for CIBA as described in the CAMARA Security and Interoperability Profile [7].

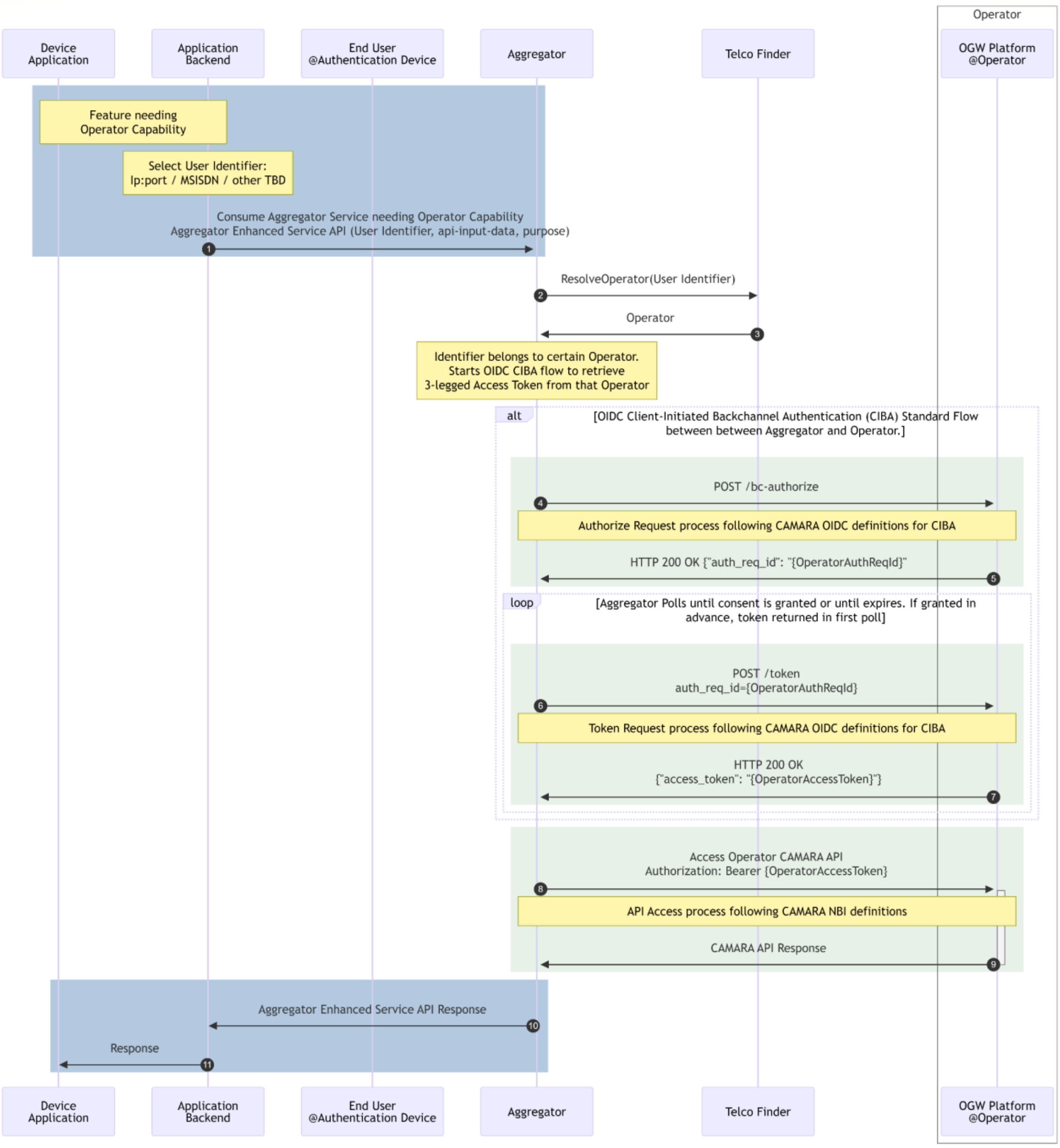


Figure 12: Enhanced Service API - CIBA flow

3.1.1.4.2 Enhanced Service API - Authorization Code flow

The Aggregator will be responsible for redirecting the user device and triggering the Authorization code flow to the Operator when required according to its service design. Finally, it will be responsible for using Operator’s CAMARA API response as required by its enhanced service. The Authorization code flow towards the Operator follows the CAMARA OIDC definitions for Authorisation code as described in the CAMARA Security and Interoperability Profile [7].

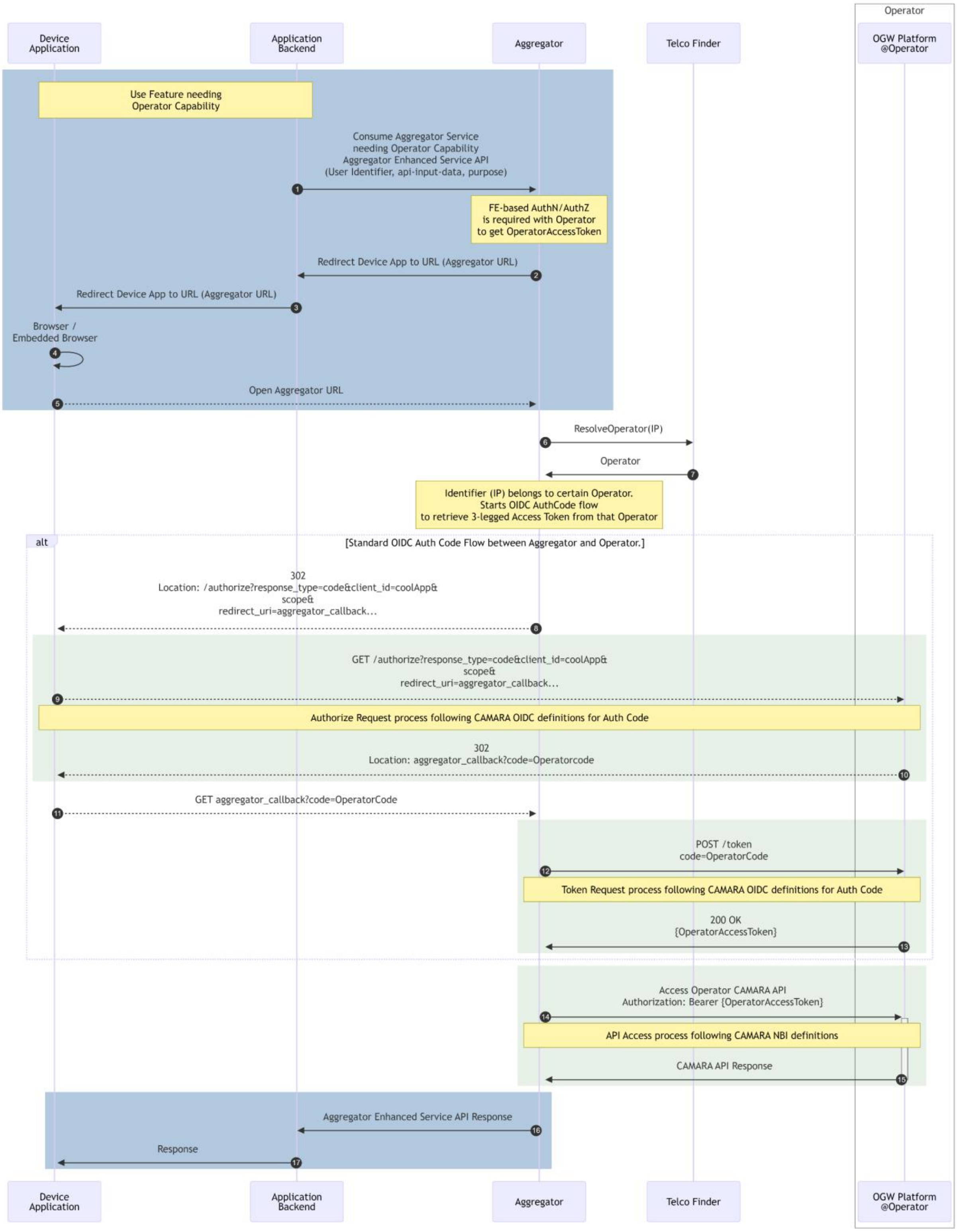


Figure 13: Enhanced Service API - Authorization Code flow

3.1.1.4.3 Enhanced Service API JWT Bearer flow

The Aggregator will be responsible for triggering the JWT Bearer standard flow to the Operator when required according to its service design. Finally, they will be responsible for using Operator’s CAMARA API(s) response as required by its enhanced service. The JWT

Bearer flow towards the Operator follows the CAMARA definitions for JWT Bearer as described in the CAMARA Security and Interoperability Profile [7].

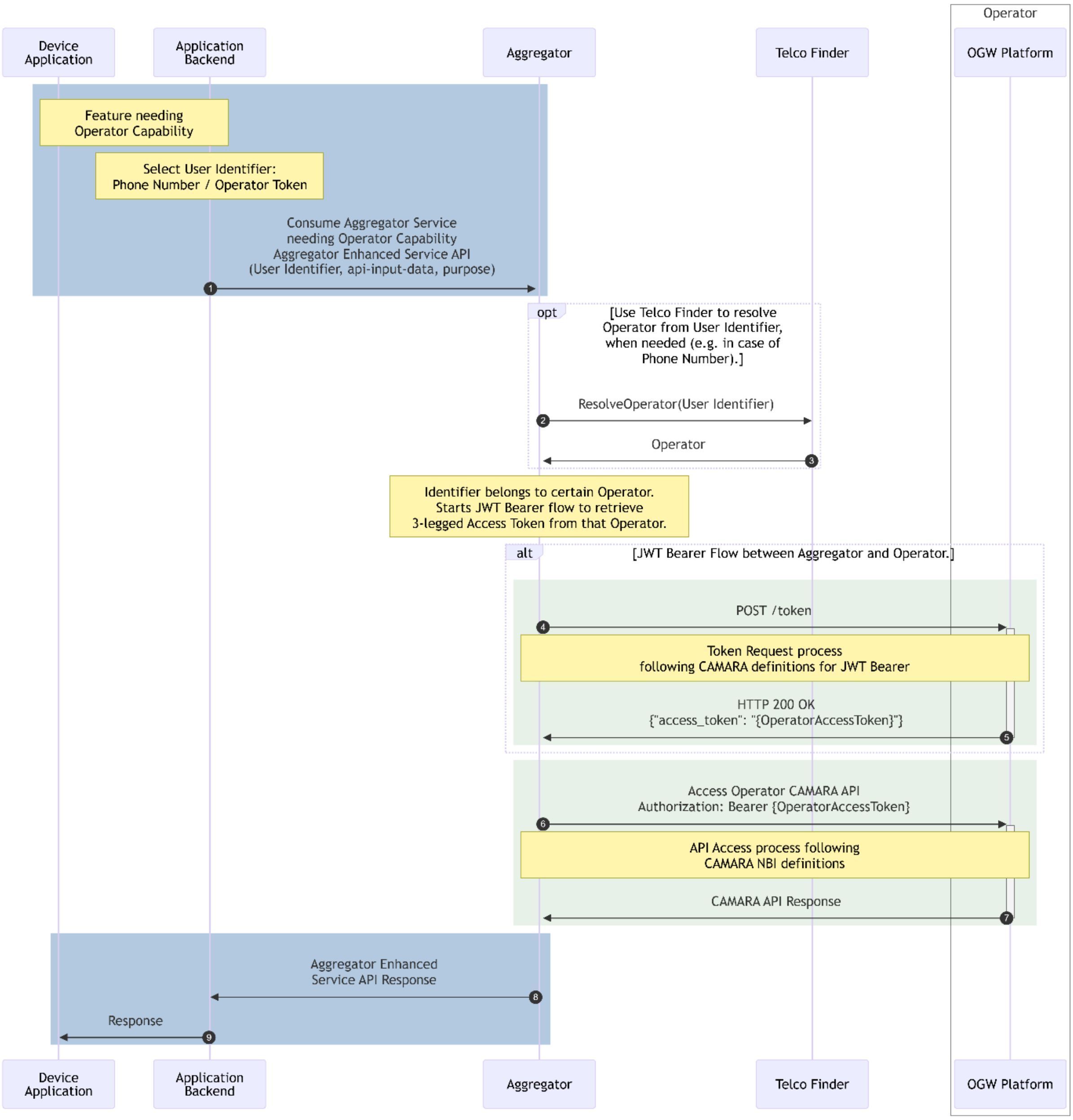


Figure 14 Enhanced Service API - JWT Bearer flow

3.1.1.5 Offline Access – Refresh Token

The defined solution uses the OAuth 2.0 refresh token to enable Service APIs access in offline scenarios. In Open Gateway, online access means that there is an existing connection to the Operator's network, either a mobile network connection in mobile use cases or a fixed network connection in fixed use cases.

- The refresh token can be issued as a result of the authentication/authorisation process if requested by the API client and allowed by the use case.

  - OAuth 2.0 Authorization Framework [10]: a refresh token is a string representing the authorisation granted to the client by the resource owner. The string is usually opaque to the client. The token denotes an identifier used to retrieve the authorisation information. Therefore, the refresh token is bound to the authorisation of the End-User to the application.

- As indicated in the OIDC Specification [11], one of the uses of the refresh token is to allow offline access.
- Thus, a refresh token can be used for offline access because it represents an existing authorisation, that was collected in a previously online access.
- It can also be used for online access to avoid a re-authentication. This would be use case driven and depends on the security requirements for each API, e.g., APIs may require more frequent authentication or even authentication for each use (like Number Verification API).

This chapter includes the flows for Open Gateway showing the usage of the refresh token.

- How to request it in both frontend and backend flows
- How to handle refresh tokens in an architecture involving an Aggregator.

The flows described below follow the “CAMARA Security and Interoperability Profile” [7] technical specification.

3.1.1.5.1 Request a Refresh Token in frontend-based flow

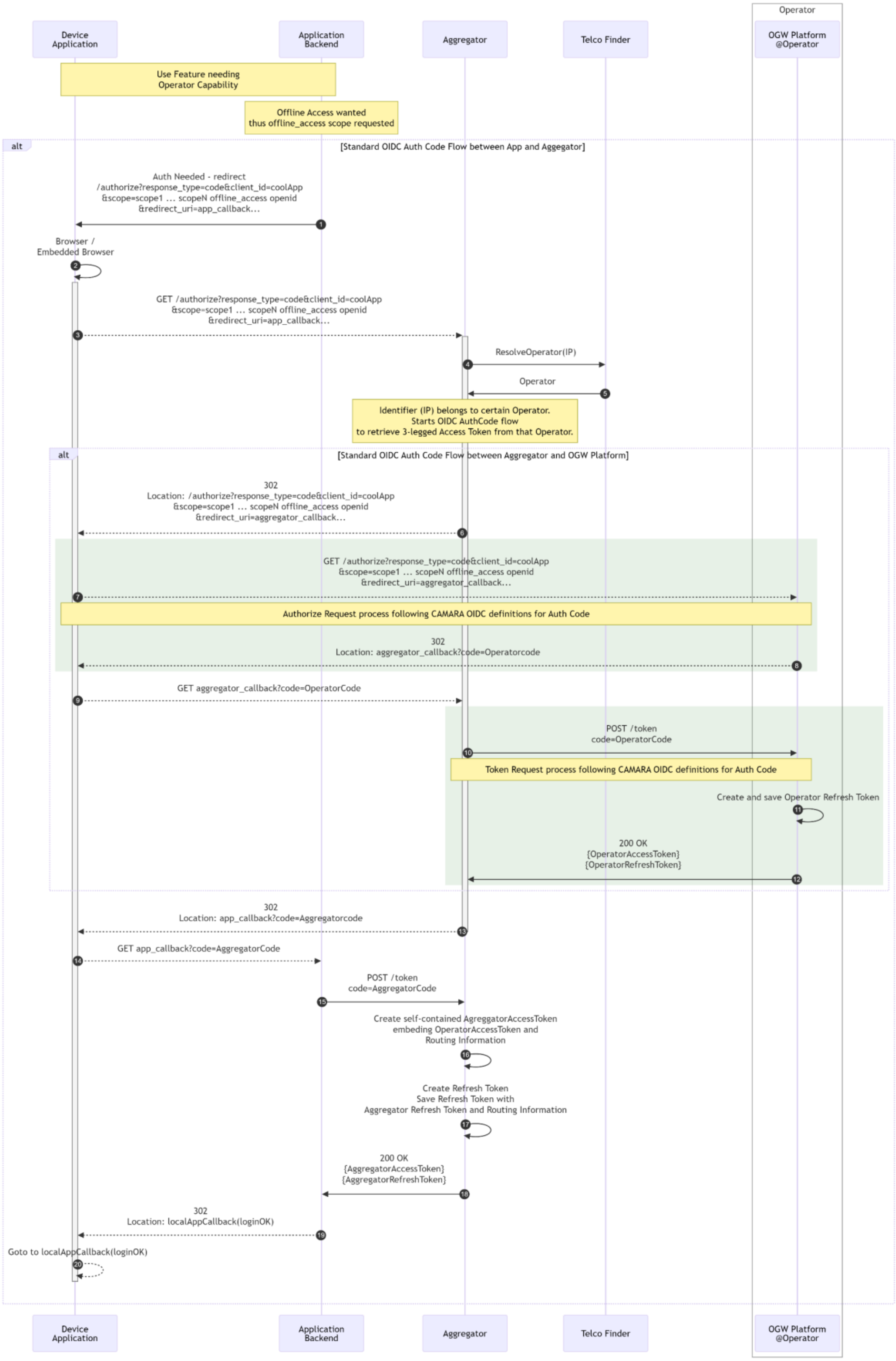


Figure 15: Request a Refresh Token in frontend-based flow

Scenario description:

- The same considerations that apply to general call flows in section 3.1.1.2 apply to this flow.
- Refresh token issuance is achieved by requesting offline_access scope.

Flow description

The Application Backend instructs the Application Frontend client in the device to initiate the OIDC Authorization code flow with the Aggregator as described in the general call flows in section 3.1.1.2.

The Application includes offline_access scope to signal that a refresh token is expected (step 1). The same flow depicted in the general call flows occur (steps 2-10) following the “CAMARA Security and Interoperability Profile” [7] technical specification.

Note: The process has been simplified, full details are provided in the general call flow description in section 3.1.1.2.

When the Operator issues the access token, it also issues a refresh token (OperatorRefreshToken) (step 11), because offline_access scope was requested.

Note: Depending on the use case this will be allowed or not. i.e., not every Application will have the right to have a refresh token by default.

The Aggregator receives both the access token and the refresh token (step 12) and continues with the regular Authorization code flow between the Application and the Aggregator (steps 13-16).

In addition, the Aggregator creates its refresh token (AggregatorRefreshToken) and stores it along with the OperatorRefreshToken and routing information (step 17).

Note: Other implementation options are possible for the same concept. For example, self-contained refresh tokens may be used instead.

The flow is completed normally, so the Aggregator provides both the access token and the refresh token to the Application (steps 18-20).

3.1.1.5.2 Request a Refresh Token in backend-based flow

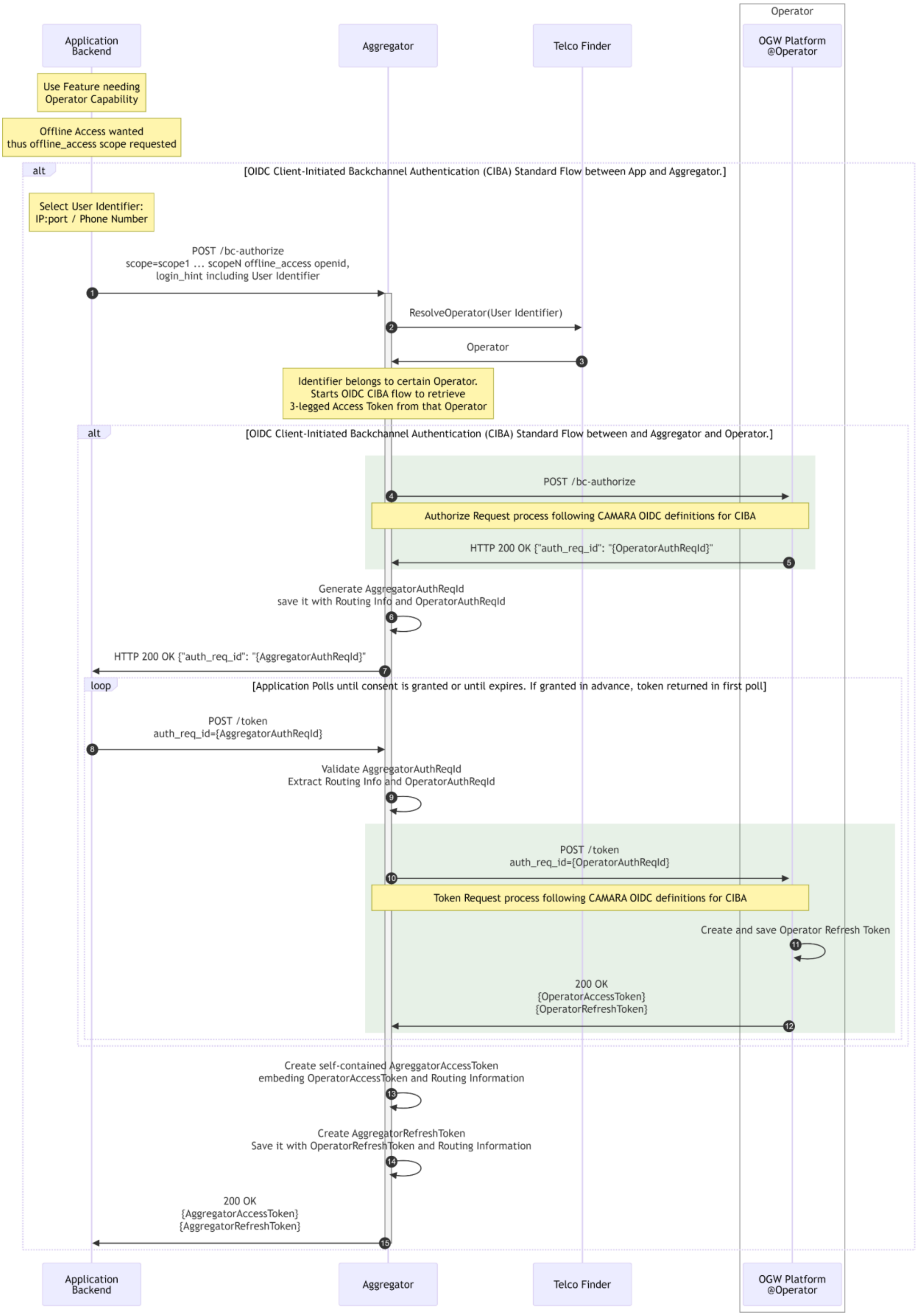


Figure 16: Request a Refresh Token in backend-based flow

Scenario description

- The same considerations that apply to general call flows in section 3.1.1.1 apply to this flow.
- Refresh token issuance is achieved by requesting offline_access scope.

Flow description

The Application Backend requests an OIDC access token to the Aggregator. The process follows the OpenID Connect Client-Initiated Backchannel Authentication (CIBA) flow as described in the general call flows in section 3.1.1.1.

The Application determines the user identifier to use and performs the authorisation request (/bc-authorize) including the offline_access scope to signal that a refresh token is expected (step 1). The same flow as shown in the general call flows occur (steps 2-10) following the “CAMARA Security and Interoperability Profile” [7] technical specification.

Note: The process has been simplified, full details are provided in the general call flow description in section 3.1.1.1.

When the Operator issues the access token, it also issues a refresh token (OperatorRefreshToken) (step 11), because offline_access scope was requested.

Note: Depending on the use case this will be allowed or not. i.e., not every Application will have the right to have a refresh token by default.

The Aggregator receives both the access token and the refresh token (step 12) and continues with the regular CIBA flow, creating the AggregatorAccessToken (step 13).

In addition, the Aggregator creates its refresh token (AggregatorRefreshToken) and stores it along with the OperatorRefreshToken and routing information (step 14).

Note: Other implementation options are possible for the same concept. For example, self-contained refresh tokens may be used instead.

The flow is completed normally, so the Aggregator provides both the access token and the refresh token to the Application (step 15).

3.1.1.5.3 Refresh token flow

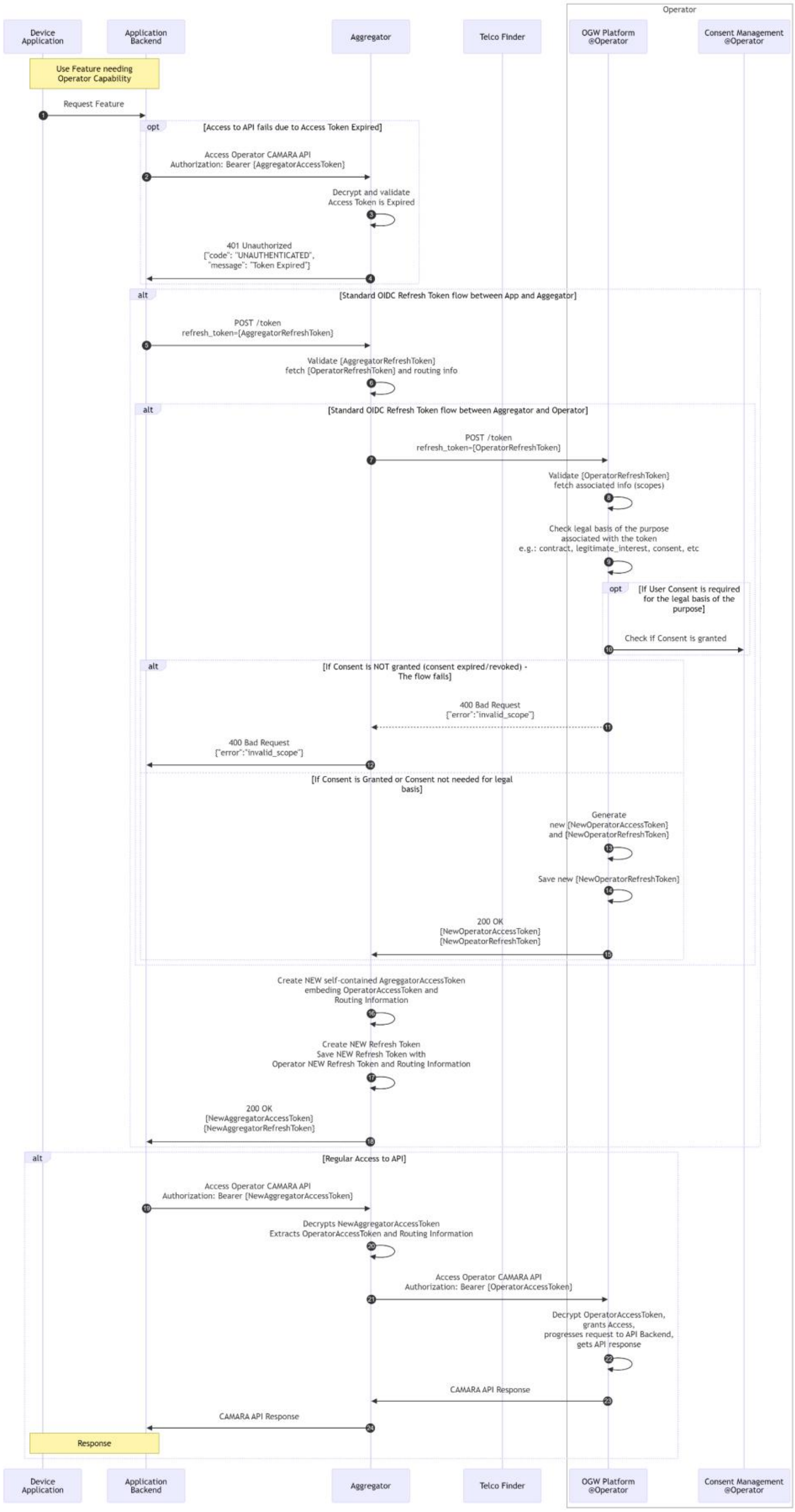


Figure 17: Refresh token flow

Scenario description

- The same considerations that apply to general call flows in the previous sections apply to this flow.

Flow description

When a network feature is needed (step 1), the Application tries to consume an API with an access token, but the token has already expired, so it gets an error (steps 2-4).

Upon this error, the Application performs the standard refresh token grant flow, providing the Aggregator refresh token to obtain a new pair of Aggregator refresh token and Aggregator access token (step 5).

The Aggregator validates the refresh token and retrieves the Operator refresh token and the routing information (step 6).

In turn, the Aggregator performs the standard refresh token flow with the Operator, providing the Operator refresh token to obtain a new pair of Operator refresh token and Operator access token (step 7).

The Operator validates the refresh token and retrieves the related information, i.e., scopes/purpose (step 8) and checks the legal basis of the purpose associated with the token (step 9). If the purpose requires the user's Consent, the Operator validates whether the Consent is granted (step 10). If the Consent is NOT granted e.g., the Consent has expired or the user has revoked the Consent, the Operator will not issue a new token and will return an error (step 11) and the refresh token flow will fail (step 12).

If the user's Consent is granted or not required for the applied legal basis, the Operator generates a new pair of Operator access token and Operator refresh token (step 13), saves the refresh token (step 14), and responds to the Aggregator with this information (step 15).

Same as in general call flows, the Aggregator will generate a new Aggregator access token (step 16) and this time will also generate a new Aggregator refresh token (step 17) that will be saved bound to Operator refresh token and routing information.

Note: As mentioned on other occasions, using self-contained tokens or any other options such as storing them, is an implementation decision that applies to both access tokens and refresh tokens.

The Aggregator will provide the Application with the new generated Aggregator access token and Aggregator refresh token (step 18).

The Application can then access the API normally using the new Aggregator access token (steps 19-24). This part of the flow is the same as the general call flows.

3.1.2 How to consume Operate APIs

Following GSMA requirements, Operate APIs are defined in the TM Forum domain to fulfil the required Service API management operations, administration and maintenance needs for Aggregators. Using these Operate APIs the Aggregator can discover, order, get reports and assure Service API access.

3.1.2.1 Overview of Operate APIs

The Operate APIs currently assessed for this domain are summarized below, with expected plans for later addition or detailed breakdown of the Operate API functions.

Function	Description	TMF #	Status
Application & Developer Management	Enables automation when registering developers and onboarding their Applications through an Aggregator/Aggregator portal. Operators shall provide the capability for Aggregators to request the registration of Application Providers and the onboarding of their Applications. This includes the end-to-end lifecycle management including validation of the legal entity, modification of existing Applications or management of Terms and Conditions acceptance.	TMF 931 [19]	Published
API Catalogue Management	Provides a standard catalogue among Operators and Aggregator to align the lifecycle management of the Service APIs. Operators shall provide the capability for Aggregators to manage (CRUD) Service APIs building up the North Bound Open Gateway Platform catalogue. Catalogue synchronization is needed, so that Channel Partners can keep an up-to-date list of Service APIs made available for consumption to the developers	TMF 936 [30]	Published
Usage Reporting	Definition of a unified way to report usage information of the Service APIs, so that Aggregators can make it available to the developers accordingly. Information related to consumption of Service APIs is managed by Operators, who are the ones having an end-to-end view of the services and network resources. This information is exposed towards Aggregators for consolidation and billing management.	TMF 937	Planned
Service Assurance	Enabling a common framework for supervision activities over Service APIs. Operators need to provide developers and Aggregators the capability to supervise the status of running Service APIs, using appropriate performance management (e.g., monitoring) and fault management (i.e., issue/ticket management) mechanisms.	Planned	Planned

Table 2 Operate APIs description

The E2E journey for Service API exposure, once an agreement is established between a Operator and an Aggregator, consists of six distinct phases that represent the complete lifecycle of Service API management and consumption. Multiple detailed scenarios might require other variation of capabilities as described in the section 5.2, but this section covers the initial onboarding and consumption to elaborate on the segregation of roles between TMForum and CAMARA.

TMForum provides further elaboration on the behaviour of the Operate APIs, expected sequence of Operate API operation use, and reference design for the internal fulfilment on

Operators systems in the TMFS026 use case: Commercializing CAMARA APIs with Operate API [33]. This reference internal design can be used as a non-binding guideline of which internal Operator entities would be affected by an Operate API operation and how.

Figure 18 showcases the flow covering operational setup of Aggregator and Operator Catalogues, onboarding, ordering, consumption, and usage reporting capabilities.

Note: Service assurance is expected as a future domain within the journey, more information can be included once requirements and solution are ready.

The elaboration shows the segregation between the TM Forum supported operational phases and CAMARA supported runtime phase.

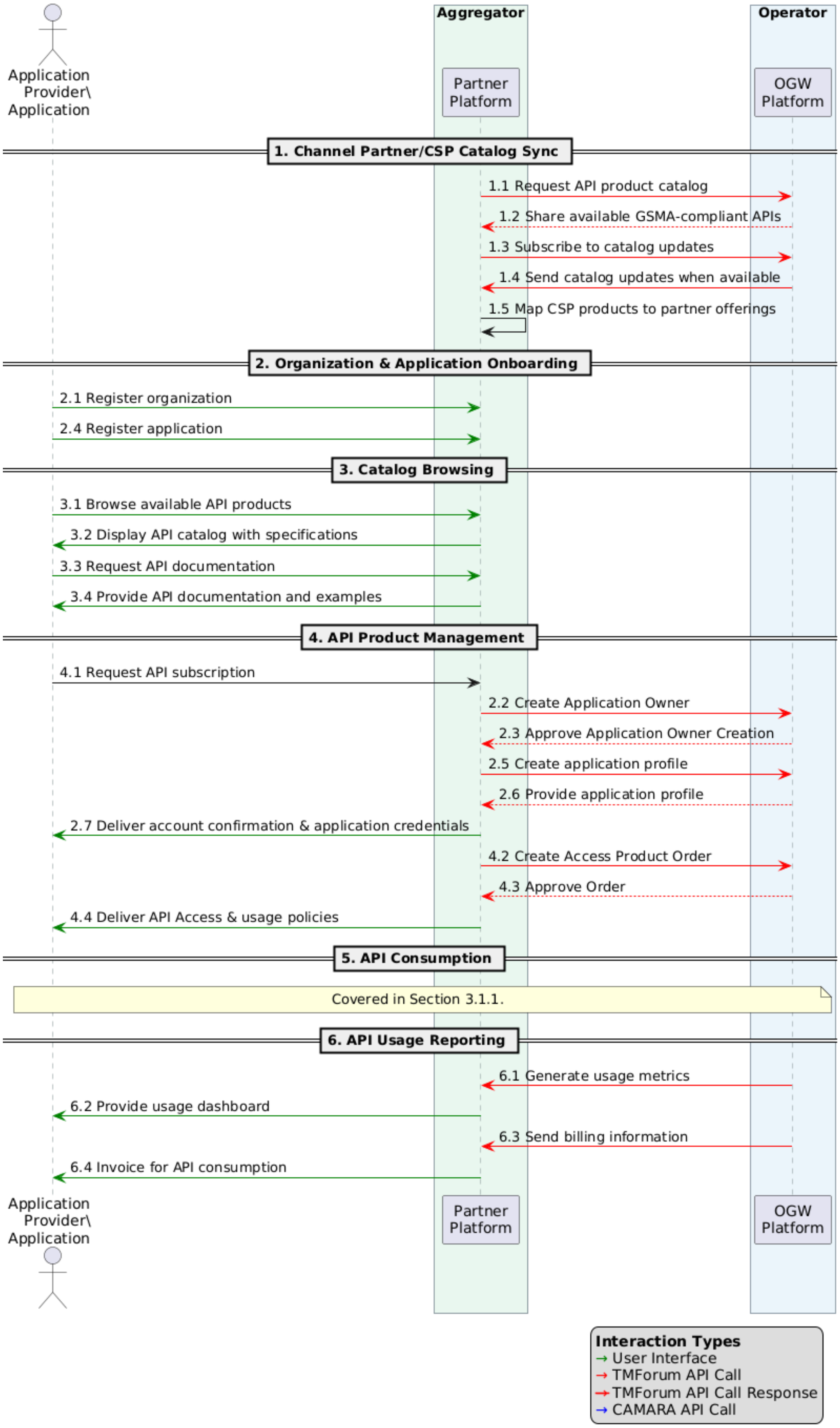


Figure 18: Catalogue setup, onboarding, ordering, consumption, and usage reporting

3.1.2.2 Aggregator/Operator Catalogue Sync

This phase covers the synchronization of Service API product information between the Operator and Aggregator platforms, ensuring that the Channel Partner has an up-to-date catalogue of available network capabilities.

The capabilities required for information sharing for this phase are covered by TMF936 Open Gateway Product Catalog API.

3.1.2.3 Organization & Application Onboarding

This phase covers the registration of developer organisations and their Applications in both the Aggregator and Operator systems, establishing the identities and relationships necessary for Service API access. Initial Application Provider and Application information is captured by the Aggregator, but not propagated to any Operator until an initial Service API is requested by an Application Provider for an Application from that specific Operators.

Information capture is managed by the Aggregator and no Operate APIs are required to pass the information to the Operator at this phase.

3.1.2.4 Catalogue Browsing

This phase depends on the catalogue being synchronised before the browsing and allows developers to explore the available Service API offerings through the Aggregator's interface, reviewing specifications and documentation to select suitable Service APIs for their Applications.

Catalogue browsing by the developer is managed by the Aggregator and no Operate APIs are required to pass the information at this phase.

3.1.2.5 Service API Product Management

This phase covers the subscription process where developers request access to specific APIs and receive the necessary credentials to make Service API calls. Information is captured and maintained by the Operator related to Application and Application Provider, as well as the selected Service API, it’s pricing model, purpose of use, etc.

The capabilities required for information sharing for this phase are covered by TMF931 Open Gateway onboarding and ordering component suite API.

3.1.2.6 Service API consumption

This phase covers the actual use of the Service APIs, including the authentication flow and subsequent Service API calls as described in section 3.1.1. No TM Forum APIs are utilized for that phase.

3.1.2.7 Service API Usage Reporting

This phase covers the tracking, reporting, and billing of Service API usage, providing visibility to all parties and enabling proper monetization of the API ecosystem.

The capabilities required for information sharing for this phase are covered by TMF937, which is still a work in progress.

3.2 Variants and simplified models

The aggregation model described in section 3.1 has some variants in scenarios where not every actor/role is present and/or an actor plays more than one role.

The following variants exist:

- Direct Integration Developer – Operator (Single Operator)
- Direct Integration Developer – Operator (Multiple Operators)
- Operator - Operator Integration

Note: Roaming scenarios are not considered variants or alternative scenarios for the Aggregation model.

The Aggregation flows described in section 3.1 ensure that API calls to a Service API will always be routed to the user's home Operator, ensuring consistent API access regardless of the user's roaming status. Once an access token is obtained through the authentication and authorisation procedures, it contains the necessary routing information to direct the Service API request to the user's home network Service Northbound Interface (NBI). The Telco Finder step is performed only once during this initial process and is not required for each subsequent API call.

However, the specific network functionality provided by the Service API and its handling in the Southbound Interface (SBI) between the API server and the Operator network (Transformation function) could be affected by the user's roaming status. Federation agreements among Operators may be required to provide the API functionality under roaming conditions, or the API functionality might not be supported if the user is roaming.

3.2.1 Direct Integration Developer – Operator (Single Operator)

In this model, the Aggregator's role is omitted, simplifying the integration process. The Developer/Application Provider directly communicates with a single Operator, bypassing the need for an intermediary Aggregator.

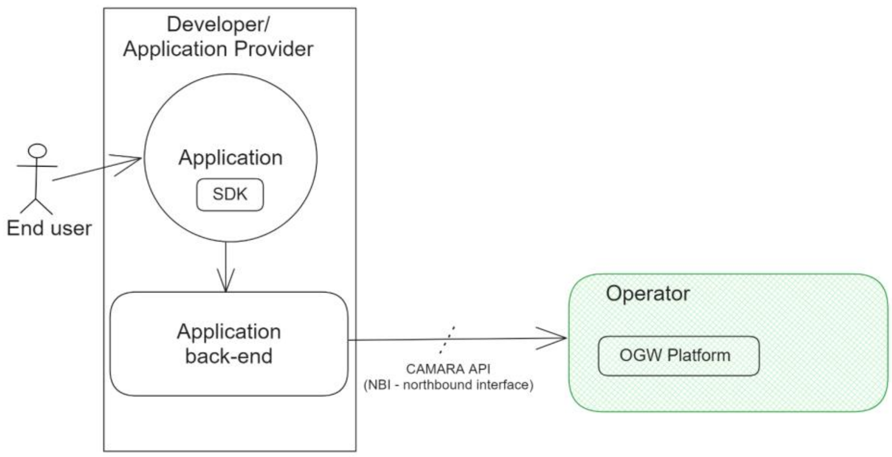


Figure 19: Direct Integration Developer – Operator (Single Operator)

This simplified variant allows the developer's Application to directly interface with a single OGW Platform. The Operator exposes its network capabilities and services through Service

APIs. By eliminating the Aggregator, the integration process becomes more straightforward and reduces the overhead associated with managing multiple Operator endpoints or services.

Key Points:

- Simplicity: without the Aggregator, the developer's Application connects directly to the single Operator, simplifying the integration architecture.
- Direct communication: the developer interacts directly with the Operator's Service APIs using the CAMARA standard NBI, ensuring a clear and consistent integration process.
- Known endpoint: since there is only one Operator, the endpoint for Service API calls is predefined and consistent, eliminating the need for dynamic endpoint resolution or Telco Finder services.

Simplification:

- Reduced complexity: the absence of an Aggregator reduces the layers of interaction.
- Cost efficiency: without the need for an Aggregator, potential costs associated with intermediary services are eliminated.
- Improved performance: direct communication with the Operator can result in lower latency and faster response times since there are fewer intermediary steps.

Limitations:

- Limited scalability: this model is less scalable when the Application needs to interface with multiple Operators in the future. Each new integration would require additional development effort.
- Feature limitation: the Application's capabilities are limited to the services provided by the single Operator, potentially restricting functionality compared to an environment with aggregated services from multiple Operators.
- Lack of interoperability.
- How the Operator knows which users are valid (Operator's Subscribers) is outside the scope and should be resolved by the Application Provider.

In this model, the Telco Finder service is not required. Since the developer's Application always communicates with the single known Operator, there is no need to dynamically discover or resolve endpoints. This further simplifies the integration process, as the endpoint for API calls is predetermined and static.

3.2.2 Direct Integration Developer – Operator (Multiple Operators)

In this model, the Aggregator is not explicitly present. Instead, the Application Provider communicates directly with multiple Operators, effectively taking on part of the role of the Aggregator itself. This approach simplifies the Aggregator model in section 3.1 by allowing the Application to manage integrations with multiple Operators.

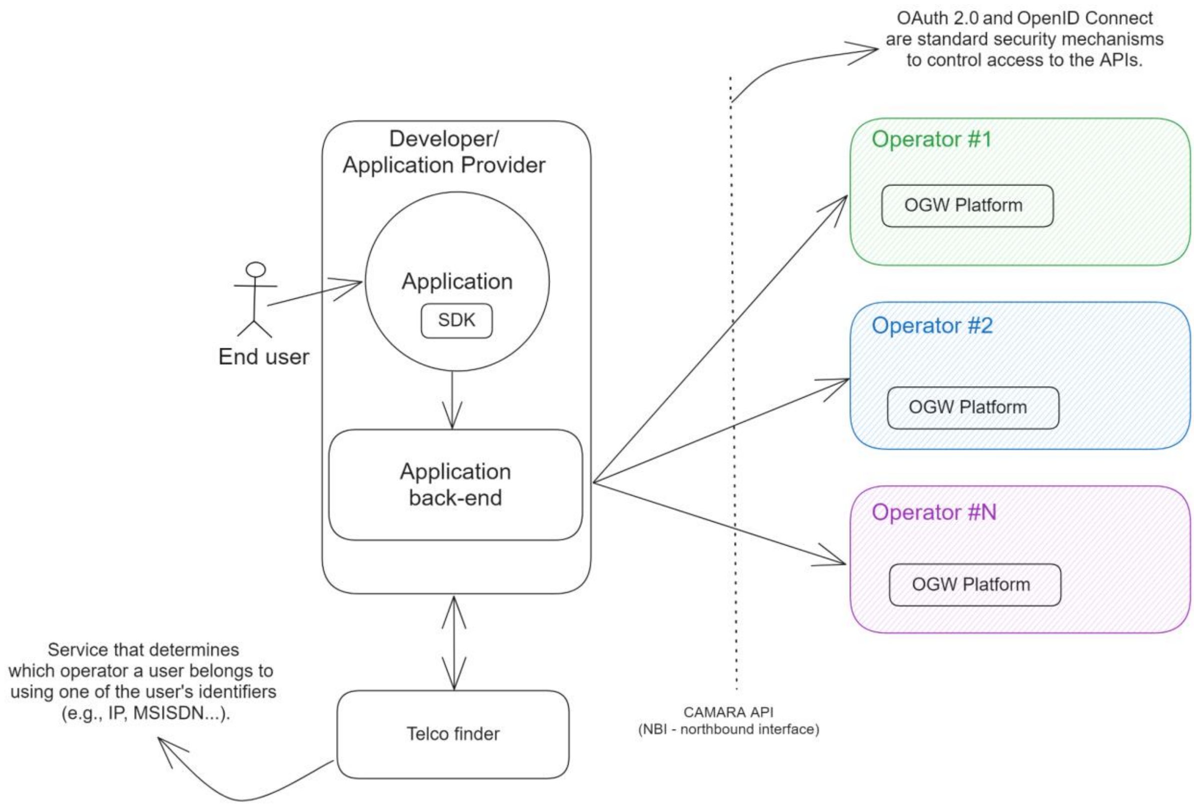


Figure 20: Direct Integration Developer – Operator (Multiple Operators)

This model allows the developer's Application to interface directly with multiple OGW Platforms. Each Operator exposes its network capabilities and services through Service APIs. The Application handles the aggregation, orchestration, and management of these multiple Operator endpoints.

Key Points:

- Direct communication: the developer’s Application interfaces directly with each Operator's Service APIs, maintaining multiple direct connections.
- Application takes on part of the Aggregator role: the Application itself aggregates the services from different Operators, taking on responsibilities typically handled by an Aggregator.
- Flexibility: the developer can choose which Operators to integrate with based on specific needs and use cases.

Simplification:

- Increased control: the developer has full control over how integrations are managed, allowing for custom aggregation logic and optimisation.
- Tailored functionality: direct access to multiple Operators enables the developer to leverage specific features and services unique to each Operator, creating a more customised Application experience.
- Cost efficiency: by eliminating the Aggregator, the developer can potentially reduce costs associated with intermediary services.

Limitations:

- Complexity: managing multiple direct integrations increases the complexity of the Application, requiring more sophisticated handling of different Service APIs and services.
- Scalability challenges: as the number of integrated Operators grows, the overhead of managing these connections can become significant, potentially impacting performance and maintenance.
- developer burden: the developer assumes additional responsibilities typically managed by an Aggregator, such as endpoint orchestration, service discovery, and error handling across multiple Operators.

In this model, the developer's Application can utilise a Telco Finder service similarly to how an Aggregator would. The Telco Finder assists in identifying and selecting the appropriate Operator endpoints for different services, facilitating dynamic endpoint resolution and reducing the complexity associated with managing multiple direct integrations.

3.2.3 Operator – Operator Integration

As indicated in section 3.1, the Aggregator role can be played by an Operator acting as an Aggregator, i.e., aggregating other Operators and exposing CAMARA APIs available at these Operators.

Therefore, the Operator-to-Operator integration model simply represents the same Aggregation model described, the same flows and descriptions apply, but just with an Operator playing the role of Aggregator.

In some cases, an Operator A, acting as the Aggregator, may aggregate its services along with those of other Operators (Operator B, C, etc.). This scenario may simplify the communication flows when Operator A interacts with itself as both an Aggregator and an Operator:

- Internal optimisation: operator A can implement internal optimisations to streamline interactions between its Aggregator and Operator roles.
- Simplified routing: when communicating between Operator A's Aggregator and Operator services, the routing and flow of Service API calls can be simplified.

4 Consent capture: use cases and flows

Several scenarios for Consent capture are possible depending on (among others):

- Whether the Consent is captured while trying to consume network capabilities or the Capture is done upfront,
- The number of devices involved in the use case as well as their respective capabilities and their connectivity status,
- Whether the End-User is the person entitled to consent access to protected data or not,
- Whether the Consent capture takes place within the Application execution context or not.

Some of those scenarios are covered in the following sections of this section.

4.1 In-flow Consent capture

This type of capture leverages the mechanisms provided by OAuth 2.0 authorization code flow [16].

A first use case covered by this type of capture considers an End-User (acting also as a person entitled to consent access to protected data) using an Application on a device with frontend capabilities (e.g., capable of handling HTTP redirects) as shown in Figure 21.

In Figure 21, it is shown how the Application Backend may communicate with an Authentication/Authorization Function through the OGW Platform:

- the dashed lines indicate indirect requests (typically HTTP redirects) being handled in the UE (typically known as front channel interactions)
- the solid lines indicate direct interactions (typically known as backchannel interactions).

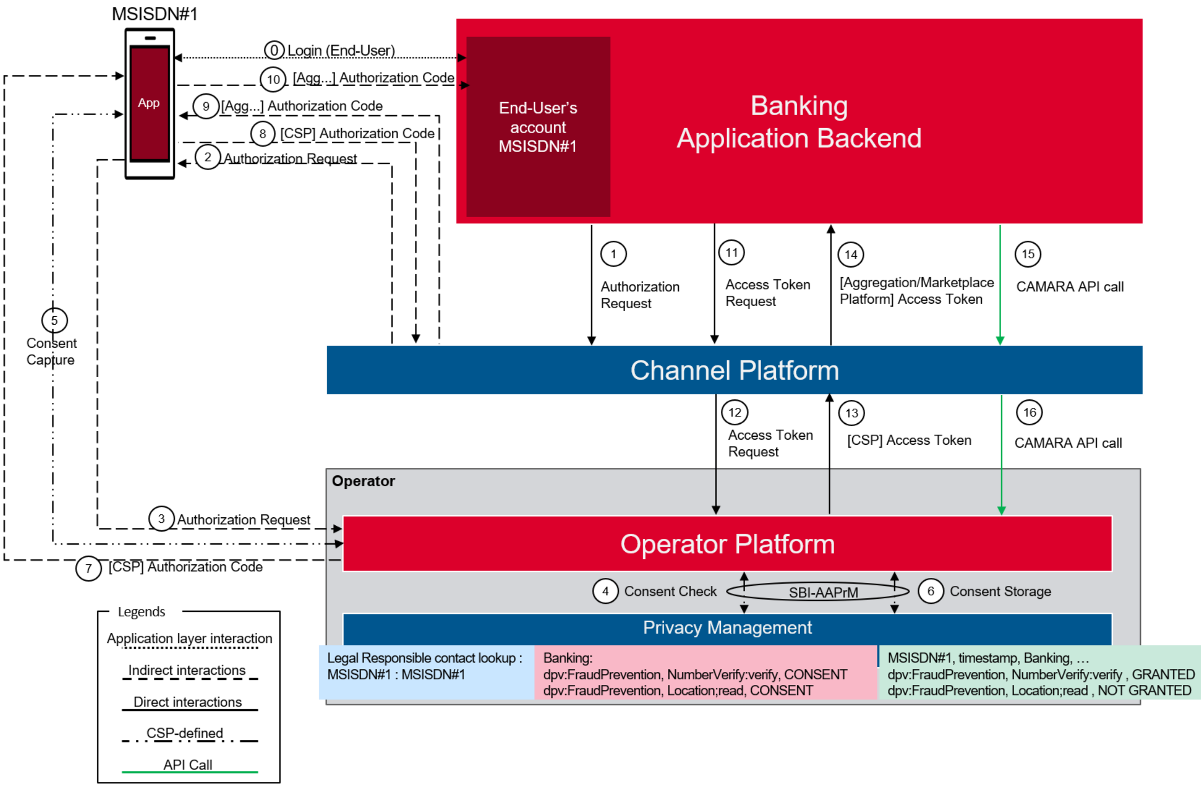


Figure 21: Consent capture - authorization code flow

In Figure 21, while the Application is trying to access some network capabilities (exposed via a CAMARA API),

- (step 1) an authorization request is sent by the Application Backend to the Channel Platform,
- (step 2-3) the Application is redirected to the right Operator's Authentication/Authorization (via the OGW Platform),

- (step 4) the OGW Platform will check via SBI-AAPrM the Application Privacy Profile to find out whether Consent is needed as a suitable legal basis and, if so, whether it is in place or needs to be captured,
- (step 5) if Consent is missing, a dialog is presented to the End-User for explicit opt-in, and the End-User may grant the Consent.

Note: the actual mechanisms for interacting with the End-User for notifying and capturing the Consent are left to the Operator.

- (step 6) the result of the End-User opt-in will be stored (e.g., for auditing processes) within the Privacy Information records,
- (step 7-8) Authentication/Authorization Function (via the OGW Platform) will send to the Channel Platform a first [Operator] authorization code (this code will be later on exchanged for a [Operator] access token),
- (step 9-10) the Channel Platform will send to the Application Backend a second [Channel Platform] authorization code (this code will be later on exchanged for a [Channel Platform] access token),
- (step 11) the Application Backend sends a request to exchange the [Channel Platform] authorization code for an [Channel Platform] access token,
- (step 12) the Channel Platform sends a request to exchange the [Operator] authorization code for a [Operator] access token,
- (step 13) Authentication/Authorization Function (via the OGW Platform) sends a [Operator] access token to the Channel Platform,
- (step 14) the Channel Platform will send an [Channel Platform] access token to the Application Backend,
- (step 15) the Application Backend sends a Service API request with the [Channel Platform] access token
- (step 16) the Channel Platform sends a Service API request with the [Operator] access token

More details about this particular scenario are shown in Figure 22.

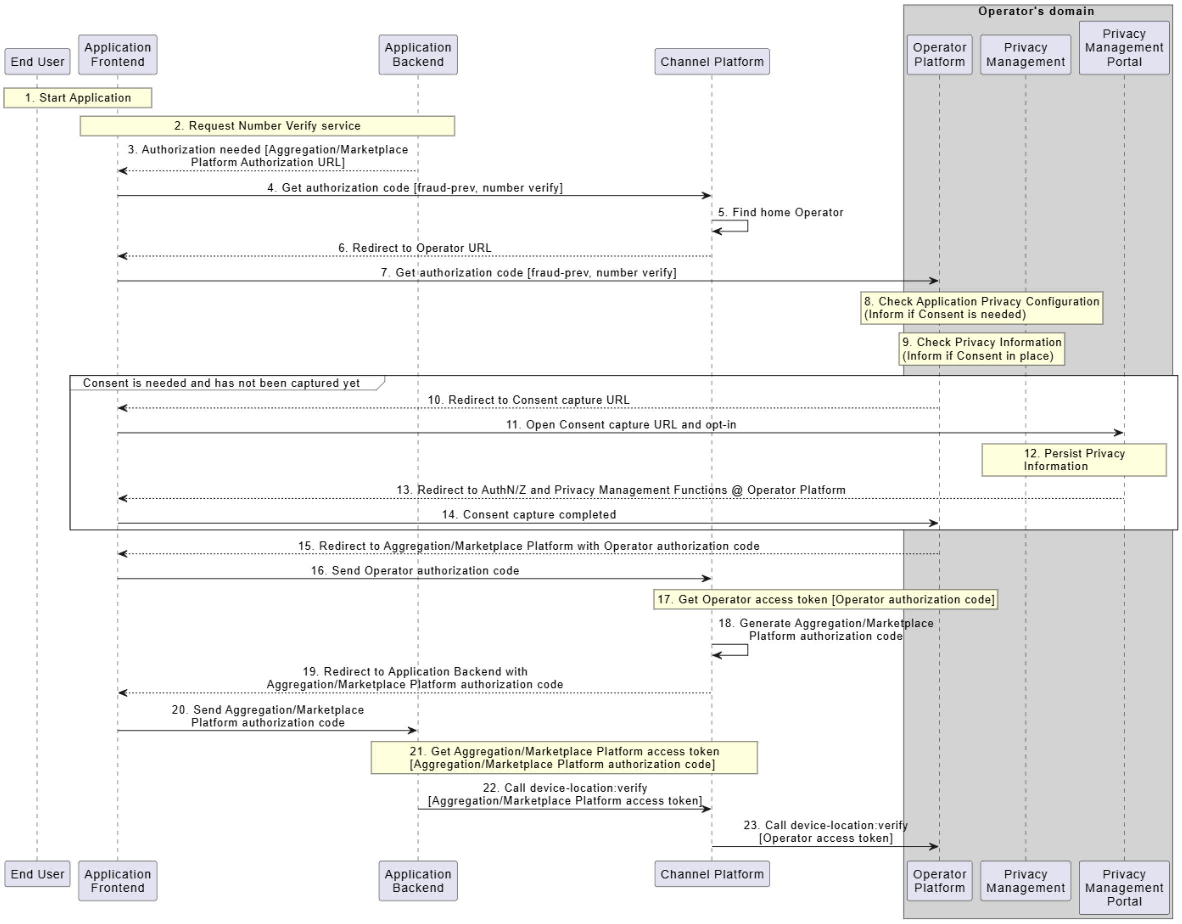


Figure 22: In-flow Consent capture

As shown in Figure 22, if some Operator Capabilities are needed while using an application (e.g., Number Verify in steps 1-2), the Application Backend redirects the Application Frontend to the authorization endpoint of the Channel Platform providing a redirect URI pointing to the Application Backend to whom an [Channel Platform] authorization code should be sent eventually (steps 3-4) and including also the Purpose of Data Processing. Each Purpose of Data Processing is associated with a legal basis based on the local regulations.

Upon reception of the Authorization request (step 4), the Channel Platform discovers the home Operator that provides the connection to the device where the Application Frontend is running (step 5). Once the home Operator is discovered (e.g., using the API Call Routing Functions in section 2.2.1.4), the Channel Platform itself initiates an Authorization code flow with the Operator redirecting the Application Frontend to the OGW Platform's authorization endpoint (see section 2.2.1.3) providing a redirect URI pointing to the Channel Platform to whom an [Operator] authorization code should be sent eventually and including also the Purpose of Data Processing (steps 6-7).

Upon reception of the authorization request, a network-based authentication may take place to obtain a network-specific identifier. Based on the declared purpose and the local regulations, the OGW Platform checks (in step 8) via SBI-AAPrM what is the applicable legal basis (e.g., "legitimate interest", "Consent", etc.). If the applicable legal basis is Consent, then the OGW Platform queries the Privacy Management Function over the SBI-AAPrM interface to check whether the relevant Privacy Information is already in place (step 9).

If Consent needs to be captured, the Application Frontend is redirected to the Operator Consent capture portal wherein the End-User needs to explicitly opt-in (steps 10-11). Based on the End-Users decision / selection, the Privacy Information is persisted in the Privacy Management Function in the Operator domain (step 12) and the flow control is given back to the OGW Platform with an indication that the Consent was successfully captured (steps 13-14).

If the Consent is in place, then an Operator Authorization Code is sent back to the Channel Platform (steps 15-16) and that Operator Authorization Code can be later exchanged for an Operator access token (step 17). After obtaining the Operator Authorization Code, the Channel Platform generates its own Authorization Code (step 18) which is sent back to the Application Backend (step 19-20).

If the Consent is not granted by the End-User (for instance the user does not agree with sharing their information for the signalled Purpose of Data Processing) the authorization process fails and the access to the personal information resources is denied.

Upon receiving the Channel Platform Authorization Code, the Application Backend obtains an access token bound to the Channel Platform (step 21) by exchanging the code for a token. Before delivering the access token to the Application Backend, the Channel Platform needs to exchange the Operator Authorization Code for an Operator access token (step 17).

Now the Application Backend holds a valid [Channel Platform] access token which can be used to invoke a CAMARA API towards the Channel Platform (step 22). The Channel Platform will decrypt the access token, check the relevant claims and obtain the routing information within.

Finally, the Channel Platform will use the [Operator] access token obtained earlier in the CAMARA API request to the corresponding OGW Platform (step 23) where the [Operator] access token will be validated and the request will be further processed (e.g., through Transformation Functions as indicated in section 2.2.4).

4.2 Out-of-band Consent capture in a different device

Another use case that may be covered takes place when a redirection cannot be applied to the device being used by the End-User since e.g., the person entitled to consent access to protected data is reachable via other device. This is shown in Figure 23.

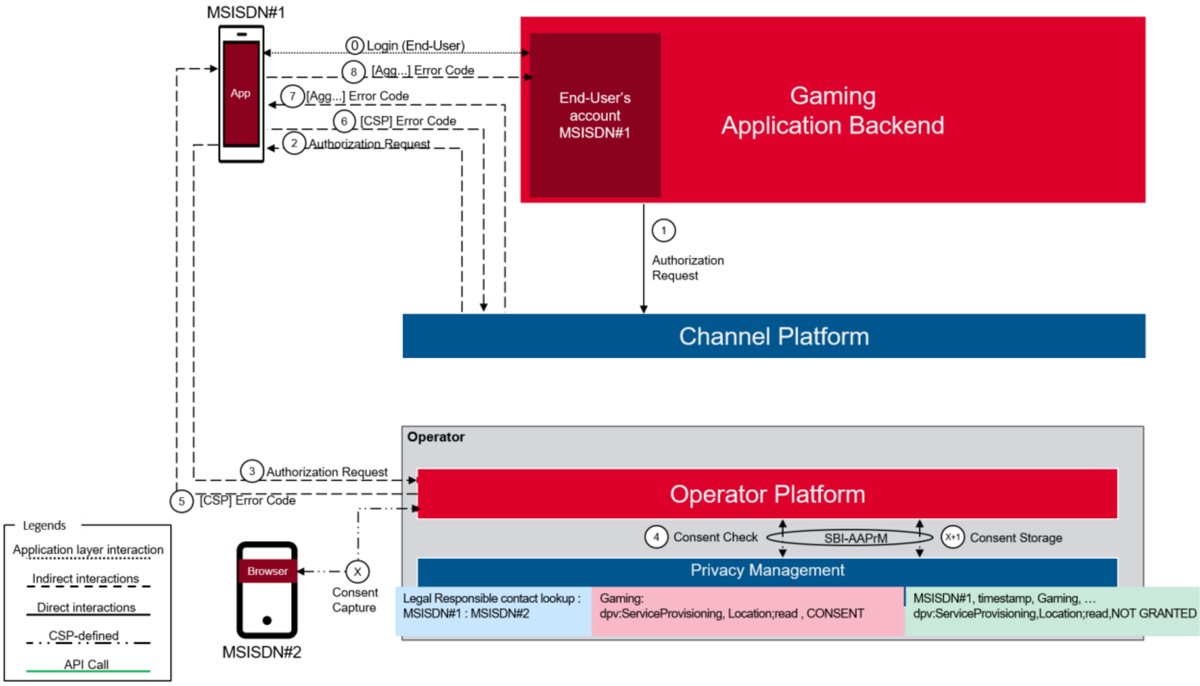


Figure 23: Consent capture is done in a different device

In-flow Consent capture (as in section 4.1) cannot happen if the person entitled to consent access to protected data is in a different device as shown in Figure 23. If Consent is missing, the authorization code generation will fail, and an error will be returned to the device as response to the authorization code request. An out-of-band mechanism must be introduced to inform the person entitled to consent access to protected data that an explicit opt-in is needed.

In Figure 23, while the Application is trying to access some network capabilities (exposed via a CAMARA API),

- (step 1) an authorization request is sent by the Application Backend to the Channel Platform,
- (step 2-3) the Application is redirected to the right Operator's Authentication/Authorization (via the OGW Platform),
- (step 4) the OGW Platform will check via SBI-AAPrM the Application Privacy Profile to find out whether Consent is needed as a suitable legal basis and, if so, whether it is in place or needs to be captured. If Consent is missing, a lookup for finding the person entitled to consent access to protected data is performed,
- (step 5-6) If person entitled to grant Consent is reachable via a different device, a [Operator] error code is sent to the Channel Platform,
- (step 7-8) an [Channel] error code is sent to the Application Backend
- (step X) an out-of-band notification is sent to the person entitled to grant Consent for capturing the Consent.

Note: the actual mechanisms for interacting with the End-User for notifying and capturing the Consent are left to the Operator.

- (step X+1) the result of the End-User opt-in will be stored (e.g., for auditing processes) within the Privacy Information records.

As shown in Figure 24, once Consent is granted, for a subsequent Authorization Request (steps 1-3), a [Operator] authorization code is delivered to the Channel Platform (steps 5-6) and a [Channel Platform] authorization code is delivered to the Application Backend (steps 7-8). Once an authorization code is received, the Application Backend will exchange it for an access token (steps 9-12) which will be subsequently used for API calls (step 13-14).

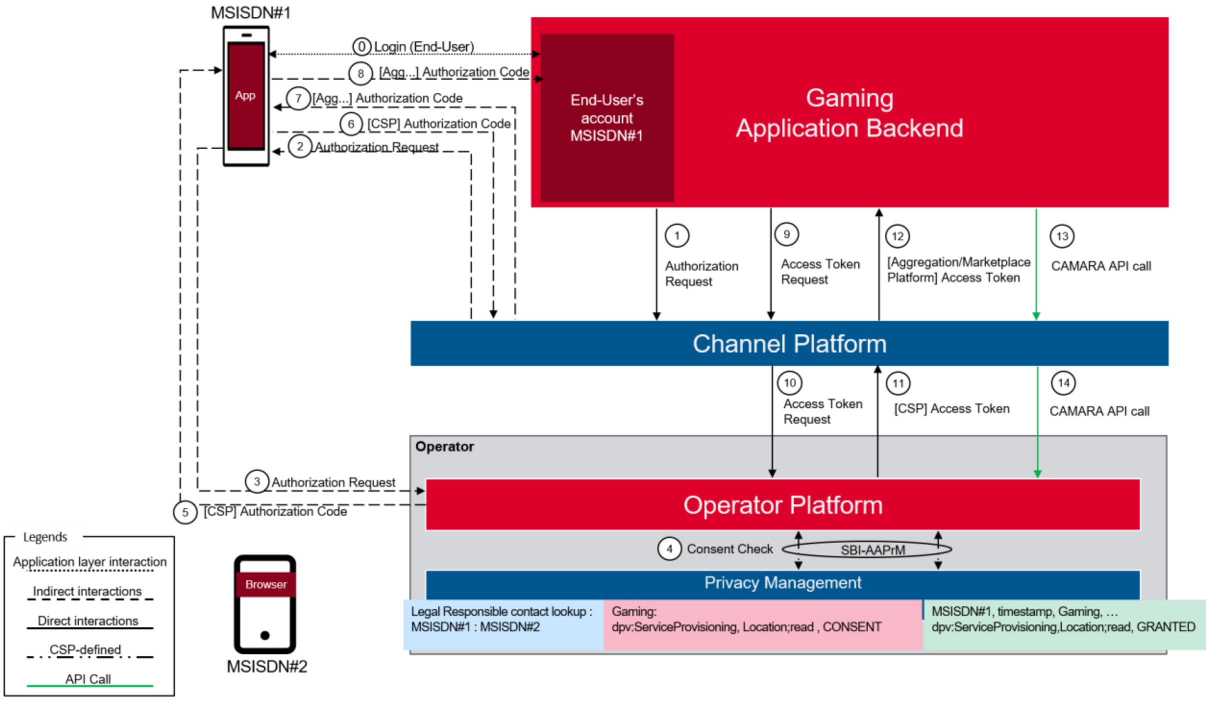


Figure 24: Consent has been granted in a different device

4.3 Out-of-band Consent capture using Client-Initiated Backchannel Authentication

This type of Consent capture leverages the mechanisms provided by OIDC CIBA flow [27].

A first scenario covered by this type of Consent capture happens when the End-User and the person entitled to consent access to protected data are in different devices and the privacy-sensitive information is bound to a third device.

As shown in Figure 25, End-User#1 provides the Consent after an out-of-band notification (e.g., an email) received in a different device (under control of the same End-User#1). Until Consent is granted, the Application Backend polls the token endpoints to retrieve an access token which will be used for CAMARA API calls subsequently.

Note that the same sequence would be also applicable if the End-User#1 devices are the same, while accessing a privacy-sensitive resource bound to another device, or if a second End-User (e.g., End-User#2) is in control of the device receiving the out-of-band notification.

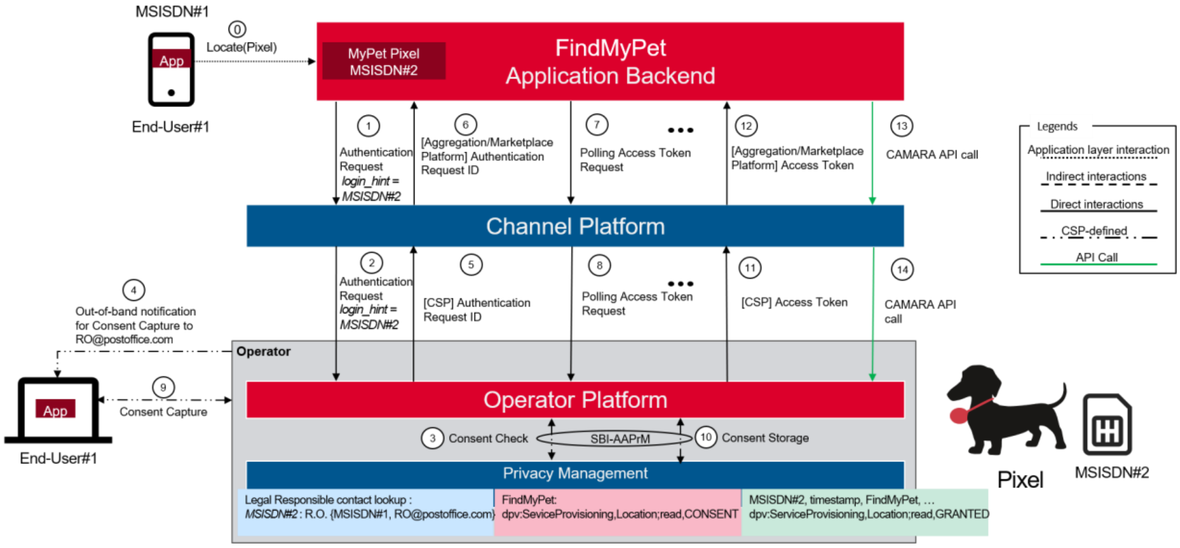


Figure 25: Consent capture: Client-Initiated Backchannel Authentication

In Figure 25, while the Application is trying to access some network capabilities (exposed via a CAMARA API),

- (step 1) an authorization request is sent by the Application Backend to the Channel Platform including a hint of the device associated with the privacy-sensitive data which the Application Backend is trying to obtain access (i.e., MSISDN#2),
- (step 2) the Channel Platform finds the corresponding Operator and forwards the authorization request to the respective OGW Platform,
- (step 3) the OGW Platform will check via SBI-AAPrM the Application Privacy Profile to find out whether Consent is needed as a suitable legal basis and, if so, whether it is in place or needs to be captured. If Consent is missing, a lookup for finding the person entitled to consent access to protected data is performed,
- (step 4) an out-of-band notification is sent to the person entitled to grant Consent (e.g., an email in Figure 25).

Note: the actual mechanisms for interacting with the End-User for notifying and capturing the Consent are left to the Operator

- (step 5) a [Operator] authentication request identifier is sent from the Authentication/Authorization (via the OGW Platform) to the Channel Platform (this identifier will be used for the subsequent polling in step 8),
- (step 6) a [Channel Platform] authentication request identifier is sent from to Application Backend (this identifier will be used for the subsequent polling in step 7),
- (step 7) until Consent is granted, the Application Backend will poll the token endpoint providing the [Channel Platform] authentication request identifier,
- (step 8) until Consent is granted, the Channel Platform will poll the token endpoint providing the [Operator] authentication request identifier,
- (step 9) the person entitled to grant Consent is requested for explicit opt-in,
- (step 10) the result of previous opt-in operation will be stored (e.g., for auditing processes) within the Privacy Information records,

- (step 11) the Authentication/Authorization Function (via the OGW Platform) sends a [Operator] access token to the Channel Platform,
- (step 12) the Channel Platform will send an [Channel Platform] access token to the Application Backend,
- (step 13) the Application Backend sends a Service API request with the [Channel Platform] access token
- (step 14) the Channel Platform sends a Service API request with the [Operator] access token

More details about the flow are provided in Figure 27.

Another scenario in which OIDC CIBA flow [27] may be applicable is when two (or more) End-Users (interacting across an Application) target each other's privacy-sensitive information (e.g., location), and each End-User needs to provide Consent for letting the Application to share privacy-sensitive information among End-Users. This scenario (for the case of two End-Users) is shown in Figure 26.

The use case in Figure 26 depicts also the aspect of split responsibility. The Application Provider is responsible for collecting Consent from the End-Users to be located under the Application execution context and maintain their "friends" relationship at the Application level. Especially this means that in a preparation phase, the End-Users, must have used the Application and given Consent for being located and for keeping their "friends" status at the Application level.

The location resource for End-User#2's device is in the responsibility of the Operator. Here the Operator must collect Consent from End-User#2 for exposing the device location to the Application.

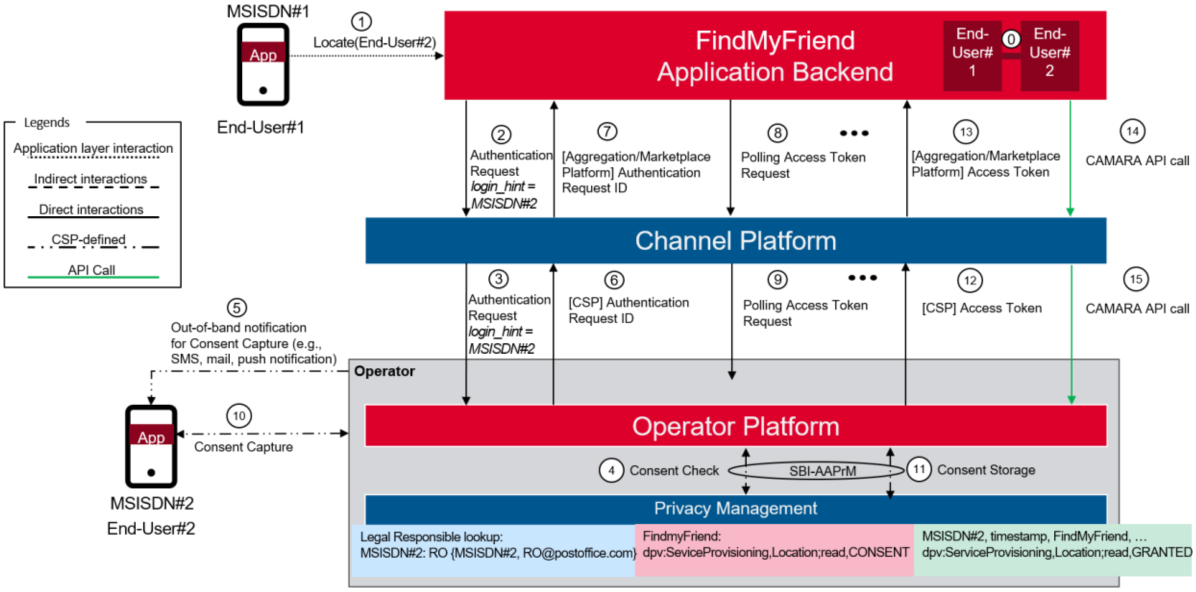


Figure 26: Consent capture: FindmyFriend-alike use case

As shown in Figure 26, both End-Users agree to keep a "friend" association at the Application layer (step 0). Both End-Users own their location data (or any other privacy-sensitive information). Both End-Users may or may not be Subscribers of the same

Operator. To address the right Operator, a hint for the End-User being targeted must be provided.

- (step 0) Both End-Users agree to keep a “friend” association at the Application layer
- (step 1) End-User#1 wants to locate End-User#2,
- (step 2) an authorization request is sent by the Application Backend to the Channel Platform including a hint of the device associated with the privacy-sensitive data which the Application Backend is trying to obtain access (i.e., MSISDN#2),
- (step 3) the Channel Platform finds the corresponding Operator and forwards the authorization request to the respective OGW Platform,
- (step 4) the OGW Platform will check via SBI-AAPrM the Application Privacy Profile to find out whether Consent is needed as a suitable legal basis and, if so, whether it is in place or needs to be captured. If Consent is missing, a lookup for finding the person entitled to consent access to protected data is performed,
- (step 5) an out-of-band notification is sent to the person entitled to grant Consent (e.g., SMS to MSISDN#2 in Figure 26).

Note: the actual mechanisms for interacting with the End-User for notifying and capturing the Consent are left to the Operator

- (step 6) a [Operator] authentication request identifier is sent from the Authentication/Authorization Function (via the OGW Platform) to the Channel Platform (this identifier will be used for the subsequent polling in step 9),
- (step 7) an [Channel Platform] authentication request identifier is sent from to Application Backend (this identifier will be used for the subsequent polling in step 8),
- (step 8) until Consent is granted, the Application Backend will poll the token endpoint providing the [Channel Platform] authentication request identifier,
- (step 9) until Consent is granted, the Channel Platform will poll the token endpoint providing the [Operator] authentication request identifier,
- (step 10) the person entitled to grant Consent is requested for explicit opt-in,
- (step 11) the result of the previous opt-in operation will be stored (e.g., for auditing processes) within the Privacy Information records,
- (step 12) the Authentication/Authorization Function (via the OGW Platform) sends a [Operator] access token to the Channel Platform,
- (step 13) the Channel Platform will send an [Channel Platform] access token to the Application Backend,
- (step 14) the Application Backend sends a Service API request with the [Channel Platform] access token
- (step 15) the Channel Platform sends a Service API request with the [Operator] access token

Please note that in Figure 26, the End-User#1 may be a Subscriber of a different operator, and the device End-User#1 is using for interacting with the Application may not be a smartphone.

More details about the OIDC CIBA flow are provided in Figure 27.

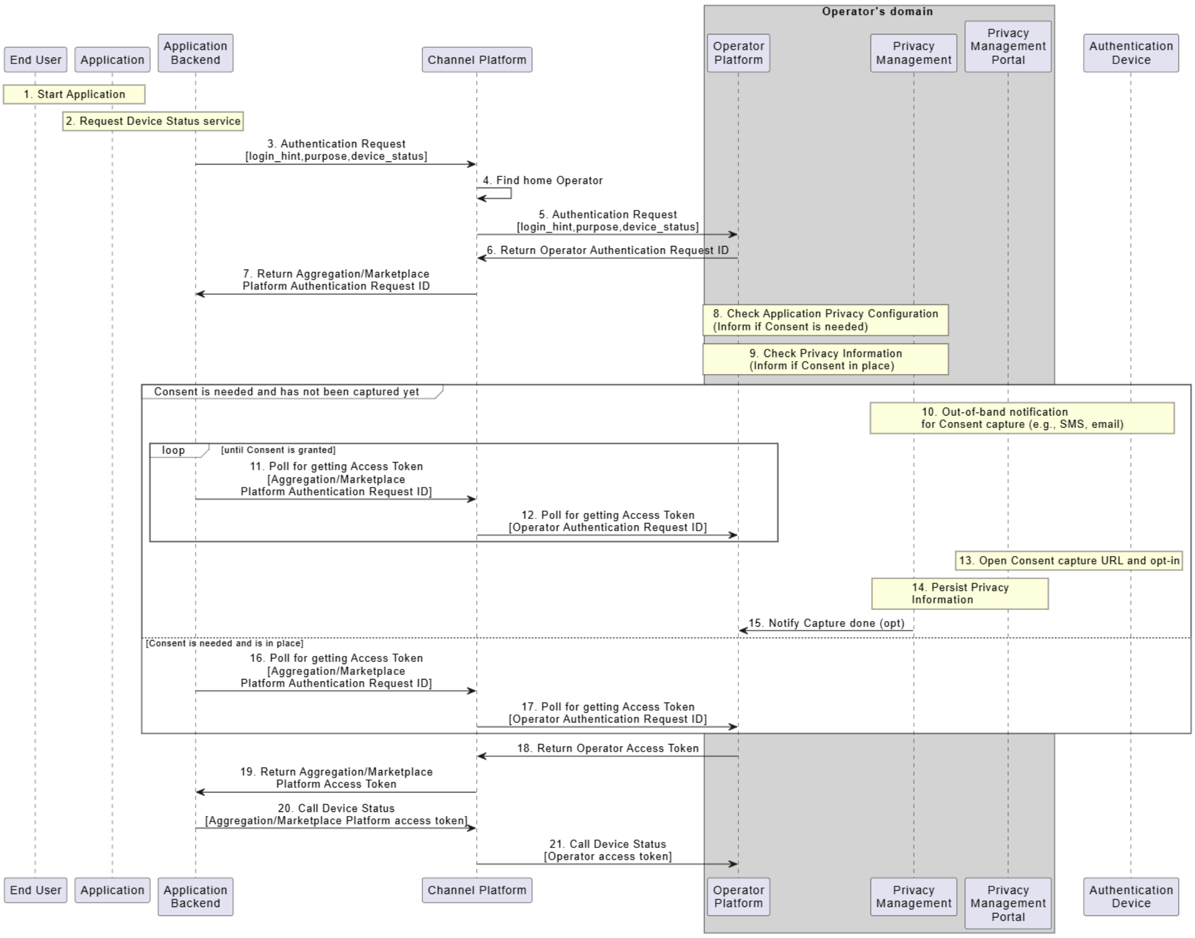


Figure 27: Out-of-band Consent capture

As shown in Figure 27, if some Operator Capabilities are needed while using an Application (e.g., Device Status in steps 1-2), the Application Backend sends an Authentication Request [27] including a valid identifier of the user for whom authentication is being requested (login_hint), the Application credentials and the Purpose of Data Processing (step 3). In step 4, e.g., using the API Call Routing Functions in section 2.2.1.4, the Channel Platform discovers the home Operator that provides the connection to the user identifier received in the previous step. Once the home Operator is discovered, the Channel Platform itself initiates an OpenID Connect CIBA flow with the Operator (step 5) sending an Authentication Request to the OGW Platform carrying same information received in step 3. The OGW Platform returns a [Operator] Authentication Request Identifier to the Channel Platform to indicate that the Authentication Request has been accepted and will be processed (step 6). The Channel Platform generates and returns to the Application Backend (in step 7) an [Channel Platform] Authentication Request Identifier while storing the relevant Operator routing information.

Based on the declared purpose and the local regulations, the OGW Platform checks (in step 8) via SBI-AAPrM what is the applicable legal basis (e.g., “legitimate interest”, “Consent”, “Contract”, etc.). If the applicable legal basis is Consent or legitimate interest (in some cases), then the OGW Platform queries the Privacy Management Function over the SBI-AAPrM interface to check whether the relevant Privacy Information is already in place (step 9).

If Consent is to be captured, an out-of-band exchange will be triggered to notify the authentication device that an explicit action is requested for obtaining the Consent.

In the meanwhile, the Application Backend polls the Channel Platform (step 11) until the Consent is granted providing the [Channel Platform] Authentication Request Identifier which is then used to retrieve the [Operator] Authentication Request Identifier which is sent to the token endpoint (step 12).

Once the out-of-band notification is received, the person entitled to consent access to protected data may explicitly opt-in (step 13). The result of that decision / selection is persisted in the Privacy Management Function in the Operator domain as Privacy Information (step 14) and the flow control is given back to the OGW Platform with an indication that the Consent was successfully captured (steps 15).

Next time the token endpoint is polled by the Application Backend (steps 16-17), the OGW Platform will issue a [Operator] access token to the Channel Platform (step 18). In step 19, the Channel Platform returns to the Application Backend an [Channel Platform] access token which may include information for routing subsequent API calls coming from the Application Backend.

Now the Application Backend holds a valid [Channel Platform] access token which can be used to invoke a CAMARA API towards the Channel Platform (step 20). The Channel Platform will decrypt the access token, check the relevant claims, obtain the routing information within, and retrieve the [Operator] access token.

Finally, the Channel Platform will forward the CAMARA API request to the corresponding OGW Platform (step 21) where the [Operator] access token will be validated and the request will be further processed (e.g., through Transformation Functions as indicated in section 2.2.4).

4.4 Upfront Consent capture

Unlike the flows proposed in section 4.1 or section 4.3, the Consent check and capture is decoupled (in time) from an authorization process (to obtain an access token) and could be triggered even before the Application needs access to Operator capabilities.

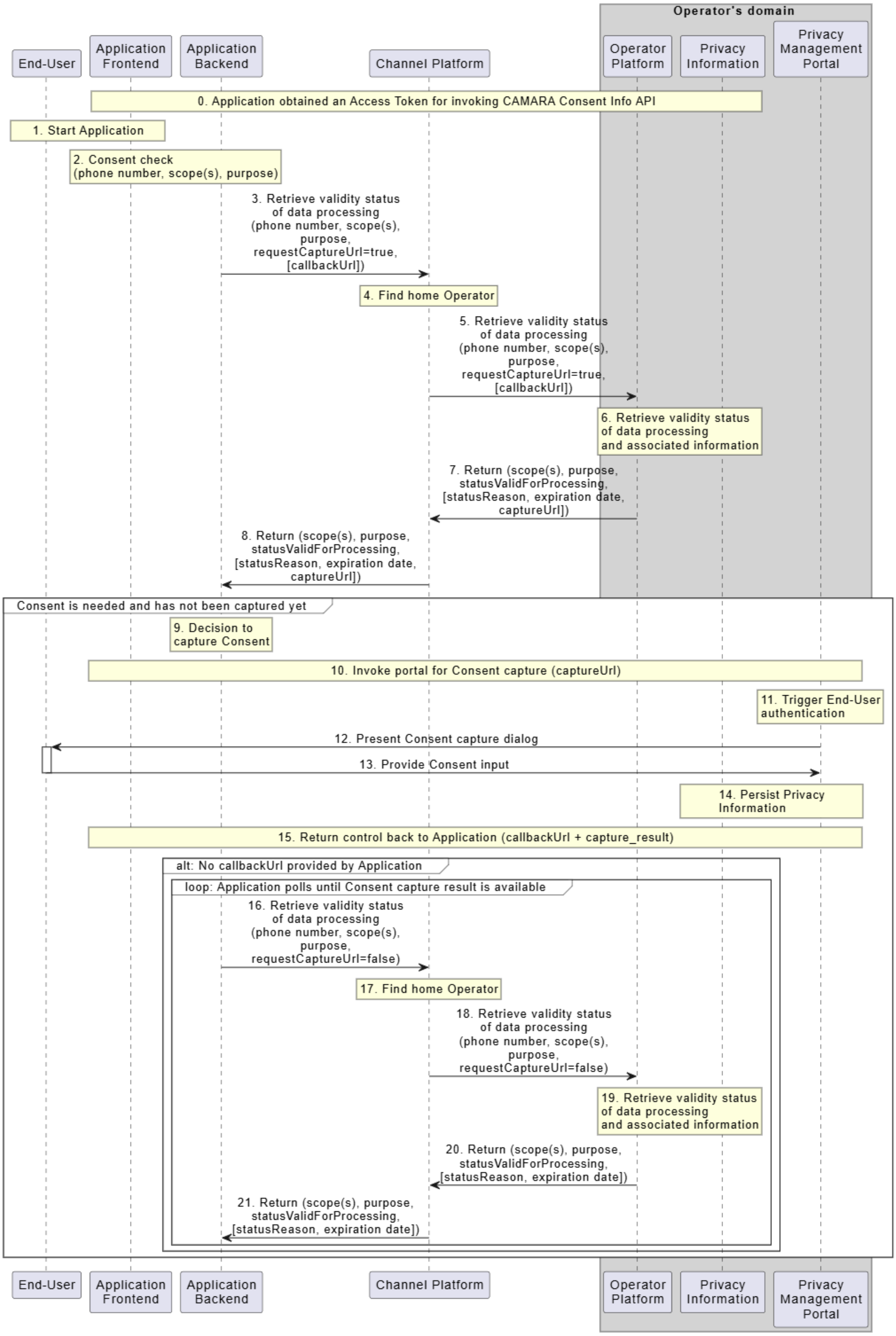


Figure 28: Upfront Consent capture

Pre-condition: the Application has already obtained an access token for calling CAMARA Consent Info API (step 0). The detailed use cases for this API are specified in section 4.4.1.

A decision to verify the validity status of data processing (which is held within the Operator domain under the Privacy Information) for a given End-User, scope(s) and purpose is made within the Application (step 1-2), depending on the scenario (e.g., Consent is to be obtained upfront). The Application Backend sends a request (step 3) to the Channel Platform for checking whether Privacy Information for the given End-User identifier is in place within the Operator domain. The request may indicate that a [Operator] Consent capture URL needs to be provided in the response if Consent is missing or expired, and may also include a callback URL to which the End-User should be redirected after providing the Consent input.

The Channel Platform determines the home Operator for the provided End-User identifier (step 4), for instance, by using the API Call Routing Functions described in section 2.2.1.4. Once the corresponding Operator is discovered, the Channel Platform forwards the request to the respective OGW Platform (step 5).

Upon receiving the request, the OGW Platform will query via SBI-AAPrM the Privacy Management Function (step 6) to check if there is Privacy Information in place associated with the provided filtering criteria (End-User identifier, scope and purpose). The result of this check (i.e. the status of data processing and the associated information) is then returned through the Channel Platform back to the Application Backend conditionally including the Operator Consent capture URL (step 7-9).

Note: more details about the transformation of the CAMARA Consent Info API call into SBI-AAPrM procedures could be captured in PRD OPG.09 [4].

Based on the Consent check result (e.g., Consent expired or missing), the Application will present the Consent capture dialog within the context of the Application execution (e.g., presenting an app-in webview or redirecting the End-User to the [Operator] Consent capture portal) as in step 10-12. Based on the presented dialog, the End-User will decide and select its Consent preferences (step 13) which will be persisted in the Privacy Management Function within the Operator domain (step 14).

After providing the Consent input, the End-User is redirected back to the Application (step 15) if a callback URL was provided by the Application in the initial request (step 3). This redirection includes a query parameter containing the Consent capture result (e.g., granted, denied, etc). Otherwise, if no callback URL was provided, the Application needs to poll again the CAMARA Consent Info API until the Consent capture result is available (step 16-21) to determine the final outcome of the Consent capture process and to confirm whether the data processing is permitted. Further information on the functional behaviour of the callback mechanism and its security considerations can be found in the Consent Info API specification [31].

4.4.1 CAMARA Consent Info API use cases

The CAMARA Consent Info API [31] provides a standardized mechanism for Application Providers to validate whether they have the necessary permissions to process End-User's personal data for a specific purpose before invoking other CAMARA Service APIs. This enables an upfront validation step that improves the developer and End-User experience and ensures compliance with privacy regulations.

The API is designed as a generic mechanism to retrieve the validity status of the data processing for a given End-User, a set of scopes, and a specific purpose. It provides a simple boolean (`statusValidForProcessing`) response regarding the validity status. When applicable (e.g., End-User Consent is missing or expired), the API can also provide a Consent capture URL (`captureUrl`) to direct the End-User to the Operator's Consent management channel when requested by the Application (`requestCaptureUrl` request parameter set to `true`). The Application can optionally provide a `callbackUrl` when requesting a Consent capture URL. This allows the Operator to redirect the End-User back to the Application's specified URL once the Consent capture process is completed, along with the outcome of the process (`capture_result`). Further details and examples of the API request and response format and fields can be found in [31].

It should be noted that when the combination of the requested scope(s) and purpose is not permitted for the Application Provider — for example, if they have not onboarded the relevant API(s) with the Operator for the declared purpose — the Consent Info API will return an error (see [31] for details), because the necessary permissions to access the requested information are not in place.

This API does not delegate Consent capture to Application Providers but rather empowers them to present the Operator's Consent capture URL at the most opportune time and place. The actual Consent capture occurs within the Operator's secure environment, ensuring the End-User's authentication with the Operator.

The Consent Info API supports two primary use cases, determined by the value of the `requestCaptureUrl` request parameter.

4.4.1.1 Trigger Consent Capture

This use case applies when an Application, operating under a Consent legal basis, wants to check for Consent and, if it is not present or valid, proactively initiate the capture flow. The Application Backend sets `requestCaptureUrl = true` in the request.

The flow, as depicted in Figure 29, proceeds as follows:

1. The Application Backend sends a request to the Consent Info API with `requestCaptureUrl = true` and, optionally, a `callbackUrl` to which the End-User should be redirected after the Consent capture is completed (this corresponds to step 3 in Figure 28).

2. The OGW Platform checks the End-User's Consent status for the given scopes and purpose (corresponding to step 6 in Figure 28).

3. Based on the status, one of several scenarios may occur:

    a. **Scenario A: Consent is valid.** The OGW Platform returns `statusValidForProcessing = true`. The captureUrl is not provided. The Application can proceed to request a Service API access token.

b. **Scenario B: Consent is invalid** (e.g., Denied). The OGW Platform returns `statusValidForProcessing = false` and a `statusReason` of `DENIED`. No `captureUrl` is provided. The Application must handle this by, for example, informing the End-User that the feature is unavailable.

c. **Scenario C: Consent is missing or expired.** The OGW Platform returns `statusValidForProcessing = false` and a `statusReason` of `PENDING` or `EXPIRED`. Because `requestCaptureUrl` was `true`, the response includes the `captureUrl` (corresponding to steps 7-9 in Figure 28).

4. In Scenario C, the Application redirects the End-User (e.g., via an in-app browser) to the provided `captureUrl` to complete the Consent journey managed by the Operator (e.g., at the Privacy Management Portal).

5. After the End-User completes the Consent journey, the Application retrieves the capture result through one of two mechanisms, depending on whether a `callbackUrl` was provided in step 1:

   c) With `callbackUrl`: The Operator redirects the End-User back to the Application at the provided URL, appending a `capture_result` query parameter with the outcome of the Consent capture process (e.g., `granted`, `denied`, etc). No polling is needed. The result can map to Scenario A (Consent granted) or Scenario B (Consent explicitly denied by the End-User during the journey).

   d) Without callbackUrl: The Application Backend must poll the Consent Info API again to check for a status update (as described in section 4.4.1.2). Polling can result in Scenario A (Consent granted) or Scenario B (Consent explicitly denied by the End-User in the journey).

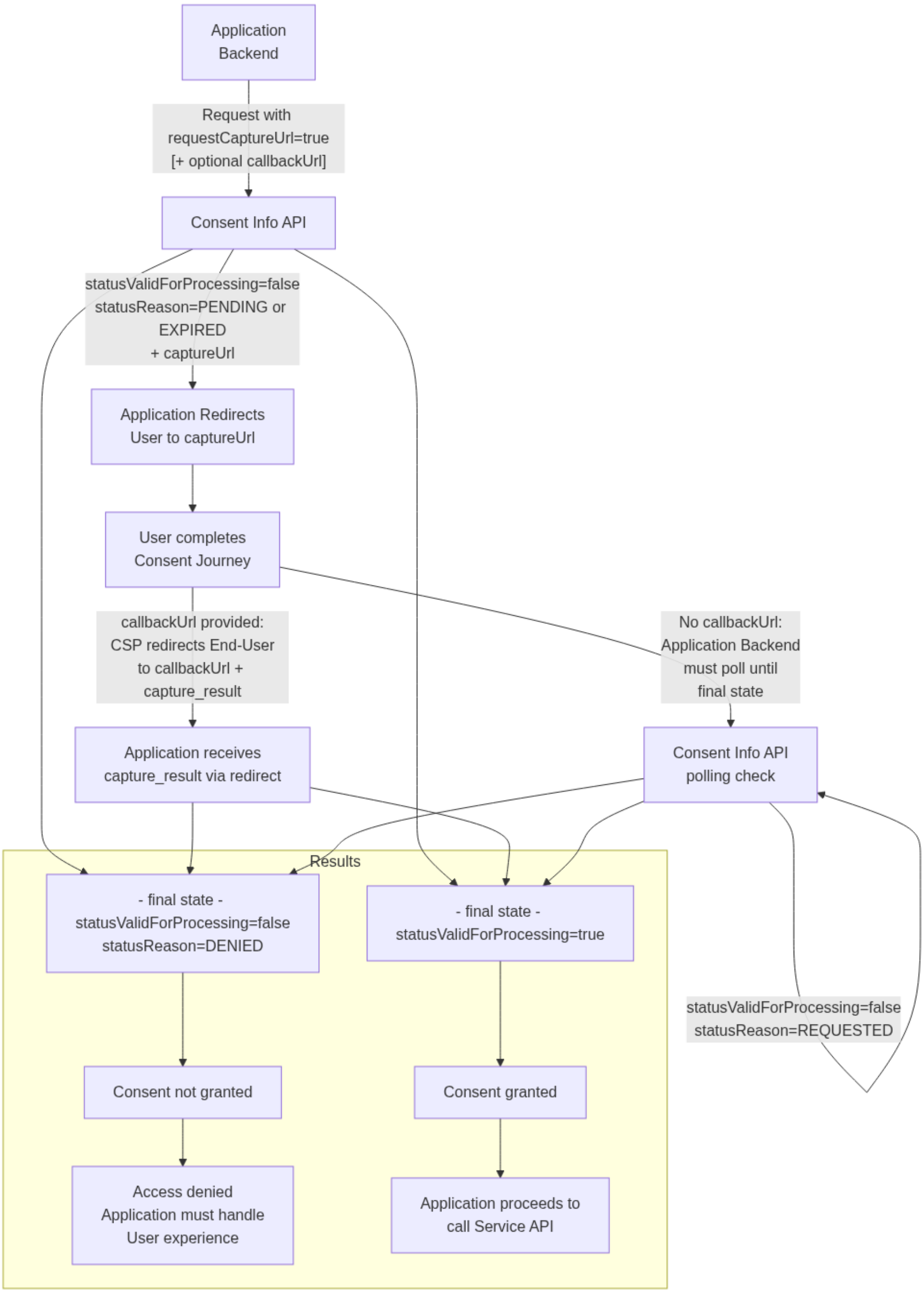


Figure 29 Trigger Consent Capture

4.4.1.2 Data Processing Status Check

This use case applies when an Application Backend wants to validate in advance if data processing is permitted, without intending to trigger a new Consent flow. This is useful for checking status under *any* legal basis (e.g., Legitimate Interest) or when polling a previously triggered Consent capture flow, as described in section 4.4.1.1. The Application Backend sets `requestCaptureUrl = false` in the request.

The flow, as depicted in Figure 30, is a simple check:

1. The Application Backend sends a request to the Consent Info API with `requestCaptureUrl = false`.
2. The OGW Platform checks the End-User's data processing status.
3. The OGW Platform responds with the status:
    a. **Scenario A: Processing is allowed.** The OGW Platform returns `statusValidForProcessing = true`. The Application can proceed.
    b. **Scenario B: Processing is not allowed (final state).** The OGW Platform returns `statusValidForProcessing = false` and a `statusReason` of `DENIED` or `OBJECTED` (e.g., End-User has opted-out). No `captureUrl` is provided. The Application must handle the End-User experience accordingly.
    c. **Scenario C: Processing is not allowed (pending state).** The OGW Platform returns `statusValidForProcessing = false` and a `statusReason` of `PENDING` (e.g., Consent was never captured), `REQUESTED` or `EXPIRED`. No `captureUrl` is provided, as it was not requested.
4. In Scenario C, the Application is informed that processing is not allowed. It must then decide its next action. It can either trigger Use Case 1 (i.e., call the API again with `requestCaptureUrl = true`) to capture Consent , or it can attempt a standard token request (e.g., Authorization Code flow or CIBA flow), which will then prompt the End-User for Consent as part of that in-flow process (as described in section 4.1 or section 4.3).

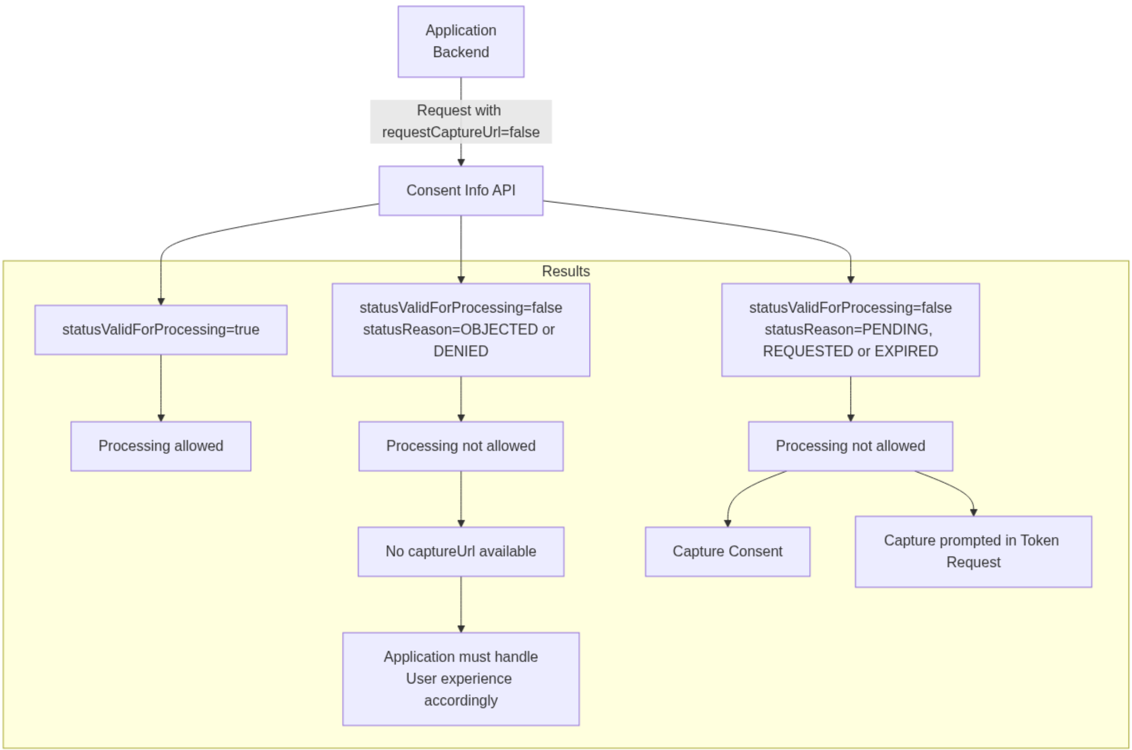


Figure 30 Data Processing Status Check

4.5 Delegated Consent capture

Delegated Consent capture is a controlled model that allows trusted Application Providers to capture End-User Consent directly within their own Applications. Unlike the procedures described in sections 4.1 to 4.4, where the Consent capture is managed through Operator-managed channels, this model enables a more flexible and integrated user experience while maintaining Operator accountability and legal compliance.

The key principles underlying this delegated model are:

- Access to this functionality is restricted to trusted Applications under Operator's policy and is subject to a whitelisting, contractual controls, auditing or Vetting Processes.
- The Application is responsible for End-User authentication and for linking the End-User Application account to their corresponding Operator subscription and designing the overall user experience of the Consent capture flow within the Application.
- The Application must retrieve the legal Consent texts from the Operator via API. The Application presents these texts to the End-User using its own interface but is strictly prohibited from altering the legal content.
- The Operator remains fully responsible for the storage, audit, transparency, and opt-out management. The Application submits the End-User's response (evidence) back to the Operator for validation and registration in the Operator's Privacy Management system.
- The Consent validity period and expiration date are determined and controlled by the Operator according to its internal policies.
- The delegated model is limited to scenarios based on Consent as a legal basis and does not cover cases based on other legal bases, which are handled by the Operator through non-delegated mechanisms.
- This model complements the other Consent capture flows without affecting their availability.

Technically, the delegated Consent capture is realised through the CAMARA Consent Management API [34]. The Figure 31 below illustrates the delegated Consent capture flow:

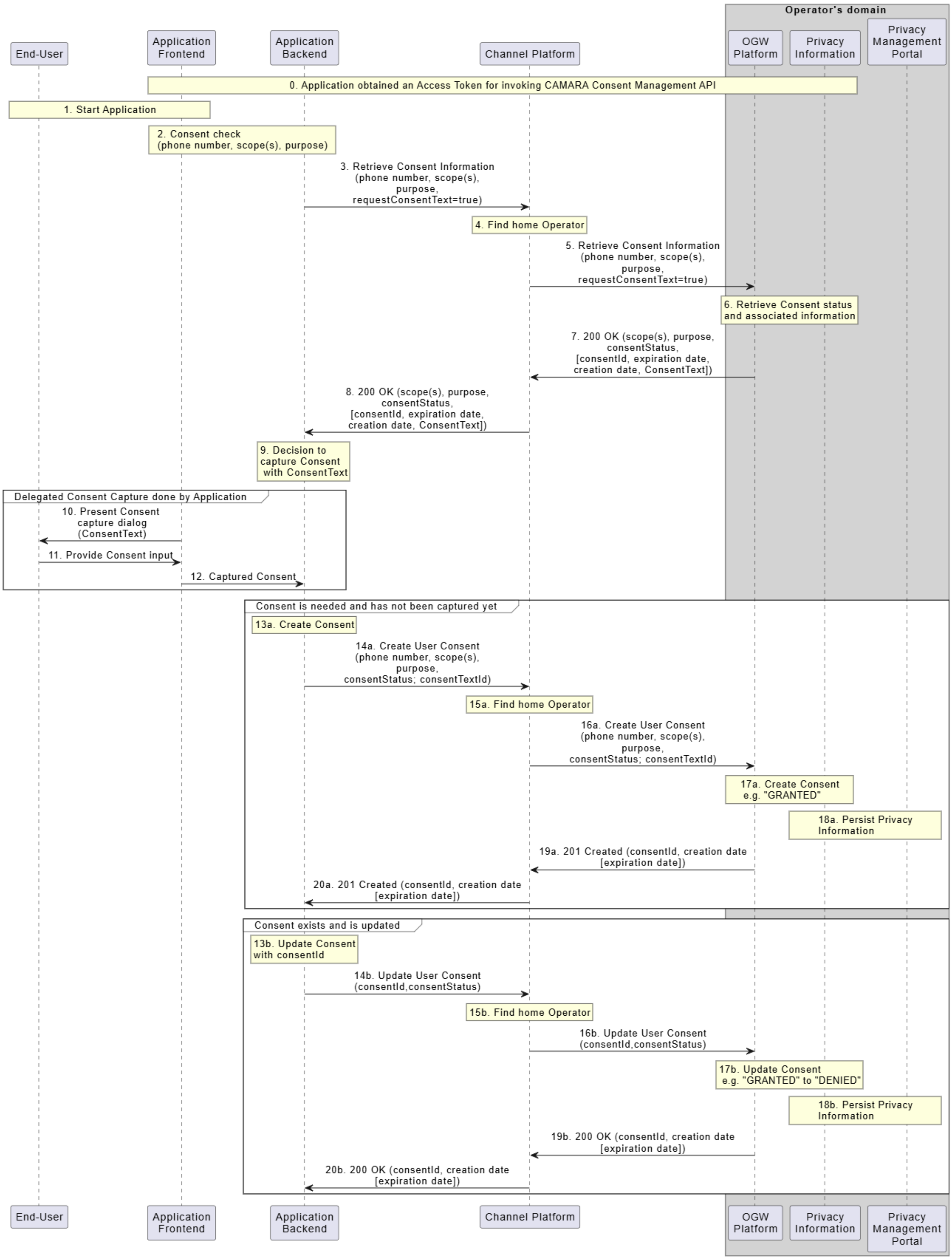


Figure 31: Delegated Consent capture

Pre-condition: The Application has been registered and whitelisted by the Operator for use the CAMARA Consent Management API for delegated Consent capture. The Application has obtained the necessary access token for API invocation. Section 4.5.1 specifies the detailed use cases for this API.

The Application, according to its internal logic and user experience design, decides to check the Consent status for a specific End-User (identified by phone number) and a given scope(s) and purpose (step 1-2). The Application Backend initiates the flow by invoking the CAMARA Consent Management API through the Channel Platform to retrieve the Consent information (step 3), including in this case the legal Consent texts to be presented to the End-User during the capture process.

The Channel Platform determines the home Operator for the provided End-User identifier (step 4), for instance, by using the API Call Routing Functions described in section 2.2.1.4. Once the corresponding Operator is discovered, the Channel Platform forwards the request to the respective OGW Platform (step 5).

Upon receiving the request, the OGW Platform will query via SBI-AAPrM the Privacy Management Function (step 6) to check if there is Privacy Information in place associated with the provided filtering criteria (phone number, scope and purpose). The OGW Platform then responds to the Channel Platform with the Privacy information, which may include the Consent status, the Consent ID, creation and expiration dates, and optionally the Consent texts for End-User presentation (step 7). The Channel Platform forwards this information to the Application Backend (step 8).

Note: more details about the transformation of the CAMARA Consent Management API call into SBI-AAPrM procedures could be captured in PRD OPG.09 [4].

Based on the retrieved information, the Application Backend decides whether to proceed with the delegated Consent capture or not. If the Application decides to proceed, the Application Frontend presents the End-User with a Consent capture dialog that includes the legal Consent texts provided by the Operator (step 10). The End-User provides their input (e.g., granting or denying Consent) through the Application interface (step 11), and the Application captures this input and sends it back to the Application Backend (step 12).

The Application Backend then determines whether this is a new Consent capture (i.e., there was no existing Consent) or an update to an existing Consent. If it is a new Consent, the Application Backend initiates the creation of the Consent by invoking the CAMARA Consent Management API through the Channel Platform (step 14a). The Channel Platform identifies the home Operator (step 15a) and forwards the request to the respective OGW Platform (step 16a). The OGW Platform processes the request, creates the Consent in its system, and persists the relevant information in the Privacy Management Function (step 17a). The OGW Platform then responds with a confirmation of the created Consent, including the Consent ID and creation date (and optionally expiration date) (step 19a), which is forwarded back to the Application Backend (step 20a).

If the Application Backend determines that this is an update to an existing Consent, it initiates the update process by invoking the CAMARA Consent Management API with the Consent ID and the new Consent status (step 14b). The Channel Platform identifies the home Operator (step 15b) and forwards the request to the respective OGW Platform (step 16b). The OGW Platform processes the update request, updates the Consent information in its system, and persists the changes in the Privacy Management Function (step 17b). The OGW Platform then responds with a confirmation of the updated Consent information (step 19b), which is forwarded back to the Application Backend (step 20b).

The End-User Consent is captured or updated in the Operator's system, and the Application is informed of the outcome. The Operator maintains the necessary records for audit and compliance purposes, and the End-User has a seamless experience while interacting with the Application.

4.5.1 CAMARA Consent Management API use cases

The CAMARA Consent Management API facilitates the creation of new Consents, specifying the requested scope(s) and purpose, the updating of the status of existing Consents, and the retrieval of Consent information. This information can optionally include Consent texts for End-User presentation during the capture process when requested by the Application (`requestConsentText = true`). Further details and examples of the API request and response formats and fields can be found in [34].

It should be noted that when the combination of the requested scope(s) and purpose is not permitted for the Application Provider — for example, if they have not onboarded the relevant API(s) with the Operator for the declared purpose — the Consent Management API will return an error (see [34] for details), because the necessary permissions to access the requested information are not in place.

If the requested combination of scope(s) and purpose is allowed for the Application Provider, but there is no matching API(s) under Consent legal basis, the API will return an empty Consent information response, indicating that there is no existing Consent for the provided criteria.

Additionally, the Operator may restrict the capture of delegated Consent for certain phone numbers based on its privacy policies or regulatory requirements. This may occur, for instance, when the requested phone number belongs to a minor on a family Operator plan where the Legal Responsible party may differ from the End-User of the Application or when the phone number is associated with a specific type of Operator subscription that is not eligible for delegated Consent capture. In such cases, the Operator, as the ultimate authority responsible for Consent validation, will determine that delegated Consent capture is not applicable for the requested phone number, and the Consent Management API will return an appropriate error to inform the Application (see [34] for details). The Application should handle this situation gracefully and, if required, redirect the process to the alternative non-delegated Consent capture flows managed directly by the Operator.

The Consent Management API supports several uses cases as described below.

4.5.1.1 Consent Capture

In this use case, the Application initiates the delegated Consent capture flow. The Application Backend invokes the Consent information retrieval operation of the CAMARA Consent Management API [34], specifying the End-User identifier, the requested scope(s) and purpose, and requesting that the legal Consent texts be included in the response so they can be presented to the End-User during the capture dialog. This corresponds to steps 3–8 in Figure 31.

When the Application initiates the delegated Consent capture flow, it must ensure compliance with the principles outlined in section 4.5.1, in particular that the legal Consent texts are retrieved from the Operator and presented to the End-User without alteration, and that the End-User's response is submitted back to the Operator for validation and registration.

The Figure 32 illustrates this use case:

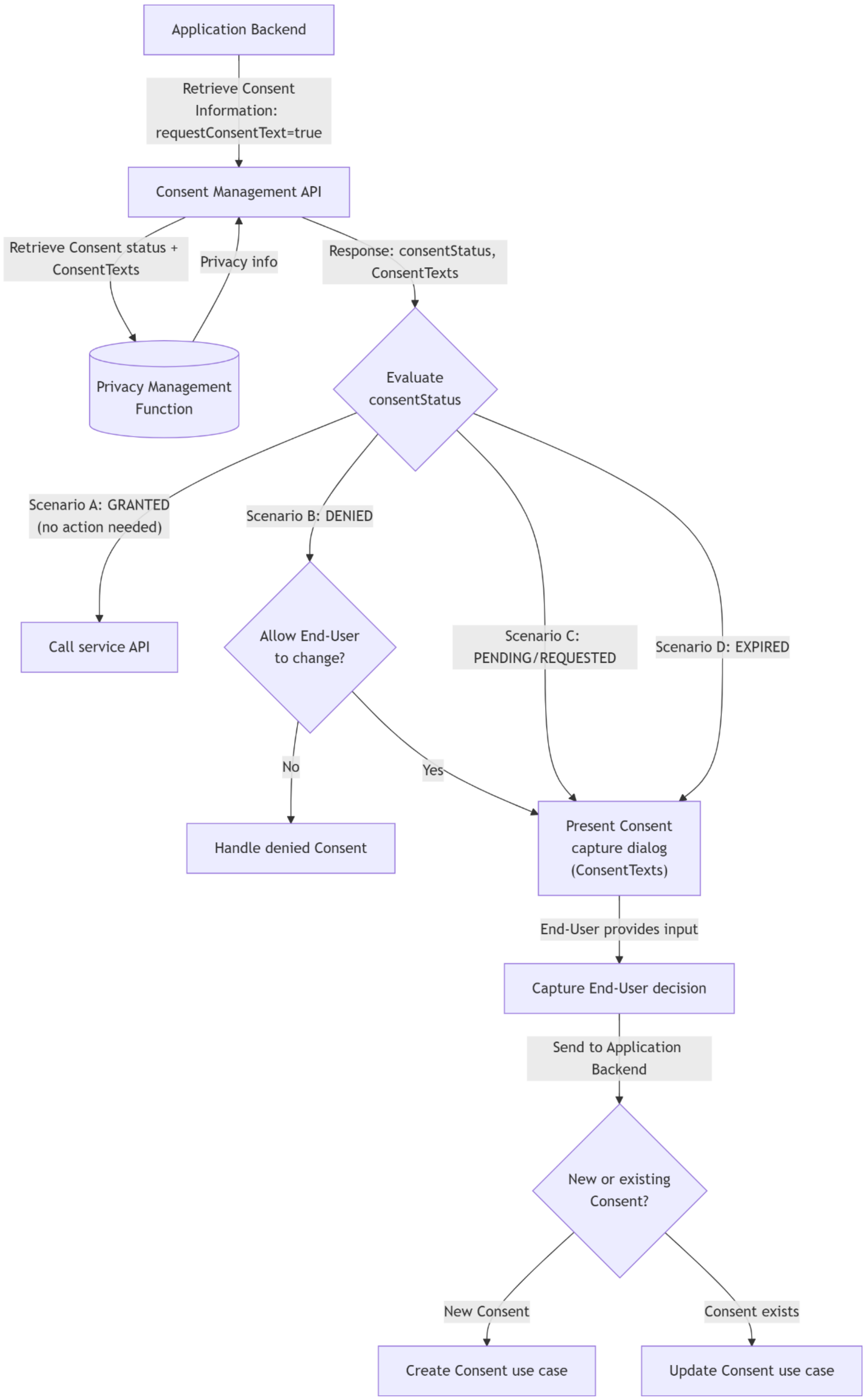


Figure 32: Consent Capture

1. The Application Backend sends a Retrieve Consent Information request to the Consent Management API with `requestConsentText = true`.

2. The OGW Platform retrieves the Privacy information from the Privacy Management Function, including the Consent status (`cosentStatus`) and the associated legal Consent texts (`consentText`). This corresponds to step 6 in Figure 31.

3. Based on the Consent status returned, the Application decides how to proceed:

   a. **Scenario A: Consent granted.** If the Consent status is `GRANTED` and the Application logic determines that no further action regarding Consent capture is needed, the flow ends here and the Application proceeds to call the relevant Service API(s) for the requested scope(s) and purpose.

   b. **Scenario B: Consent denied.** If the Consent status is `DENIED` and the Application logic determines that the End-User should be given the opportunity to change their decision, the flow proceeds to step 4. Otherwise, if the Application logic determines that no further action is needed, the flow ends here and the Application must handle the case of denied Consent accordingly.

   c. **Scenario C: Consent pending or requested.** If the Consent status is `PENDING` or `REQUESTED`,i.e. the Consent has not yet been captured or is awaiting End-User action, the flow proceeds to step 4.

   d. **Scenario D: Consent expired.** If the Consent status is `EXPIRED`, i.e. the Consent is no longer valid, the flow proceeds to step 4 to perform a new Consent capture.

4. If the Application decides to proceed with the delegated Consent capture, the Application Frontend presents the End-User with a dialog that includes the legal Consent texts retrieved from the Operator. The End-User provides their input (e.g., granting or denying Consent), and the Application captures this and sends it back to the Application Backend (steps 10-12 in Figure 31).

5. The Application Backend then determines whether this is a new Consent or an update to an existing one, and invokes the Consent Management API accordingly, as described in the Consent Creation use case (section 4.5.1.3) or the Consent Update use case (section 4.5.1.4).

If an Application is considered trusted and it is permitted to capture Consent in a delegated manner by the Operator, the JWT Bearer flow should be used to obtain an access token for the target Service API to which the captured Consent applies (JWT bearer flow detailed description can be found in section 3.1.1.3). This is aligned with CAMARA recommendations [8] for backend-to-backend use cases where direct End-User interaction is not needed at the time of the API request because Consent is not required or has already been obtained. If the Consent is not initially in place, the JWT Bearer flow will return an error indicating that Consent is required. The Application can then proceed with the delegated Consent capture flow, if needed, as described above.

4.5.1.2 Consent Status Check

This use case covers the scenario where the Application only needs to verify the Consent status for a given End-User, scope(s) and purpose, without necessarily performing a complete delegated capture process. Unlike section 4.5.1.1, where the Consent information retrieval is the entry point of a full capture flow, here the operation is invoked autonomously — for instance, when the Application needs to confirm whether a valid Consent already exists before invoking a Service API, or to monitor Consent status as part of a backend process (e.g. checking whether an existing Consent has expired). In this case `requestConsentText` will typically be set to `false`, as the Application does not intend to present a Consent capture dialog to the End-User. Further details on the API request and response formats can be found in [34].

The following diagram illustrates this use case:

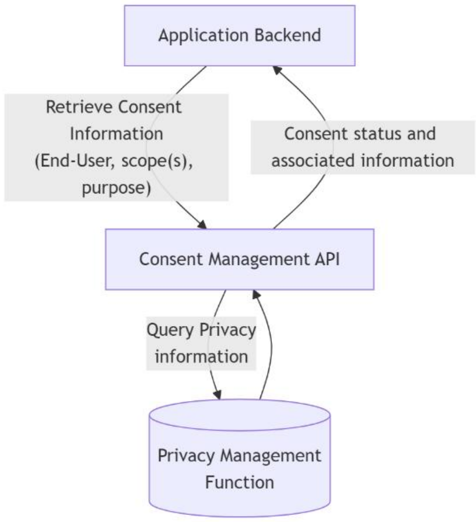


Figure 33: Consent Status Check

4.5.1.3 Consent Creation

In this use case, the Application Backend creates a new Consent for a specific End-User, scope(s) and purpose, following the completion of the Consent capture dialog (see section 4.5.1.1). The Application Backend invokes the Consent creation operation of the CAMARA Consent Management API [34], providing the End-User identifier, the requested scope(s), the purpose, the Consent status resulting from the capture dialog, and a reference to the specific version of the legal Consent texts that were presented to the End-User. This reference creates a traceable connection between the User's Consent and the exact legal text they were shown. This use case corresponds to steps 13a–20a in Figure 31.

The Application can only register a Consent status of `GRANTED` or `DENIED`. The Operator determines the Consent expiration date according to its internal policies; the Application cannot set or modify it. Further details on the request and response formats, including applicable error conditions, are described in [34].

The following diagram illustrates the Consent Creation flow:

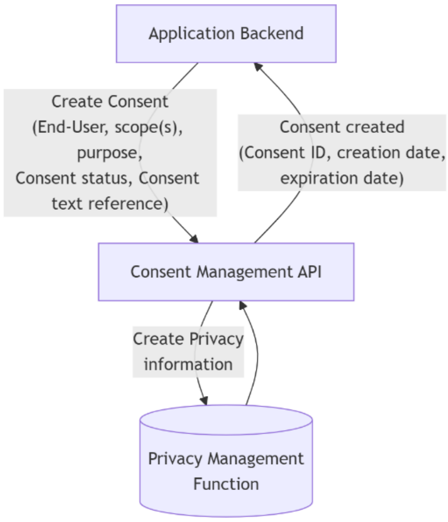


Figure 34: Consent Creation

4.5.1.4 Consent Update

In this use case, the Application Backend updates the status of an existing Consent for a specific End-User, following a change in the End-User's Consent decision — for example, revoking a previously granted Consent, re-consenting after a previous denial, or renewing an expired Consent. The Application Backend invokes the Consent update operation of the CAMARA Consent Management API [34], supplying the identifier of the existing Consent and the new Consent status. This use case corresponds to steps 13b–20b in Figure 31.

As with the Consent Creation use case, the Application can only set a Consent status of `GRANTED` or `DENIED`. The Operator determines the updated Consent expiration date according to its internal policies. Further details on the request and response formats, including applicable error conditions and allowed status transitions, are described in [34].

The following diagram illustrates the Consent Update flow:

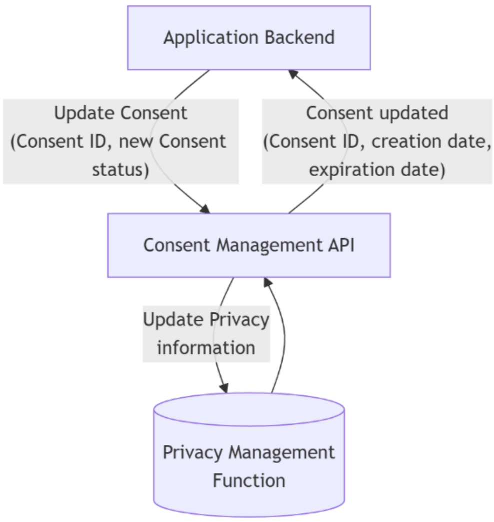


Figure 35: Consent Update

4.6 Consent Capture in federated environments

In this scenario, an End-User, an Application Provider, a Leading OGW Platform and a Partner OGW Platform are present, as shown in Figure 36. The Application Provider interacts with the Leading OGW Platform via NBI whereas the Leading OGW Platform and the Partner OGW Platform interact via EWBI. Additionally, the subscription of the End-User belongs to the Operator that owns the Partner OGW Platform (Operator domain 2 in Figure 36). In this scenario, the End-User is the person entitled to consent access to protected data, so the OAuth 2.0 authorization code flow could be leveraged.

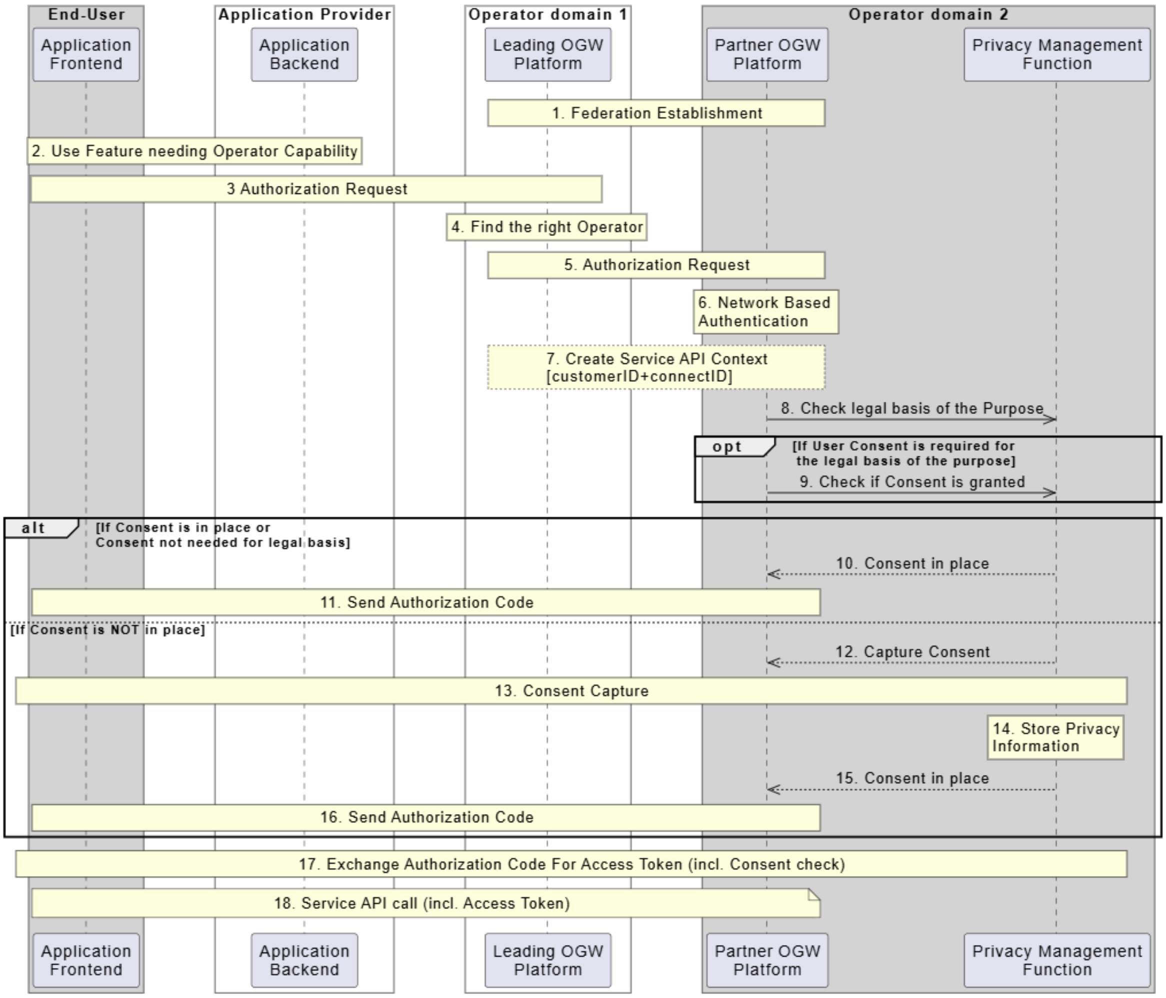


Figure 36: In-flow Consent capture in federated environments

As shown in Figure 36, first a federation needs to be established (Step 1) between the two Operators [32]. To let the Application access services and privacy-sensitive information hosted in the Partner OGW Platform, first an authorization request is sent to the Leading OGW Platform (Step 2,3). Based on a user identifier, the Leading OGW Platform can find the right Operator to forward the authorization request to (Step 4). The Leading OGW Platform generates a `customerID` [32] mapping to the Application Provider, which may be used in the Partner OGW Platform to provide more context to the End-User while capturing Consent (provided it is the applicable legal basis).

The Authorization request is sent to the right Partner OGW Platform (Step 5). Upon reception of the authorization request, a network-based authentication may take place in the Partner OGW Platform to obtain a network-specific identifier (Step 6). A Service API Context may be created (Step 7) considering the `customerID` (signalled by the Leading OGW Platform) and a `connectID` (generated at the Partner OGW Platform) [32]. The Service API Context may be used for logging and auditing purposes. If the applicable legal basis is Consent (Step 8), then the Partner OGW Platform queries the Privacy Management Function in the Operator domain over the SBI-AAPrM interface to check whether the relevant Privacy Information is already in place (Step 9).

If the Consent was already granted, then an Authorization Code is sent back (Step 10, 11) and that Authorization Code can be later exchanged for an Operator's access token (Step 17) that should be used for the Service API call in Step 18.

If there is no associated Privacy Information, the Privacy Management Function indicates that Consent capture from the End-User is needed (Step 12). Once the Consent is requested (Step 13), the result of that operation is stored in the Privacy Management Function (Step 14). If the Consent is not granted by the End-User (for instance the user does not agree with sharing its information for the signalled Purpose of Data Processing) the authorization process fails and the access to the privacy-sensitive information is denied. If the Consent is granted an Authorization Code is sent back (Step 15,16) which can be later exchanged for an Operator access token (Step 17). Upon reception of NBI API call, the request is routed to the EWBI (carrying the Operator access token) in Step 18, the Partner OGW Platform decrypts the access token and forwards the request in order to receive a response which is sent back.

Note: Even though in Figure 36 a frontend interaction is needed to capture the Consent from the End-User (e.g., an in-App Webview), there could be other Consent capture in federation scenarios which are for further study.

5 Use cases and Operational User Stories

The OGW Platform as a GSMA Operator Platform realisation (OGW) is designed to provide seamless integration between telecom Operators and external systems such as portals, marketplaces, and aggregators. To support this, the OGW Platform includes a comprehensive set of Operation, Administration, and Management (OAM) capabilities that streamline Service API management, resource provisioning, and service delivery. These capabilities ensure the OGW Platform supports critical business operations like onboarding Application Providers, managing Service API access, monitoring usage, and handling billing processes. By enabling real-time usage monitoring, flexible billing models, and automated invoicing, the OGW Platform simplifies the management of Service API consumption and financial transactions, ensuring a smooth and efficient integration process for both telecom Operators and external partners. The following requirements detail the functionalities necessary for the OGW Platform to effectively manage the lifecycle of applications, API access, usage, and billing.

The next section describes the use case requirements for the OGW Platform.

5.1 Integration to the OGW Platform

Aggregators are an ideal go-to-market for Operators seeking to sell their Service APIs to a broad range of developers who may not wish to integrate individually with each of them. For more detailed information refer to the GSMA Open Gateway Channel Partner Onboarding Guide [5].

For effective aggregator integration, the OGW Platform shall expose the TMF Operate APIs (e.g., TMF931 [19]). These Operate APIs allow aggregators to handle orchestration and management of resources across different service providers, ensuring standardised communication and interoperability in multi-vendor environments. Additionally, to allow direct connection for Application Provider, a dedicated marketplace or portal can be used, providing an interface where developers can easily discover, access, and manage Service API offerings. This setup will promote a more efficient ecosystem, empowering developers to integrate telecom network services into their applications with minimal friction while ensuring scalability and consistent service quality across platforms. An elaboration on how to consume the Operate APIs is detailed in section 3.1.2.

5.2 Developer / Application Provider management

5.2.1 Application Provider Onboarding

The OGW Platform shall enable functionality to onboard new Application Providers, allowing them to register their organisation, set up credentials, and configure API offerings that they require for integrating with network services.

5.2.2 Application Provider Inquiry

The OGW Platform shall provide an inquiry function that allows authorised users to search for and retrieve detailed information about registered Application Providers, including their profile, contact information, and active services.

5.2.3 Application Provider Update

The OGW Platform shall support the functionality for updating Application Provider details. This includes changing contact information, updating business credentials, and reflecting these changes across all associated applications and API access.

5.2.4 Application Provider Deactivation

The OGW Platform shall allow deactivation of an entire Application Provider entity, deactivating all associated applications, services, and API access linked to the Application Provider in one streamlined process.

5.3 Application management

5.3.1 Application onboarding

The OGW Platform shall allow existing Application Providers to create new applications, providing a user interface for configuring application settings, linking resources, and selecting relevant network capabilities for the application.

5.3.2 API Ordering

The OGW Platform shall enable Application Providers to order APIs for their existing applications. This includes the ability to browse available APIs, request access, and configure the integration within the application settings.

5.3.3 Application Inquiry

The OGW Platform shall allow Application Providers and administrators to query details about existing applications, such as configuration, status, associated API subscriptions, and usage statistics.

5.3.4 Application Update

The OGW Platform shall provide the ability for Application Providers to update the configuration of their applications. This includes modifying application settings, changing API subscriptions, and adjusting resource allocations.

5.3.5 API Access Removal

The OGW Platform shall allow Application Providers to remove API access for their applications when no longer needed. The system should manage the de-provisioning of the API, ensuring that access is securely revoked.

5.3.6 Application Deactivation

The OGW Platform shall provide a feature for deactivating applications. This shall include the ability to gracefully deactivate the application, stopping all active services, and revoking API access while preserving application data for future reference in line with local regulatory requirements.

5.4 Order management

5.4.1 Product Order Inquiry

The OGW Platform shall offer functionality to inquire about the status of product orders. This may include real-time updates on order progress, service provisioning, and activation timelines.

5.4.2 Product catalogue

The OGW Platform shall provide a searchable catalogue of products and services, including APIs. Application Providers shall be able to view product features, pricing, and service-level agreements, helping them make informed decisions about service subscriptions.

5.4.3 API Access Product Modification Ordering

The OGW Platform shall allow Application Providers to modify their existing API access orders. This includes upgrading service tiers, adding new API functionalities, or downgrading to remove unnecessary features.

5.5 Catalogue Management

5.5.1 API Product definition

The OGW Platform may provide functionality for the definition and management of API products. This includes the ability to define and group individual APIs into products that can be offered to consumers or external partners

5.5.2 Catalogue management functions

The OGW Platform may provide comprehensive catalogue management functionality, enabling API product owners to organise, update, and manage API products in a structured manner. This includes categorising APIs, supporting version control, managing access control.

5.6 Usage Monitoring

5.6.1 Real time usage monitoring

The OGW Platform may provide real-time usage monitoring and reporting capabilities for applications and API access to authorised parties. This shall include metrics such as API call volumes, response times, data consumption, and performance trends, giving Application Providers visibility into their resource utilisation.

5.6.2 Usage limits

The OGW Platform may support alerts for API usage limits or performance thresholds being reached, enabling proactive management and adjustments by Application Providers to avoid service disruption or overuse costs.

5.7 Billing and Payment

5.7.1 Real time charging

The OGW Platform may support real-time charging information for API usage, providing Application Providers with detailed breakdowns of charges based on consumption (e.g., API calls, bandwidth, or data usage).

5.7.2 Billing models

The OGW Platform may support multiple billing models, such as pay-per-use, subscription-based, and tiered pricing options, enabling flexibility based on API consumption patterns.

5.7.3 Payment gateway integration

The OGW Platform may integrate with a payment gateway to allow for seamless payment processing, including features for managing payment methods (e.g., credit cards, direct debits), invoices, and transaction history.

5.7.4 Automated invoicing

The OGW Platform may provide automated invoicing, generating invoices based on API consumption at predefined intervals (e.g., monthly, quarterly). The invoicing should reflect detailed charges and be exportable in various formats

The OGW Platform may allow Application Providers to view their billing history and manage overdue payments, including receiving notifications or alerts for pending invoices or reaching usage limits that might incur additional charges.

5.7.5 Audit logs

The OGW Platform shall provide audit logs for all transactions and may provide logs for billable activities, ensuring compliance with regulations. API Service Performance Monitoring and Assurance.

5.7.6 Real-Time Performance Monitoring

The OGW Platform shall provide real-time monitoring of critical metrics, including latency, bandwidth, API response times, and application uptime and service performance.

5.7.7 Threshold Configuration

The OGW Platform may provide functions to set performance thresholds for critical metrics to monitor service performance. These metrics can be varied per API.

The OGW Platform may trigger alerts when these thresholds are breached.

5.7.8 Historical Data and Reporting

The OGW Platform may store historical performance data and allow to generate reports for analysis adhering to local regulations Platforms

5.7.9 FCAPS management

The OGW Platform may also implement a comprehensive set of Fault, Configuration, Accounting, Performance, and Security (FCAPS) management functionalities to ensure robust network and service management of the OGW Platform.

5.8 Performance Measurement

5.8.1 Business Outcomes

Poor API performance can impact API adoption and usage in the following ways:

1. Contributes to poor user experience. Delays in API responses means that application features such as buttons, pages, or data-heavy features (e.g., search, chat, dashboards) appear unresponsive. End-Users perceive the app as slow or broken, even if everything else works correctly.
2. Impacts End-User retention and conversion rates. A Google study [28] found that as page load time goes from 1 second to 3 seconds, the probability of bounce increases 32%. Application Providers may leverage Network APIs during critical services such as account sign-up or online payments where conversion metrics are monitored closely. For the Application Provider, the financial value of the Network API must outweigh any loss of conversion due to additional latency.

5.8.2 Performance Framework

This section defines a common methodology, including metrics, to define and measure API performance. Specific requirements on API performance (for example, API Response Time per API call) are out of scope, however recommended targets are provided, and these are based on best practice in high transactional API industries.

5.8.3 Scope

- Metrics and how they are measured are described from the perspective of OGW Platform Owners (Operators) and Aggregators.
- Monitoring for availability should only include successful 'final responses' as deemed appropriate by the API Consumer. For example, 200, 204 responses or friendly 400 error responses.
- Exception error responses such as 50x should be excluded.
- Subscription creation, as in Device Status and Geofencing APIs should follow the same metrics.

5.8.4 Metrics

Metrics and how they are measured are described from the perspective of OGW Platform Owners and Aggregators.

To measure API performance, the following metrics are used:

Metric	Measure	Unit	Responsible for Measurement
API Response Time	Percentiles P50, P95, P99	Milliseconds	Leading Operator
Total API Latency	Percentiles P50, P95, P99	Milliseconds	API Initiator / Aggregator / Leading Operator
Availability	Percentage of time a given API is available, divided by the total time it's expected to be available, then multiplied by 100.	Percentage	Leading Operator
Success Rate	The number of successful API requests divided by the total number of attempts, then multiplied by 100. 'Success' is defined in section 5.8.8.	Percentage	API Initiator / Aggregator / Leading Operator

Table 3 Performance Metrics

Metrics of API Response Time, Total API Latency and Success Rate may be measured using Synthetic Monitoring deployed and maintained by the northbound Aggregators or OGW Platform Owners.

5.8.5 API Response Time

5.8.5.1 Model Definition

API Response Time is the delay incurred at an API Endpoint between receiving and responding to an API request.

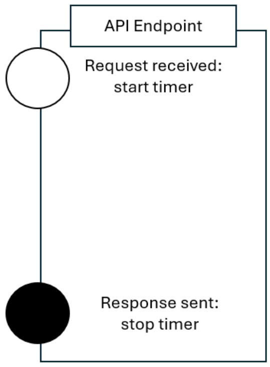


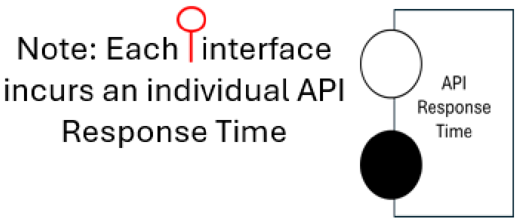
Figure 37 API Response Time for an API endpoint.

API Response Time at a given API endpoint is calculated as follows:

*API Response Time = (API Endpoint dispatch epoch − API Endpoint receive epoch) * 1000*

5.8.5.2 API Response Time in the Open Gateway System

The model in the previous section measures the API Response Time for a single API endpoint. An Open Gateway System may involve a sequence of API calls between multiple API endpoints to fulfil the initiating API Consumer's request, as per the scenarios described in section 3. The following legend applies to each diagram:



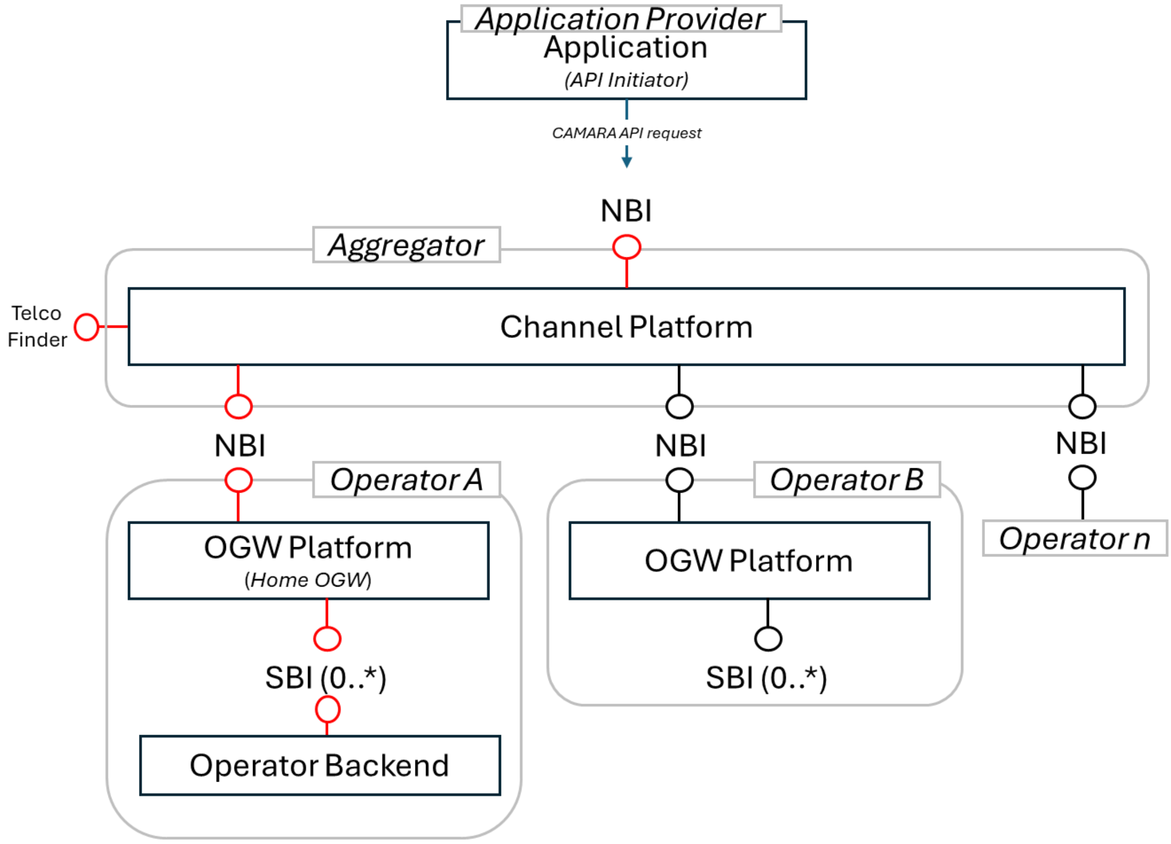


Figure 38 Aggregation model (Multiple OGW Platform Owners)

In the scenario in Figure 38 a Channel Platform receives the request from the API Initiator (Application) at its NBI. The interfaces used to fulfil the request are coloured red in the diagram and include an API request to the Telco Finder to determine the Target OGW Platform to route to. API Response Time can be measured at each of these individual interfaces and contribute to the API Response Time measured at the Channel Platform NBI. Note this is not the same as the overall API Latency experienced by the API Initiator, which is covered under the heading 'Total API Latency' below.

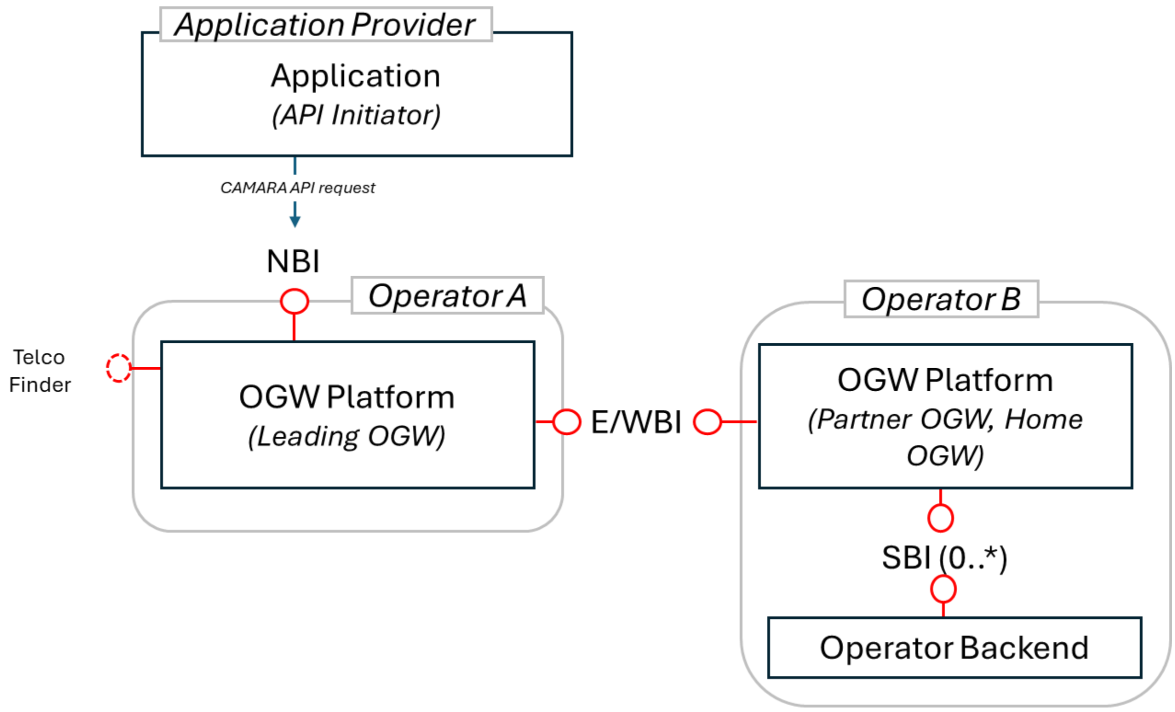


Figure 39 Partner OGW Platform Owner model (East/West federation)

The scenario in Figure 39 is similar to the one described in clause 3.1, with one OGW Platform Owner (Operator A) playing the role of the Aggregator. The difference is that in this scenario, the East/Westbound Interface is used to route the request between the Leading OGW Platform Owner (the Operator playing the role of Aggregator) and the OGW Platform of the Subscriber's Operator (the Operator able to fulfil the initial API request, as it is the Home Operator of the target user identifier).

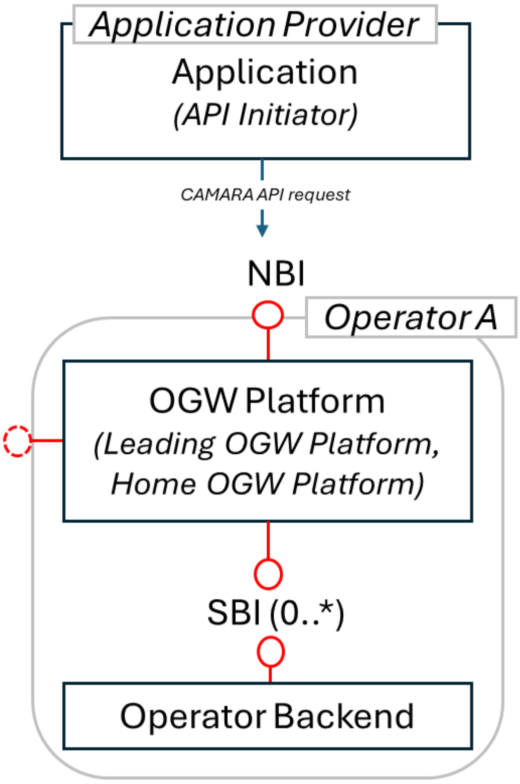


Figure 40 Direct API Initiator model

In the scenario in Figure 40, the OGW Platform Owner exposing the NBI to the Application is also the Home Operator of the target user identifier. Fulfilment of the API call will typically involve querying one or more SBIs (e.g. to retrieve data or request a network action) and may involve a Telco Finder API call in establishing that the OGW Platform Owner is the Home Operator. Each SBI has a distinct API endpoint with its own API Response Time. Hence when the NBI queries the SBI(s), the NBI's API Response Time will depend on the API Response Time of the SBI(s), the network delay between NBI and SBI(s), and processing delay at the NBI.

Alternatively, the NBI may fulfil the API request from local cache or throw an immediate exception (4XX or NBI server unavailable), in which case the SBI is not required to be called.

5.8.5.3 Components of API Response Time

- Delay in authorization of the API Consumer's API request (access token validation).
- Delay in checking for Consent (and potentially acquiring Consent).
- Network delay, which is itself composed of propagation delay, serialization delay, and network queuing. Note this refers to any network delay in forwarding the API request between parties involved in the flow, including routing of requests to a roaming customer's home network. It does not include the Internet hop when the initiating API Consumer makes the API request.
- Server infrastructure load (e.g. CPU, memory, bandwidth)
- Route discovery delay, where the Northbound API establishes where to forward the API request (i.e. to the Southbound interface or an East/West interface)

- API fulfilment delay. This is composed of the delay in creating a response to the API request, typically by querying internal network functions. Considerations for fulfilment delay include:
  - The number of actors involved in routing and fulfilling an HTTP request, each adding network delay and additional authorization delay.
  - Synchronous vs. asynchronous API calls. An asynchronous API call may immediately return an 'Accepted' acknowledgement, however, that is not fulfilment.
  - Caching, which may be appropriate for certain APIs but not others.
  - Quality of response - for example, an accurate location lookup would be expected to take longer than an approximate lookup. A server-side error response may be returned quickly but does not fulfil the API call (from the API Consumer's perspective)

5.8.6 Total API Latency

This is the delay experienced by the API Initiator between submitting an API Request and receiving an API response. Where the API Initiator is an Application Provider (application developer), the Total API Latency will be the sum of the API request time northbound from the Application to the Channel Platform and to the OGW Platform, plus the response time back to the Application.

The following are factors affecting Total API latency:

- The network speed
- The resource server load
- The resource server geographical location
- The performance of the load balancer
- The size of the data being sent or requested
- The API design

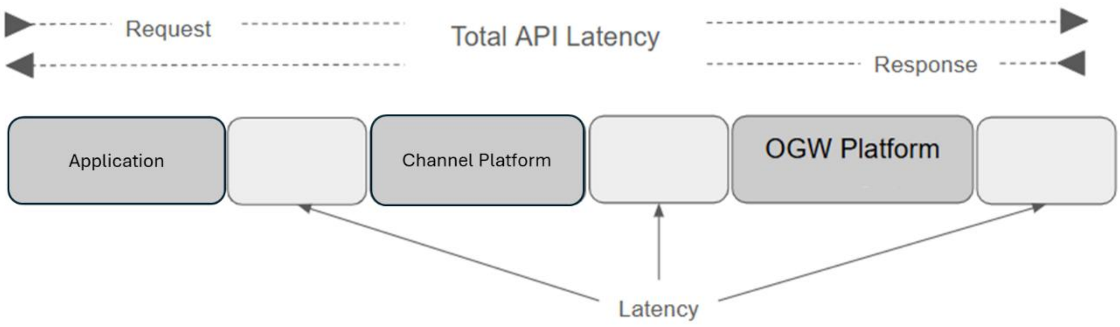


Figure 41 Total API Latency – Aggregation Model

To cover the segment including the Application and the internet segments, the API Initiator, Channel Platform or Leading OGW Platform Owner may provide external monitoring of metrics to calculate the Total API Latency. These would be achieved through Synthetic Monitoring, via testing applications that are deployed and hosted by the Channel Platform or OGW Platform Owner.

5.8.7 Availability

Measured by: Leading Operators

Reporting period time the API is available divided by total time of reporting period, then multiplied by 100.

5.8.8 Success Rate

Measured by: API Initiators / Aggregators / Leading Operators

The number of successful API requests for the reporting period divided by the total number of attempts for the reporting period, then multiplied by 100.

Synthetic monitoring at an API endpoint may be used to measure success rate.

Successful API requests are HTTP 20x response codes containing the desired payload (where ‘desired payload’ means the API publisher has fulfilled a valid request, to an available service, according to the terms of its SLA), or ‘friendly’ 4xx status codes (where ‘friendly’ means the response status and/or response message informs the API Consumer what to do next to retry the request), or a 501 response that means that the API endpoint is not implemented.

Note that the API Consumer may consider the accuracy of the response when determining ‘success’, however that is a qualitative metric rather than performance-based and depends on the SLA between the API Consumer and the OGW Platform Owner.

An API Initiator, Aggregator, or OGW Platform Owner may use Synthetic Monitoring at an API Endpoint to measure success rate.

The API Initiator, Aggregator and OGW Platform Owner may conduct their own log analysis and compare success rates.

5.8.9 Percentiles

To analyse the data captured in ‘API Response Time’ and ‘Total API Latency’ we can use percentiles.

The data set is sorted in descending order and split into % points. Commonly used percentiles are P50, P95, P99.

- P50, or the Median, is the value below which 50% of the data falls. This is the typical performance of your API.
- P95 is more useful for server applications where the data is more consistent.
- P99, also known as Peak Response Time, is also useful for server applications. It marks the upper limit of the performance.

5.8.10 Considerations for Performance Management

5.8.10.1 Caching

Certain APIs may have caching implemented meaning that performance measurements may be distorted.

5.8.10.2 Leveraging External Systems

Certain APIs will reuse existing internally facing infrastructure meaning that we expect the performance to be slower than, for example, native cloud applications.

5.8.10.3 Service Distribution

When Leading OGW Platform is different to OGW Platform of the Subscriber's Operator there is an additional ‘hop’ that will add latency to the Total API Latency.

5.8.10.4 Real time Performance Monitoring

5.8.10.4.1 Synthetic Monitoring

Synthetic monitoring is relevant for the following metrics:

1. Total API Latency
2. Success Rate

To cover the production segments including the application and the internet segments, the API Initiator, Aggregator or OGW Platform Owner may conduct Synthetic Monitoring to calculate the metrics of Total API Latency and Availability of production systems. These would be achieved through external testing applications that are deployed and hosted by the API Initiator, Aggregator or OGW Platform Owner. Data logged during Synthetic Monitoring should be shared freely and periodically between parties, and/or during root-cause analysis or a specific issue.

5.8.10.5 Service Status Web Page

The OGW Platform Owner may provide service status page to increase service awareness to cover the metrics listed in clause 5.8.4.

5.8.10.6 Threshold Configuration

The OGW Platform Owner may provide functions to set performance thresholds for critical metrics to monitor service performance. These metrics can be varied per API.

The OGW Platform may trigger alerts when these thresholds are breached.

5.8.10.7 Historical Data and Reporting

The OGW Platform may store historical performance data and allow to generate reports for analysis, whilst adhering to local data protection and privacy regulations.

5.9 Cross Operator alignment

5.9.1 QoS related alignment

QoS Performance Profiles are sets of parameters defining packet delivery performance, including throughput, latency, and related characteristics. They are designed to enable Application Providers to innovate on top of a predictable network baseline.

Note: In CAMARA these Performance Profiles are name QoS Profiles and can be discovered with the qos-profiles API.

To ensure that Application Providers can achieve similar experience across Operators both the set of parameters and an initial set of Performance Profiles should become widely available across markets.

Importantly, these aligned profiles are defined in terms of the packet delivery performance they provide rather than the use case they are intended for, to make them usable for any Application that can benefit from the specified performance. While minimum targets are defined for interoperability, Operators retain the autonomy to differentiate their services through better performance or confidence levels beyond the agreed baseline.

The proposed Performance Profiles in the table below are defining specific characteristics, these profiles are requested from the network. Performance Profiles for best-effort access are considered to not need alignment at this point as the Applications can and will adapt to the actual performance provided (this may however be of interest later). As a complement to the industry-aligned Performance Levels, there will be offerings based on Operator-unique Performance Profiles, particularly to serve use cases where customization can be more essential than global application scale.

Example Services	Example Services	Downlink performance parameters		Uplink performance parameters		Session characteristics	
		Throughput [Mbps]	DelayLimit [ms]	Throughput [Mbps]	DelayLimit [ms]	Downlink	Uplink
BIDIRECTIONAL0.1	OTT voice	0.1	60	0.1	60	Streaming	Streaming
BIDIRECTIONAL 4	video conferencing	4	60	4	60	Streaming	Streaming
BIDIRECTIONAL8	HD video conferencing	8	60	6	60	Streaming	Streaming
BIDIRECTIONAL12	Premium video conferencing	12	60	6	60	Streaming	Streaming
BIDIRECTIONAL30	Premium HD video conferencing	30	60	1.5	60	Streaming	Streaming
DOWNSTREAM8	Cloud gaming	8	60	1	60	Streaming	Intermittent
DOWNSTREAM16	Premium Cloud gaming VR/XR	16	60	1	60	Streaming	Intermittent
UPSTREAM0.5	Body camera	1	60	0,5	60	Intermittent	Streaming
UPSTREAM1	Remote video control light AI upstream light	1	60	1	60	Intermittent	Streaming
UPSTREAM02.5	Remote video control medium AI upstream medium	1	60	2,5	60	Intermittent	Streaming
UPSTREAM5	Remote video control AI upstream	1	60	5	60	Intermittent	Streaming
UPSTREAM10	Remote video control HD AI upstream HD	1	60	10	60	Intermittent	Streaming
INTERACTIVE0.1	Telemetry HD Voice	0.1	100	0.1	100	Intermittent	Intermittent
INTERACTIVE1	Point-of-sales terminals Drone delivery	1	100	1	100	Intermittent	Intermittent

Table 4 Performance Profiles

The characteristics described in the table above can be mapped to the CAMARA QoS Profile API, the characteristics map as follows:

Table reference	CAMARA QoS Profile parameter
Descriptive naming	QoSProfileName
Downlink performance parameter Throughput [Mbps]	targetMinDownstreamRate
Downlink performance parameter delayLimit [ms]	packetDelayBudget
Uplink performance parameter Throughput [Mbps]	targetMinUpstreamRate
Uplink performance parameter delayLimit [ms]	packetDelayBudget

Table 5 Mapping to CAMARA QoS Profile API

CAMARA specifies many more optional parameters in the QoS Profile. These parameters remain Operator specific and can be used to inform the Application Provider about additional capabilities and information around ordering the QoS Profile.

6 Reporting

OGW Platform shall support collecting, processing and reporting information about the Service API invocations as well as about the supporting operations enabling those invocations.

The collected information (either in raw format or aggregated) may be used to generate insights in several scenarios:

- Rating, billing and invoicing processes,
- (Re)-conciliation processes among several parties,
- Analytics-driven decisions,
- Root-cause analysis,
- Auditing processes (e.g., during privacy control checks, etc.)

It is expected that the OGW Platform supports gathering the relevant information across all the interfaces depicted in Figure 1 since the above scenarios may involve interactions among several parties, e.g,:

- Operator and Channel Platform (NBI)
- Operator A and Operator B (EWBI)

Additionally, the collected information may need to be aggregated at several granularities, namely (non-exhaustive list):

- Time (e.g. daily or hourly)
- Application Provider
- Application
- Channel Platform (if applicable)

- Product ID / Product Order ID
- Partner OGW Platform (if applicable)
- API name
- API version
- Coarse geographic characteristics

To enable such a collection (and aggregation) of information, the OGW Platform provides the API Management Functions (clause 2.2.1.1) and API Gateway Functions (clause 2.2.1.2) to support:

- API Monitoring
- API Usage Data Generation
- API Logging and Tracing
- API Metrics Generation
- API Audit Logging

In order to protect the privacy of the End-Users, the reporting information to be exchanged by the OGW Platform with other parties must not include any Personally Identifiable Information (PII) data.

A recommended set of reporting attributes has been identified based on [20], [5], [21], [22], [19] and clause 2.2.1.4.1 and is presented in clause 6.1 for usage reporting, clause 6.2 for administrative reporting and clause 6.3 for Operator-internal handling.

6.1 Usage reporting attributes

As stated in [19], a Service API may be considered a product from TM Forum perspective.

Note: TM Forum is defining the specification for TMF 937 Operate API – Usage for usage reports exchange between OGW Platform and Channel Platform. Therefore, alignment of terms and definitions may be needed in the future.

In accordance with [23] (but also [25], [24]), the management of usage information (i.e., creation, update, retrieval, import and export) builds on:

- product usage specification(s): the set of usage characteristics that are of interest to the business (and may have associated charges). There may be one or more usage specifications per Service API.
- product usage(s): a record that registers the usage of a product (i.e. the Service API consumption by an application). This record shall include a reference to the are established through a contract between these parties.

The consumption of API products may be based on request/response or subscribe/notify patterns.

- For request/response patterns, consumption may be influenced by the values of certain parameters in the API request (e.g. bandwidth, duration, location accuracy, information age, …),
- For subscribe/notify pattern, it may be possible to additionally set thresholds targeting specific attributes in Table 6.

Category	Attribute	Possible Consumer
Data traffic usage in the Operator's Network [21]	Data Volume imposed to the network due to the Service API consumption	Application Provider, Aggregator, Partner OGW Platform(s)
	Duration of the session requested through the Service API	Application Provider, Aggregator, Partner OGW Platform(s)
Service API invocation (both exchange patterns) [21]	Identity of the Service API Consumer e.g., Application Provider ID and Application ID	Application Provider, Aggregator, Partner OGW Platform(s)
	API name	Application Provider, Aggregator, Partner OGW Platform(s)
	API version	Application Provider, Aggregator, Partner OGW Platform(s)
	Targeted resource	Application Provider, Aggregator, Partner OGW Platform(s)
	Full URI of the targeted API resource (see Note 2)	Application Provider, Aggregator, Partner OGW Platform(s)
	Protocol invoked (e.g., HTTP 1.1., HTTP 2, MQTT, WEBSOCKET)	Application Provider, Aggregator, Partner OGW Platform(s)
	HTTP verb	Application Provider, Aggregator, Partner OGW Platform(s)
	HTTP status code	Application Provider, Aggregator, Partner OGW Platform(s)
	List of input parameters. May include (when applicable): purpose, scope, … (see Note 2)	Application Provider, Aggregator, Partner OGW Platform(s)
	List of output parameters (see Note 1 and Note 2)	Application Provider, Aggregator, Partner OGW Platform(s)

Table 6: Usage reporting attributes

Note 1: Some parameters may signal a differentiated treatment/outcome. If so, they should be collected/logged consistently. For instance, consider the case of regression error in a forecast (for the ConnectivityInsights APIs) or location accuracy (for the DeviceLocation APIs).

Note 2: The information exchange among parties is possible as long as the respective privacy controls are in place and enforced by the involved parties, so no PII is shared.

6.2 Administrative reporting attributes

Table 7 presents a list of administrative reporting attributes which may be shared with other parties but are not necessarily billable.

Category	Attribute	Possible Consumer
Operational Aspects	User identifier to Operator identifier Resolution (Telco Finder API)	Partner OGW Platform(s)
	Portability Search Mode (Telco Finder API)	Partner OGW Platform(s)
	Getting Routing Rules (Telco Routing API)	Partner OGW Platform(s)
	Resolution under portability scenarios (Network Id API)	Partner OGW Platform(s)
	API Product Order State Change Event (see Note 3)	Application Provider, Aggregator
	Application Approval Status Change Event (see Note 3)	Application Provider, Aggregator
	Application Owner Approval Status Change Event (see Note 3)	Application Provider, Aggregator
	Access Control Policies Change Event	Application Provider, Aggregator, Partner OGW Platform
	Service API Availability Change Event	Application Provider, Aggregator, Partner OGW Platform
Service API invocation (both exchange patterns)	Time elapsed between API request and response	Application Provider, Aggregator, Partner OGW Platform(s)
	List of all forwarding entities between Service API Consumer and API producer (see [26])	Application Provider, Aggregator, Partner OGW Platform(s)
Event Notification	Event Notifications related to the successful/failed of Service APIs consumption	Application Provider, Aggregator, Partner OGW Platform(s)
Operational Event Notifications	API Product Offering Create Event	Application Provider, Aggregator, Partner OGW Platform(s)
	API Product Offering Attribute Value Change Event	Application Provider, Aggregator, Partner OGW Platform(s)

Table 7: Administrative reporting attributes

Note 3: This attribute may be alternatively categorized as “Service activation”-related [21]: as part of the Application Provider onboarding process [19], it is possible to generate (one time/recurring) service activation fees.

6.3 Operator-internal reporting attributes

Not all the attributes are meant to be shared with other parties due (for instance) to:

- privacy constraints, or

- they mainly provide insights for optimizing internal operational aspects, or for internal root-cause-analysis purposes.

Table 8 provides a set of attributes which are categorized as non-shareable.

Category	Attribute
Service API invocation	Source [ipv4Addr:port, ipv6Addr:port, fqdn…]
	Destination [ipv4Addr:port, ipv6Addr:port, fqdn…]
Operational Aspects	Application Provider Onboarding
	Application Provider Inquiry/Modification
	Application Provider Deactivation
	Application Provider Approval Status
	Application Provider Approval Delay
	Application Onboarding
	Application Inquiry/Modification
	Application Deactivation
	Application Approval Status
	Application Approval Delay
	API Product Offering Creation
	API Product Offering Inquiry
	API Product Order Creation
	API Product Order Inquiry/Modification
	API Access Removal
	User identifier to Operator identifier Resolution Delay (Telco Finder API)
	Getting Routing Rules Delay (Telco Routing API)
	Resolution delay under portability scenarios (Network Id API)
Operational Event Notifications	API Product Order Create Event
	API Product Order Attribute Value Change Event
	Application Create Event
	Application Attribute Value Change Event
	Application Owner Create Event
	Application Owner Attribute Value Change Event

Table 8: Operator-internal attributes

7 MVNO implementation

Mobile Virtual Network Operators (MVNOs) function by leasing network capacity from traditional Mobile Network Operators (MNOs) instead of owning their own infrastructure. MVNOs can employ different business models, each with specific characteristics and operational scopes. There are several challenges in the open gateway ecosystem for MVNOs, depending on their operational model.

7.1 Types of MVNOs

The various business models organize MVNOs into specific categories. These categories determine the technical architecture for their deployment.

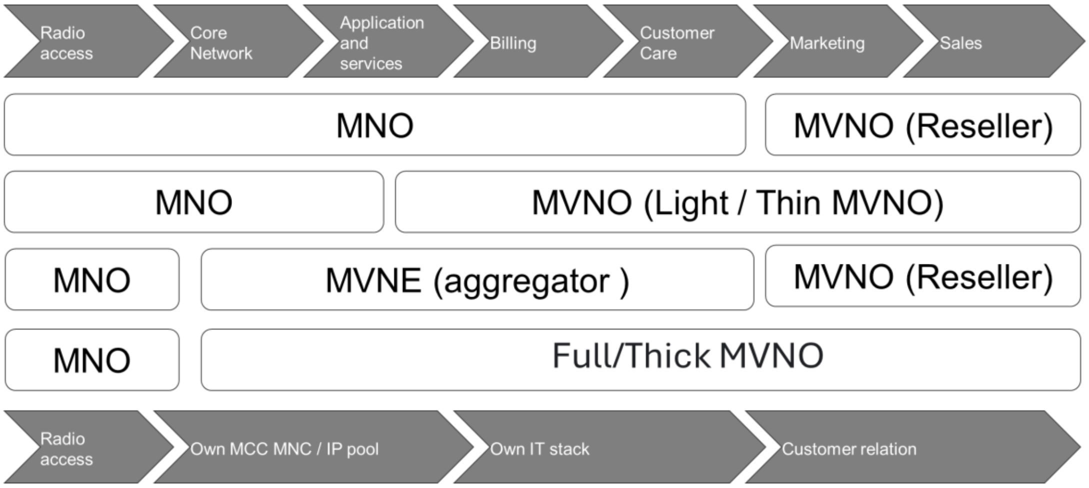


Figure 42 MVNO Categories

- Full MVNO

  A Full MVNO (Mobile Virtual Network Operator) operates almost like a traditional MNO (Mobile Network Operator) but without owning the radio spectrum or RAN infrastructure. They own BSS stack and most core network infrastructure elements, including the Home Location Register (HLR) / Home Subscriber Server (HSS), the Authentication Centre (AuC) / AUthentication Server Function (AUSF), and other systems to manage Subscriber information. This grants them greater control over their service, pricing, and customer experience.

- Thin MVNO

  A Thin MVNO, sometimes referred to as a Light MVNO, relies more heavily on the infrastructure and systems of the MNO. They do not own as many network elements (or network functions) as a Full MVNO and usually depend on their partner MNO for much of their service delivery and operational needs. They usually own the BSS stack. This model typically involves less investment and lower operational complexity.

- Reseller

  A Reseller or MVNO does not own any network infrastructure; instead, it simply resells the mobile services provided by an MNO. This type of MVNO focuses on marketing, sales, and customer service, offering branded mobile services without the need for technical investments in network assets.

- MVNE

An Aggregator MVNO serves as an intermediary between one or more MNOs and other MVNOs, providing wholesale network access and possibly additional services such as billing and customer support.

- Multi tenanted MVO
  A multi-tenanted MVNO (Mobile Virtual Network Operator) is an MVNO that leverages wholesale network access from multiple Mobile Network Operators (MNOs), often across different geographic regions or for specific service quality/cost advantages.

7.2 Open gateway MVNO deployment models

The various models offer different levels of control over the infrastructure, services offered, and the data required to fulfil the requirements set by the different APIs.

7.2.1 Resellers

Resellers typically do not own their own infrastructure and rely on the MNO's core network and BSS stack to serve their customers. It is recommended that they utilize the MNO's OGW Platform for service delivery.

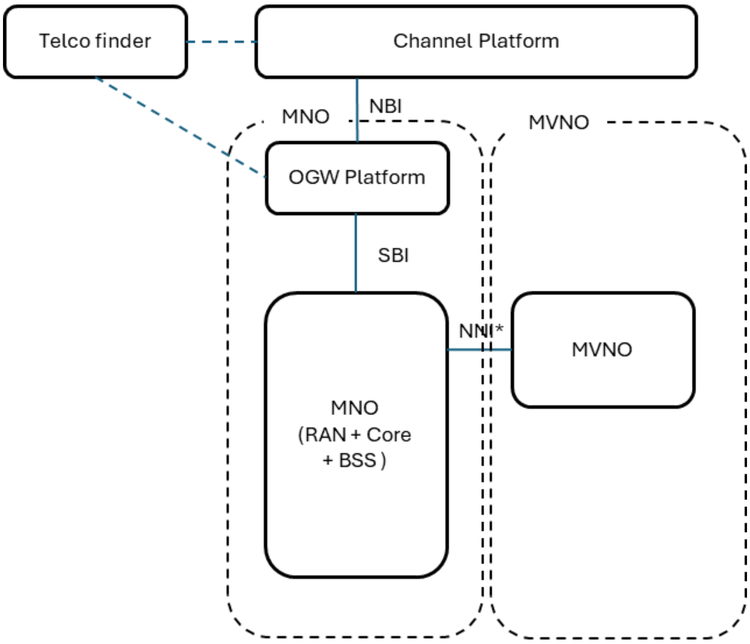


Figure 43 Reseller deployment

It is recommended that for any data transfer requirements from the MVNO to the MNO to fulfil API requirements utilize the existing Network-to-Network or BSS-to-BSS interface (NNI*).

The Telco Finder utilizes IP addresses and MSISDNs to identify the OGW Platform. This process is relatively straightforward in this case as the MVNO employs the MNO's IMSI and IP address ranges. For the detailed process please refer to section 2.2.1.4.1.

Resellers using the services of an aggregator would rely on the MVNEs OGW Platform.

7.2.2 Thin MVNOs

As Thin MVNOs own some infrastructure their capability is likely greater to host their own OGW Platform however they can utilize the MNOs OGW Platform as well. Using their own OGW Platform would reduce the dependency on the MNO for some services.

7.2.2.1 Thin MVNO deployment with OGW Platform in MNO

For smaller Thin MVNOs, it may be more practical to use the MNOs OGW Platform as it could be more cost-effective. The deployment model would be identical to the option described in the section 7.2.1.

Note: The need to transfer data from the MNO to the MVNO to fulfil the services required by some APIs is greater than in the reseller case.

To meet API requirements, it is recommended that any necessary data transfers from the MVNO to the MNO utilize the existing Network-to-Network or BSS-to-BSS interface (NNI*).

The process to locate the correct OGW Platform is identical to the resellers case described in section 7.2.1.

7.2.3 Thin MVNO deployment with OGW Platform in MVNO

For larger MVNOs it could be more practical to deploy their own OGW Platform. In this case dependent of service offerings information may need to be transferred from the MNO to the MVNO.

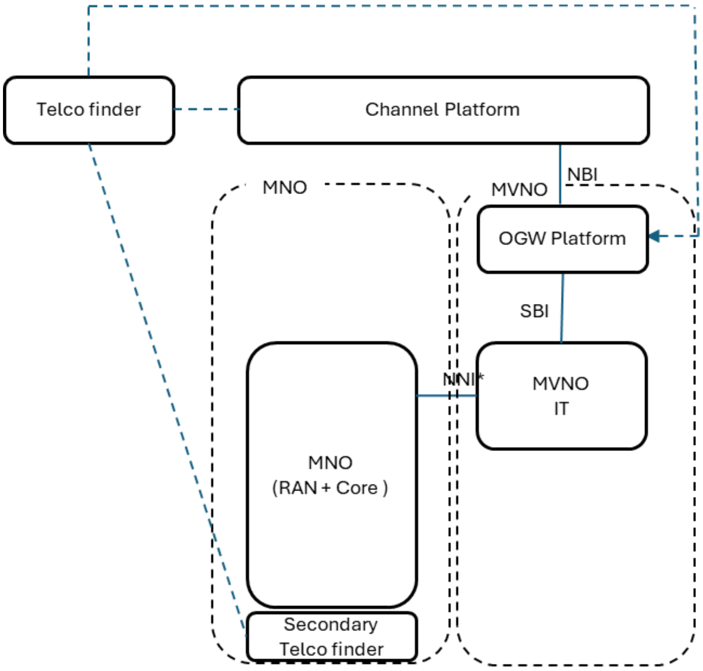


Figure 44 OGW Platform in Thin MVNO

It is recommended that for any data transfer requirements from the MVNO to the MNO to fulfil API requirements the existing Network-to-Network / BSS-to-BSS interface used (NNI*).

In this scenario as the MVNO using IMSI and IP address ranges from the MNO it is necessary to utilize the secondary Telco Finder function described in clause 2.2.1.4.1.3.

7.2.4 Full MVNOs and MVNEs

Since Full MVNOs typically own the entire core network and BSS stack, they are well-positioned to deploy their own OGW Platform. However, they also have the option to use the MNO-hosted OGW Platform. Deploying their own OGW Platform minimizes reliance on the MNO for most services.

7.2.4.1 Full MVNO deployment with OGW Platform in MNO

Similar to the Thin MVNO scenario for smaller Full MVNOs, leveraging the MNO's OGW Platform may be a more practical and cost-effective solution. The deployment approach would align with the option outlined in Section 7.2.1.

Note: The need to transfer data from the MNO to the MVNO to support certain API-driven services will increase significantly.

To meet API requirements, it is recommended that any necessary data transfers from the MVNO to the MNO utilize the existing Network-to-Network or BSS-to-BSS interface (NNI*).

The method for identifying the appropriate OGW Platform follows the approach used for resellers, as detailed in Section 7.2.1. In this case, the MVNO's IMSI and IP address ranges should be directed to the MNO's OGW.

7.2.4.2 Full MVNO deployment with OGW Platform in MVNO

Deploying the OGW Platform in their own infrastructure provides clear advantages to a larger MVNO by reducing the reliance on the MNO for API services and provides more flexibility in the future.

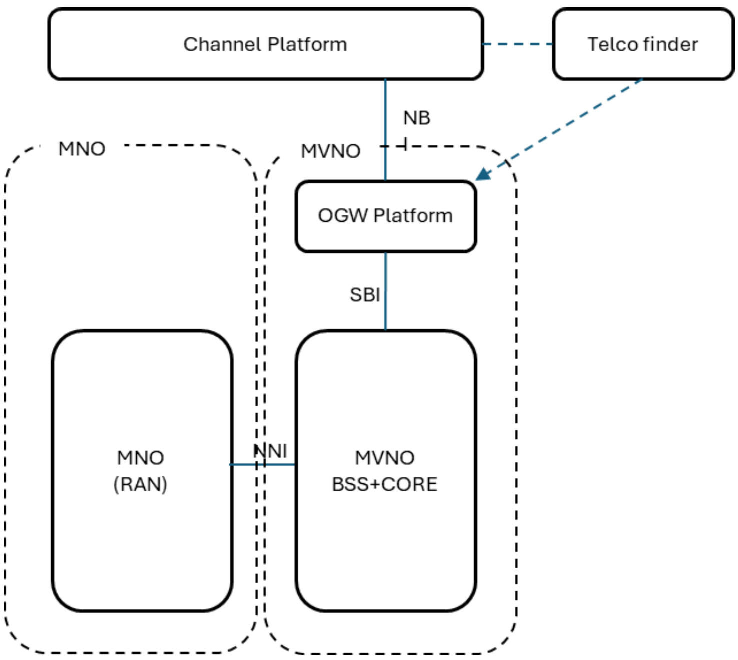


Figure 45 OGW Platform in Full MVNO

The method for identifying the appropriate OGW Platform follows the standard approach as the MVNO in this instance using their OWN IMSI and IP ranges.

To meet API requirements, it is recommended that any necessary data transfers from the MVNO to the MNO utilize the existing Network-to-Network or BSS-to-BSS interface (NNI*).

7.2.5 Multi tenanted MVNOs

7.2.5.1 Resellers

In the multi tenanted reseller model the recommended deployment model is the same as described in section 7.2.1.

For services requiring reserved MNO resources (e.g., QoS), dynamic MNO switching for a Subscriber is not recommended. Differentiated services will cease if the Subscriber moves to another MNO.

7.2.5.2 Thin MVNO

The recommended Thin MVNO deployment model closely follows the standard Thin VMNO model where the MVNO deploys their own MVNO platform. As the MVNO would utilize a multi IMSI model to switch between MNOs the Telco Finder would retrieve the routing information from the MNOs secondary Telco Finder to route the request to the MNO.

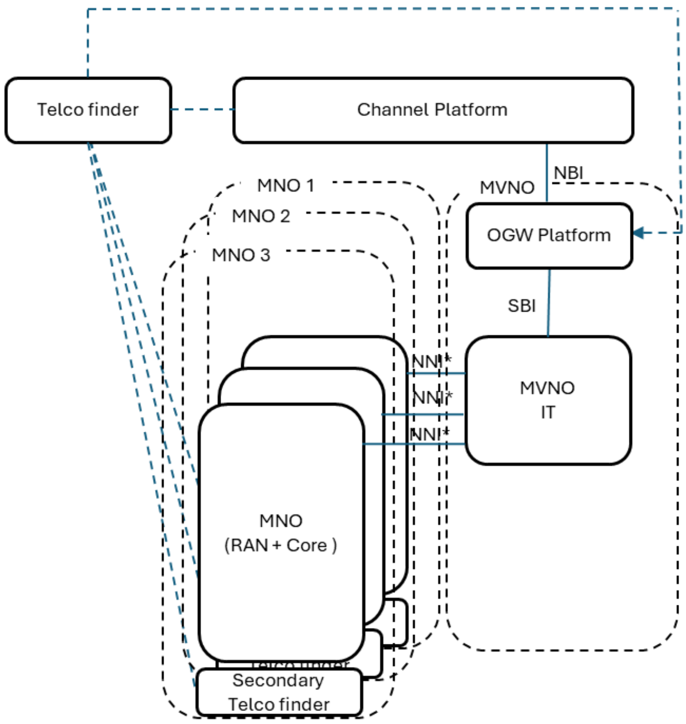


Figure 46 Multi tenanted Thin MVNO

For services requiring reserved MNO resources (e.g., QoS), dynamic MNO switching for a Subscriber requires the MVNO to request services again from the new MNO dependent on SLAs this may cause a break in the differentiated services.

Note: The scenario where OGW Platform is not deployed by the MVNO is for further study

7.2.5.3 Thick MVNO

The recommended deployment option closely follows the Thick MVNO model.

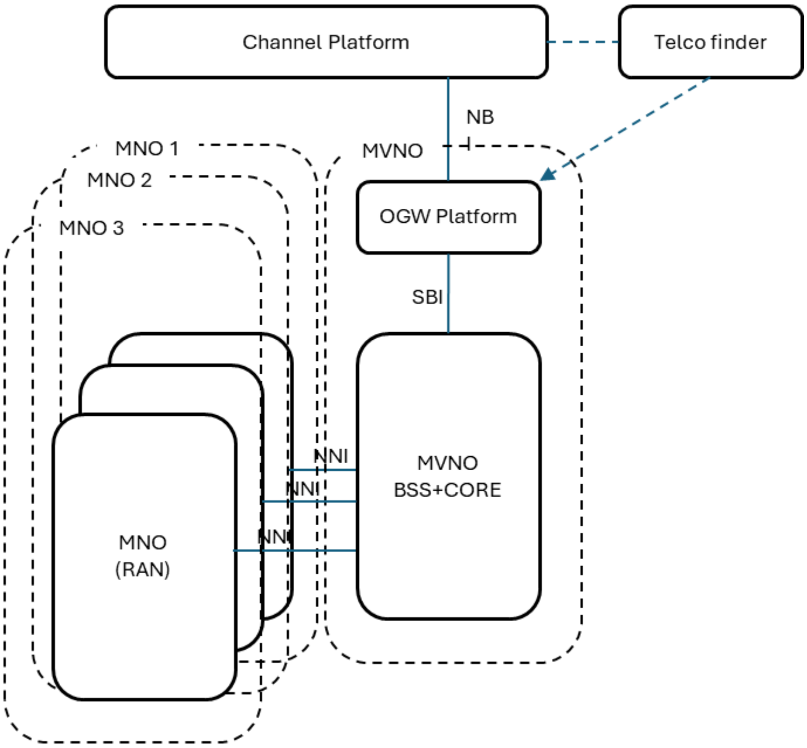


Figure 47 Multi tenanted Thick MVNO

For services requiring reserved MNO resources (e.g., QoS), dynamic MNO switching for a Subscriber requires the MVNO to request services again from the new MNO dependent on SLAs this may cause a break in the differentiated services.

Key Considerations for Multi-Tenanted MVNOs:

- **Dynamic Subscriber switching:** When the Subscriber can move freely between MNOs it is crucial to have the relevant agreements in place with all MNOs to provide a consistent user experience.
- **Billing and Charging:** Ensuring accurate and fair billing across multiple MNOs for API consumption, especially when dynamically switching networks, is a complex but vital aspect.
- **Data Consistency and Sync:** Maintaining a consistent view of Subscriber data and Consent across potentially disparate MNO systems via APIs is challenging. The recommendation is that for Multi tenanted MVNOs the MVNO holds the Consent for each of its Subscribers.
- **QoS Alignment:** If an MVNO promises certain QoS levels, its Open Gateway Platform implementation needs to ensure that the underlying MNOs can deliver on that intent, potentially using QoS APIs.

7.3 Summary

MVNOs have the flexibility to deploy the OGW Platform with varying levels of reliance on the MNO, depending on their network architecture and operational strategy. A Full MVNO with its own core network and BSS stack can implement an independent OGW Platform, minimizing dependency on the MNO for most services. Conversely, a Thin MVNO, which relies heavily on the MNO's infrastructure, may opt to use the MNO-hosted OGW Platform for cost efficiency and streamlined integration. The degree of reliance is influenced by factors such as operational control, cost considerations, regulatory requirements, and the level of customization needed for service offerings.

7.4 Limitations

The following topics require further study:

- Privacy management implications in certain scenarios
- Integration of MVNOs using the EWBI interface

8 Minimum Viable Product

To ensure interoperability between various implementations of the OGW Platform the following section outlines the critical components and functional scope recommended for the deployment. This includes guidelines for API management, security protocols, and the effective operation and discovery of APIs.

8.1 API Functional Scope (CAMARA Conformance)

Adhere to a clear API versioning and deprecation policy. CAMARA meta-releases are the target, and all API implementations shall be uplifted within a 12-month window from the corresponding CAMARA meta-release. Older API versions, if supported, should be maintained as long as they are considered active in CAMARA. Implementations should be prepared for quicker deprecation cycles due to security or privacy considerations.

8.2 Operate APIs

Operators and Aggregators implementing Open Gateway APIs should ensure they are kept up to date with the latest TM Forum Operate APIs releases. Specifically, all deployments should be updated within 12 months of a new major release from the standards body.

8.3 Authorization

A robust security framework based on 3-Legged access tokens (as defined in [8]) should be implemented. This protects user data, facilitates proper Consent management, and safeguards Operator networks. Security must be a primary concern, especially when accessing sensitive information.

8.4 Privacy and Consent

API implementations should comply with all applicable local, legal, and regulatory requirements concerning data privacy and protection.

A robust mechanism should be implemented to obtain explicit and informed user Consent for all data access and usage if mandated by local regulation and when needed. This Consent mechanism should be transparent and easy to understand.

9 Roaming

Roaming introduces unique challenges for implementing CAMARA APIs across multi-Operator environments, where maintaining service continuity, enforcing consistent policies, and ensuring accurate data exposure require enhanced coordination between home and visited networks. When considering the implementation of these APIs in roaming scenarios, two main challenges must be addressed. The first is technical, involving the need to guarantee that the API operates seamlessly from the home network to the visited network and across any intermediary transit Operators. The second challenge pertains to navigating a complex set of legal and regulatory frameworks across various jurisdictions; as user data and network activities cross international borders, it is vital to clarify issues concerning data privacy, Consent, cross-border access, and inter-Operator agreements.

This section outlines:

- The implications of roaming on API behaviour and availability
- API categorization
- Federation requirements

The high-level roaming scenario for the Open Gateway Platform depicted in Figure 48.

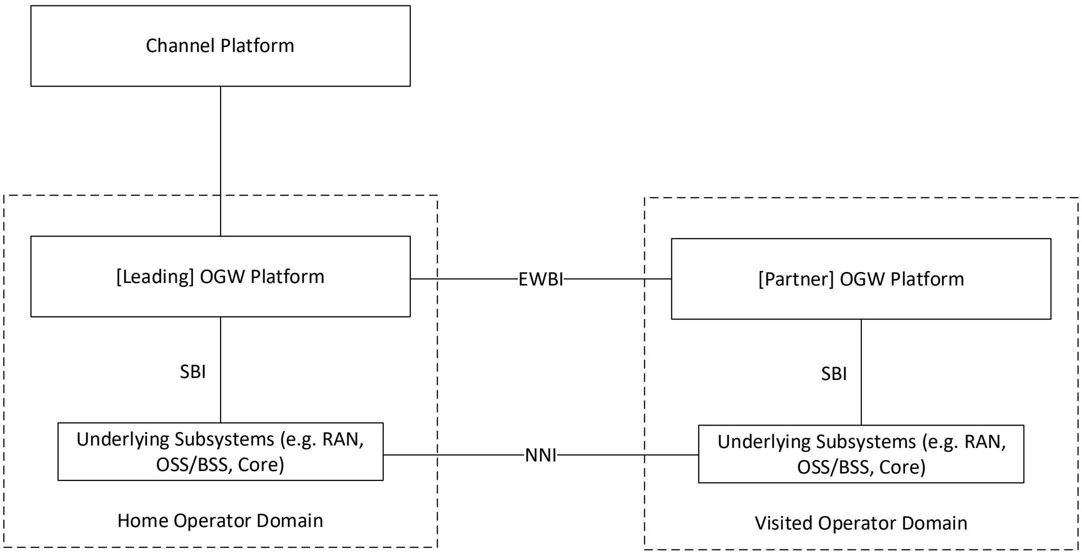


Figure 48 High level roaming scenario

9.1 Implications on API behaviour and availability

When a Subscriber roams outside their home network, the behaviour of CAMARA APIs—and the data they can expose—can change significantly due to network boundaries, regulatory constraints, and technical dependencies. The effectiveness and accuracy of API responses may vary depending on whether the required data resides in the home network, the visited network, or must be coordinated across both.

To manage API availability across complex cross-border regulations, it is recommended that Operators (both home and visited networks) implement a jurisdictional whitelist/blacklist system. This framework helps enforce compliance by pre-approving permitted APIs (whitelisting) and blocking known non-compliant ones (blacklisting) on a per-country basis.

While this system provides a technical control, each Operator must still make its own final legal judgment on whether to offer an API in any specific roaming context, as the ultimate responsibility for compliance rests with them.

To manage API availability upon application requirements, it is recommended that Operators implement a mechanism to provision application requirements with respect roaming conditions, e.g., white lists or black lists whether an application is willing to be served or not served, and if appropriate, duly charged, when the UE is roaming

9.2 API categorization

The APIs can be categorized based on several factors These factors can highly influence the technical and regulatory complexity of the API implementation.

Data Locality: Describes where the necessary data for the API resides—either in the home network, the visited network, or both.

- In roaming, data such as real-time location, session telemetry, and RAN-related metrics typically reside in the visited network. Depending on the requested accuracy (e.g. “PLMN” level) the metrics can be served from the home network.
- Home network APIs may not have direct access to this data unless federation or data synchronization mechanisms are standardized and implemented.

Execution Locality: Indicates where the API logic or service must be executed to ensure proper functioning—home, visited, or edge environment.

- Edge services (e.g., MEC APIs) and QoS enforcement need to execute close to the user to meet latency, bandwidth, or control requirements.
- APIs that are "location-aware" (e.g., geofencing) require network-local execution for timely and accurate event processing.

Latency Sensitivity: Reflects whether the API requires low-latency response times (e.g., real-time control or user-facing experiences).

- APIs like Quality on Demand or Edge Discovery are often tied to End-User experiences (e.g., gaming, streaming, or XR) and are highly delay sensitive.
- Latency introduced by home-visited network hops may break QoE expectations.

Privacy & Regulatory Sensitivity: Determines if the API handles personally identifiable information (PII), location data, or other regulated attributes that require Consent and legal safeguards.

- Location APIs and identity verification functions may invoke GDPR, ePrivacy, or other local data protection frameworks.
- Consent mechanisms, audit trails, and cross-border data transfer governance become mandatory.

Federation Dependency: Highlights whether API functionality depends on federation between home and visited Operators to access real-time data or network functions.

- APIs such as Location Retrieval or QoS Management cannot deliver valid results without visited network collaboration.

- This requires OGW Platforms to federate API exposure and trust models (e.g., via EWBI).
- The dependency relies on the underlying technology and the established roaming agreements between the home and visited networks on a per-API basis.

9.3 Federation requirements

In any scenario where the home Operator cannot fulfil the API requirement due to either the data locality or the execution locality in the visited network the home Operator must request the data or resource allocation from the visited network. Most of the requests can be managed using existing Network-to-Network Interfaces (NNI) depending on roaming agreements and the technology involved

Figure 49 provides an example model for home network Operators to handle incoming CAMARA API calls for their roaming Subscribers.

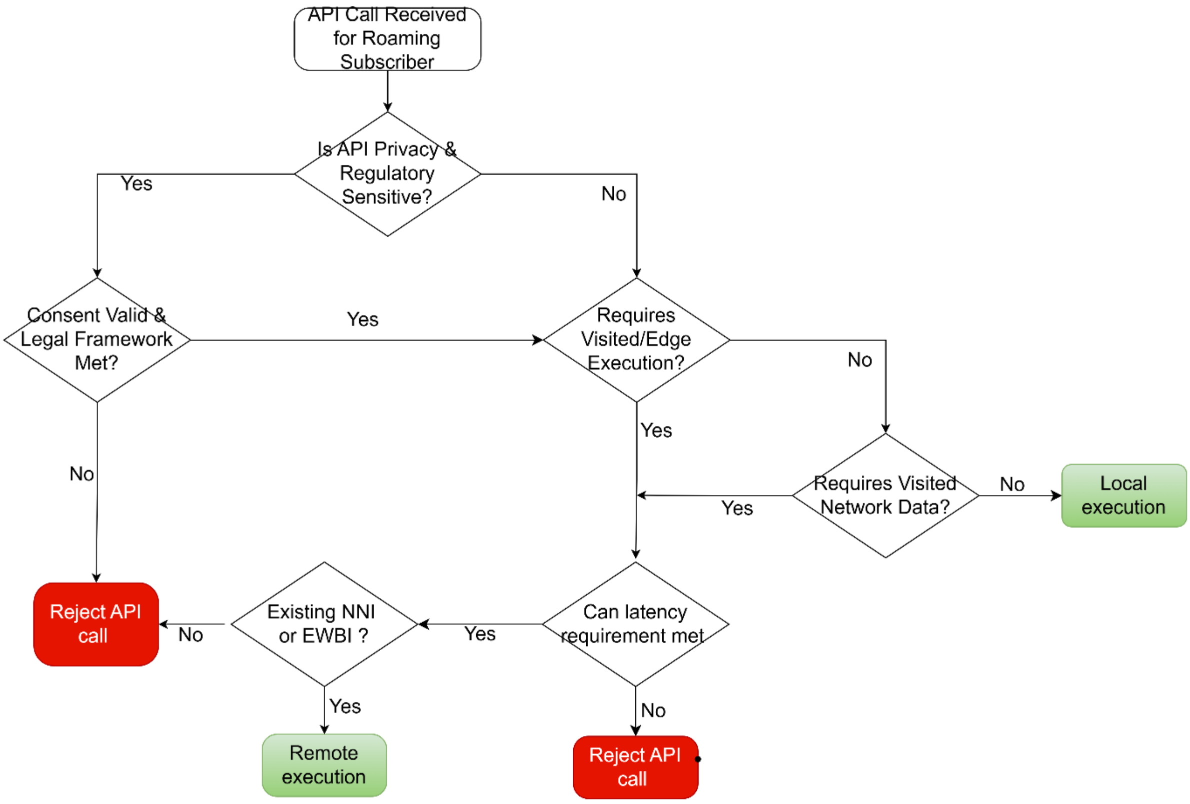


Figure 49 Example model for handling CAMARA API calls in roaming scenarios

10 Vetting Process

Vetting Process is defined as a business process (e.g. part of TM Forum Business Partner Agreement Management) whereby an Aggregator and / or Operator verify information provided by the Application Provider during Application Provider and Application onboarding as required by local Operator needs and local regulations, the so-called Vetting Information [1].

According to clause G.5.2 in [1], information in Table 9 has been identified as relevant to be verified:

Parameter	Description
Application Provider name	Part of the Application Provider Vetting Information indicating the commercial name of the Application Provider
Application Provider logo	Part of the Application Provider Vetting Information indicating the logo of the Application Provider
Application name	Part of the Application Vetting Information indicating the commercial name of the Application
Application logo	Part of the Application Vetting Information indicating the logo of the Application
Information for technical Vetting	Part of the Application Vetting Information indicating technical information associated to the Application, for instance redirect URLs format / ownership (applicable for authorization flows that imply HTTP redirects), etc.

Table 9: Recommended information to be vetted

An Operator obtains this information via the Application Provider and Application onboarding procedures. A suggested mapping of the parameters in Table 9 to the objects in TMF 931 [19] is presented in Table 10.

Information	Mapping to TMF 931 objects
Application Provider name	ApplicationOwnerOrganization_FVO => tradingName
Application Provider logo	ApplicationOwnerOrganization_FVO => organizationIdentification => attachment => content
Application name	Application_FVO => commercialName
Application logo	Application_FVO => logoUrl
Information for technical Vetting	Application_FVO => redirectUrl
	Application_FVO => jwksUri

Table 10: Mapping of the information to be vetted and TMF 931 objects

As stated in [1], the vetting services may be offered by an Operator itself, an Aggregator or a third party (vetting service(s)) provider. For the latter case, and depending on the local market and regulatory conditions and providers' availability, it is recommended to leverage the so-called verification authority service [35] and verification authority API [36].

The verification authority API in [36] is designed to provide independent, neutral, and secure processes for company/brand verification. This API defined in [36] is intended to give stakeholders at every level—from Operator, to Aggregators, to Application Providers—the ability to enhance the trust in their communications, while at the same time streamlining backend processes to enable industry-wide scale.

A suggested mapping of the parameters in Table 9 to the objects in verification authority API in [36] is presented in Table 11.

Information	Mapping to NG.131 objects
Application Provider name	Brand => BrandInfo => BrandName
Application Provider logo	Brand => BrandInfo => DefaultIcon
Application name	Chatbot => ChatbotInfo => ServiceName
Application logo	Chatbot => ChatbotInfo => ServiceIcon
Information for technical Vetting	To be defined
	To be defined

Table 11: Mapping of the information to be vetted and NG.131 objects

Note: Whether and how OGW Platform needs to provide the means to help the Vetting Service Provider to identify if an Application / Application Provider have already been vetted for another Operator is to be studied.

Note: Detailed flows on the enablement of the Vetting Process using TMF 931 and NG.131 are to be studied.

Annex A Telco Finder-related API specifications

A.1 Telco Finder API specification (OpenAPI Specification format)


```
openapi: 3.0.3

#############################################################################
#                              API Information                              #
#############################################################################

info:
  title: Telco Finder API
  version: '1.0.0-wip'
  description: |
    Telco Finder allows consumers to discover information about the operator to which a target
user belongs.

    Consumers invoke the `search` endpoint to discover the owning operator of a specific user.
Detailed information about API functionality and usage is contained below within the path
description.


  license:
    name: Apache 2.0
    url: https://www.apache.org/licenses/LICENSE-2.0.html

  termsOfService: "TBD"

  contact:
    name: Telco Finder Support
    url: https://tbc.com
    email: tbc@tbc.com

#############################################################################
#                            Server Definitions                             #
#############################################################################

servers:
  - url: "https://{baseUrl}:{port}/{domainContext}/{apiVersion}"
    description: Definition of the server URL

    variables:

      baseUrl:
        default: localhost
        description: Machine name or base URL

      port:
        enum:
          - '443'
        default: '443'
        description: Listening port of the service

      domainContext:
        default: telco-finder
        description: Domain context

      apiVersion:
        default: "v1"
        description: Major version of semantic versioning

#############################################################################
#                                   Tags                                    #
#############################################################################
```

```yaml
tags:
  - name: Telco Finder search
    description: Search API for resources

############################################################################
#                            Path Definitions                              #
############################################################################

paths:

  /search:
    post:
      summary: Create a request to search for the operator that owns a specific user
      tags:
        - Telco Finder search
      security:
        - openId:
          - telco-finder:search
      parameters:
          - $ref: '#/components/parameters/x-correlator'
      requestBody:
        required: true
        description:
          This operation retrieves information about the operator associated with a given
user.

          <br/><br/>**Request:**
          User information is conveyed within the JSON payload via the `target` object. This
object comprises of multiple optional fields to identify a target user (`phoneNumber`,
`ipv4Address`, `ipv6Address`). Consumers have the option to control the response verbosity
using the `includeApiRoot` and `includeAuthProviderConfiguration` boolean fields within the
request. These fields dictate whether the response includes the operator's API root URL and
authorisation server discovery endpoint data.


          In regions with Mobile Number Portability, consumers have the option to control a
phone number search mode by setting the `portabilitySearchMode` enum. This provides 2 options
- a Shallow search mode and a Deep search mode. The shallow option directs Telco Finder to
search only its internal records (e.g. cache). This method can be preferred to avoid higher
monetary costs associated with extended searches. The full search triggers a comprehensive
search against all external systems, providing more thorough results at a potentially higher
cost and ensuring up-to-date information by bypassing stale cached data.


          **Response:**
            The data returned by Telco Finder -

            * **Operator ID:** The operator to which the target user belongs. This field will
always be returned in the response.

            * **API Root of the Operator:** The root URL of the API Gateway managed by the
owning operator. This field is false by default but can be included in the response by setting
the request field `includeApiRoot` to true.

            * **Authorisation server discovery endpoint:** The discovery endpoint of the
operator's authorisation server. This is a standardised URL in [OpenID
Connect](https://openid.net/specs/openid-connect-discovery-1_0.html#ProviderMetadata) and
[OAuth 2.0](https://datatracker.ietf.org/doc/html/rfc8414#section-3) that allows clients to
dynamically retrieve configuration metadata about the authorisation server. This field is
false by default but can be included in the response by setting the request field
`includeAuthProviderConfiguration` to true.


            **Rationale for optional fields:** The `includeApiRoot` and
`includeAuthProviderConfiguration` request fields allow consumers to optimise the response
based on their specific needs. By default, only minimal information is returned (Operator ID)
```

```
to minimise computational costs. If a consumer is interested in further information, they can
set the aforementioned field values to `true`.

        content:
          application/json:
            schema:
              $ref: "#/components/schemas/TelcoFinderSearchRequestBody"
            examples:
              Phone number without filters:
                summary: Search using phone number
                description: A request that contains only mandatory fields (the `target`
object). Boolean filters are not specified in the request, so default values will be used
(false) for `includeApiRoot` and `includeAuthProviderConfiguration` - meaning minimal results
will be returned in response.
                value:
                  target:
                    phoneNumber: "+447709558432"
              IPv4 address with filters:
                summary: Search using IPv4 address with filters to control response
granularity
                description: A request that contains the optional boolean fields
`includeApiRoot` and `includeAuthProviderConfiguration` to control the response granularity.
Setting both filters with a value of true means that the response will contain 2 additional
result fields - `apiRoot` and `authProviderConfiguration`.
                value:
                  target:
                    ipv4Address:
                      publicAddress: "84.125.93.10"
                      publicPort: 59765
                  includeApiRoot: true
                  includeAuthProviderConfiguration: true
              Phone number with MNP Mode:
                summary: Specifying an MNP search mode
                description: A request for a phone number search in an MNP region, where the
search mode specified is a Shallow search
                value:
                  target:
                    phoneNumber: "+447709558432"
                  portabilitySearchMode: "SHALLOW"
              All request fields:
                summary: Specifying all fields
                description: A request containing all request fields (mandatory + optional).
                value:
                  target:
                    phoneNumber: "+447709558432"
                  includeApiRoot: true
                  includeAuthProviderConfiguration: true
                  portabilitySearchMode: "SHALLOW"

      responses:
        "200":
          description: OK
          headers:
            x-correlator:
              $ref: '#/components/headers/x-correlator'
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/TelcoFinderSearchResponseBody"

              examples:
                Default response - only operator ID returned:
                  summary: Default response - Only operator ID returned
                  description: Response where only operator ID is returned because the
consumer did not set the boolean filters to true
                  value:
                    operatorId: OPERATOR_ID
```

```
                All fields returned:
                  summary: Response with all fields returned
                  description: Response where all fields are returned because `apiRoot` and
`authProviderConfiguration` were both set to true in request payload
                  value:
                    operatorId: OPERATOR_ID
                    apiRoot: https://example.operator.com
                    authProviderConfiguration: https://auth.operator.com/.well-known/openid-
configuration

        "400":
          $ref: "#/components/responses/Generic400"

        "401":
          $ref: "#/components/responses/Generic401"

        "422":
          $ref: "#/components/responses/Generic422"

        "403":
          $ref: "#/components/responses/Generic403"

        "404":
          $ref: "#/components/responses/Generic404"

        "500":
          $ref: "#/components/responses/Generic500"

        "503":
          $ref: "#/components/responses/Generic503"

        default:
          description: Generic Error
          headers:
            x-correlator:
              $ref: '#/components/headers/x-correlator'

#############################################################################
#                          Component Definitions                            #
#############################################################################

components:

#---------------------------------------------------------------------------#
#                             Headers Definitions                           #
#---------------------------------------------------------------------------#
  headers:

    x-correlator:
      description: Correlation id for the different services
      schema:
        type: string

#---------------------------------------------------------------------------#
#                            Parameters Definitions                         #
#---------------------------------------------------------------------------#
  parameters:

    x-correlator:
      name: x-correlator
      in: header
      description: Correlation id for the different services
      schema:
        type: string

#---------------------------------------------------------------------------#
#                             Schema Definitions                            #
```

```
#---------------------------------------------------------------------------#
  schemas:

    ErrorInfo:
      type: object
      required:
        - status
        - code
        - message
      properties:
        status:
          type: integer
          description: HTTP status code returned along with this error response
        code:
          type: string
          description: Code given to this error
        message:
          type: string
          description: Detailed error description

    Target:
      description: |
        Represents the user that is the target of the search request. The API consumer can
choose to provide one of the the below specified target user identifiers:
            * `phoneNumber`
            * `ipv4Address`
            * `ipv6Address`
      type: object
      properties:
        phoneNumber:
          $ref: "#/components/schemas/PhoneNumber"
        ipv4Address:
          $ref: "#/components/schemas/DeviceIpv4Addr"
        ipv6Address:
          $ref: "#/components/schemas/DeviceIpv6Address"
      minProperties: 1
      maxProperties: 1

    PhoneNumber:
      description: A public identifier addressing a telephone subscription. In mobile networks
it corresponds to the MSISDN (Mobile Station International Subscriber Directory Number). In
order to be globally unique it has to be formatted in international format, according to E.164
standard, prefixed with '+'.
      type: string
      pattern: '^\+[1-9][0-9]{4,14}$'
      example: "+123456789"

    DeviceIpv4Addr:
      type: object
      description: |
        The user device should be identified by either the public (observed) IP address and
port as seen by the application server, or the private (local) and any public (observed) IP
addresses in use by the device (this information can be obtained by various means, for example
from some DNS servers).

        If the allocated and observed IP addresses are the same (i.e. NAT is not in use) then
the same address should be specified for both publicAddress and privateAddress.

        If NAT64 is in use, the device should be identified by its publicAddress and
publicPort, or separately by its allocated IPv6 address (field ipv6Address of the Device
object)

        In all cases, publicAddress must be specified, along with at least one of either
privateAddress or publicPort, dependent upon which is known. In general, mobile devices cannot
be identified by their public IPv4 address alone.
      properties:
        publicAddress:
```

```
          $ref: "#/components/schemas/SingleIpv4Addr"
        privateAddress:
          $ref: "#/components/schemas/SingleIpv4Addr"
        publicPort:
          $ref: "#/components/schemas/Port"
      anyOf:
        - required: [publicAddress, privateAddress]
        - required: [publicAddress, publicPort]
      example:
        publicAddress: "84.125.93.10"
        publicPort: 59765

    SingleIpv4Addr:
      description: A single IPv4 address with no subnet mask
      type: string
      format: ipv4
      example: "84.125.93.10"

    Port:
      description: TCP or UDP port number
      type: integer
      minimum: 0
      maximum: 65535

    DeviceIpv6Address:
      description: |
        The user device should be identified by the observed IPv6 address, or by any single
IPv6 address from within the subnet allocated to the device (e.g. adding ::0 to the /64
prefix).
      type: string
      format: ipv6
      example: 2001:db8:85a3:8d3:1319:8a2e:370:7344

    IncludeAPIRoot:
      description: |
        Boolean filter to control whether the response includes the Operator's API gateway
root URL. By default, the root URL is not included.
      type: boolean
      default: false
      example: true

    includeAuthProviderConfiguration:
      description: |
        Boolean filter to indicate whether the response should contain the discovery endpoint
of the Operator's authorisation server
      type: boolean
      default: false
      example: true

    PortabilitySearchMode:
      description: |
        For use in regions with Mobile Number Portability. This optional field represents the
consumers preference when performing phone number searches. The shallow option directs Telco
Finder to search only its internal records (e.g. cache). This method can be preferred to avoid
higher monetary costs associated with extended searches. The full search triggers a
comprehensive search against all external systems, providing more thorough results at a
potentially higher cost and ensuring up-to-date information by bypassing stale cached data.
      type: string
      enum: ["SHALLOW", "DEEP"]
      example: "SHALLOW"
      nullable: true


#---------------------------------------------------------------------------#
#                   Request & Response Payload Definitions                  #
#---------------------------------------------------------------------------#
```

```
    TelcoFinderSearchRequestBody:

      type: object
      required:
        - target
      properties:
        target:
          $ref: "#/components/schemas/Target"
        includeApiRoot:
          $ref: "#/components/schemas/IncludeAPIRoot"
        includeAuthProviderConfiguration:
          $ref: "#/components/schemas/includeAuthProviderConfiguration"
        portabilitySearchMode:
          $ref: "#/components/schemas/PortabilitySearchMode"

    TelcoFinderSearchResponseBody:
      type: object
      required:
        - operatorId
      properties:
        operatorId:
          type: string
        apiRoot:
          type: string
        authProviderConfiguration:
          type: string
      additionalProperties: false


#---------------------------------------------------------------------------#
#                   4xx and 5xx Error Response Definitions                  #
#---------------------------------------------------------------------------#

  responses:

    Generic400:
      description: Invalid input
      headers:
        x-correlator:
          $ref: '#/components/headers/x-correlator'
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ErrorInfo'
          example:
            status: 400
            code: INVALID_ARGUMENT
            message: 'Invalid input'

    Generic401:
      description: Unauthorized
      headers:
        x-correlator:
          $ref: '#/components/headers/x-correlator'
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ErrorInfo'
          example:
            status: 401
            code: AUTHENTICATION_REQUIRED
            message: 'Authentication required'

    Generic422:
      description: Target not identified by operator. For example, IP is not in range
supported by Telco Finder.
      headers:
```

```
        x-correlator:
          $ref: '#/components/headers/x-correlator'
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ErrorInfo'
          example:
            status: 422
            code: TARGET_NOT_APPLICABLE
            message: 'The service is not available for the requested target.'

    Generic403:
      description: Forbidden
      headers:
        x-correlator:
          $ref: '#/components/headers/x-correlator'
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ErrorInfo'
          example:
            status: 403
            code: PERMISSION_DENIED
            message: 'Operation not allowed'

    Generic404:
      description: Not found
      headers:
        x-correlator:
          $ref: '#/components/headers/x-correlator'
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ErrorInfo'
          example:
            status: 404
            code: NOT_FOUND
            message: 'The specified resource is not found'

    Generic500:
      description: Internal server error
      headers:
        x-correlator:
          $ref: '#/components/headers/x-correlator'
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ErrorInfo'
          example:
            status: 500
            code: INTERNAL
            message: 'Internal server error'

    Generic503:
      description: Service unavailable
      headers:
        x-correlator:
          $ref: '#/components/headers/x-correlator'
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ErrorInfo'
          example:
            status: 503
            code: UNAVAILABLE
            message: 'Service unavailable'
```

```
#---------------------------------------------------------------------------#
#                            Security Schemes                               #
#---------------------------------------------------------------------------#

  securitySchemes:
    openId:
      type: openIdConnect
      openIdConnectUrl: /.well-known/openid-configuration
```

A.2 Routing API specification (OpenAPI Specification format)

```
openapi: 3.0.0
info:
  title: API to provide Telco-Finder with Operator's routing rules
  description: |

    This is the definition of the [GSMA Telco Routing API](https://github.com/GSMA-Open-
Gateway/Open-Gateway-Documents/blob/main/Chapters/Chapter%2005.md#telco-routing-api).


    # Relevant Definitions and concepts

    * **Telco Finder**: allows any component of the Open Gateway architecture to know
information about the operator to which a users belongs as well as the endpoints that it will
have to use if it wants to carry out any operation about their.
    * **Telco Proxy**: Component in the Open Gateway Architecture which redirects Application
API calls to the proper Operator API based on the end-user id. It uses Telco Finder to look
for the end-user's operator.
    * **MSISDN**: Mobile Station Integrated Service Digital Network, phone number.
    * **MCC**: Mobile Country Code, consists of three decimal digits, the first of which
identifies the geographic region.
    * **MNC**: Mobile Network Code, consists of two or three decimal digits.
    * **Telco Finder Routing Rule**: mapping rule which match a range of user IDs (IP address,
MSISDN prefix or network ID) to an static operator resolution (operator name and related
links)
    or to a dynamic resolution which requires a second level resolution.


    # API Functionality

    Telco Routing API provides Telco Finder a set of routing rules to find the operator owning
an end-user (identified by MSISDN or IP/port).
    The Telco Finder aggregates routing rules from Operators and creates a regional routing
table to resolve search queries from a Telco Proxy.


    In countries where number portability is required, MSISDN are mappined onto network IDs.


    Each operator provides an end-point of Telco Routing API which provides routing rules.
Each routing rule is represented by a JSON Object with next members:


    * `ipv4`: array of strings in CIDR notation. List of IP V4 ranges (example:
`23.124.1.200/20`).

    * `ipv6`: array of strings in CIDR notation. List of IP V6 ranges (example:
`ff22:0:0:ab:23:1a:346:7332/64`).

    * `msisdnPrefix`: array of strings representing a msisdn prefix stating by the country
code (example: `+100234`)

    * `network`: array of strings representing a MCC_MNC code (example: `23401`)

    * `static`: JSON Object representing an static routing rule which is equivalent to the
Telco Finder result components:
```

```
        * `operatorId`: operator brand of owning the end-user.

        * `apiRoot`: the root URL of the API Gateway managed by the operator.

        * `authProviderConfiguration`: the discovery endpoint of the operator's authorisation
server. This is a standardised URL in [OpenID Connect](https://openid.net/specs/openid-
connect-discovery-1_0.html#ProviderMetadata) and [OAuth
2.0](https://datatracker.ietf.org/doc/html/rfc8414#section-3) that allows clients to
dynamically retrieve configuration metadata about the authorisation server.

    * `dynamic`: JSON Object representing the reference to a second level Telco Finder end-
point to resolve multi-brand routing:

        * `authProviderConfiguration`: the discovery endpoint of the operator's authorisation
server. This is a standardised URL in [OpenID Connect](https://openid.net/specs/openid-
connect-discovery-1_0.html#ProviderMetadata) and [OAuth
2.0](https://datatracker.ietf.org/doc/html/rfc8414#section-3) that allows clients to
dynamically retrieve configuration metadata about the authorisation server.

        * `telcoFinder`: URL of the second level Telco Finder

    Each Telco Routing Rule, at least, must have any of `ipv4`, `ipv6`, `msisdnPrefix` or
`network` member and one of `static` or `dynamic` member.

    # Resources and Operations overview

    There is a single resource in the API, which returns and array of Telco Finder routing
rules.

    This is an API intended to be used by Telco Finder to gather Operator's routing
configuration.
    No end-user personal data is managed. Therefore, API is intended to be used in 2-legged
mode.

  termsOfService: http://swagger.io/terms/
  contact:
    email: project-email@sample.com
  license:
    name: Apache 2.0
    url: https://www.apache.org/licenses/LICENSE-2.0.html
  version: 2.0.0-wip
tags:
- name: Routing
  description: Information about Telco-Finder routing table
paths:
  /routing:
    get:
      tags:
        - Routing
      operationId: getRoutingTable
      security:
        - openId:
          - telco-routing:read
      parameters:
        - $ref: '#/components/parameters/x-correlator'
      responses:
        "200":
          description: Routing table found
          headers:
            x-correlator:
              $ref: '#/components/headers/x-correlator'
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/RoutingDescription"
              examples:
                'Static Routing Rule':
```

```
                  description: 'Static Routing for IP ranges and Network Ids: Telco Finder
will return the operatorId and base api URL in the rule'
                  value:
                    - ipv4:
                        - '23.124.1.200/20'
                        - '34.231.2.120/22'
                      ipv6:
                        - 'ff22:0:0:ab:23:1a:346:7332/64'
                      network:
                        - '23405'
                        - '23411'
                      static:
                        operatorId: "OPERATOR_ID"
                        authProviderConfiguration: "https://auth.operator.com/.well-
known/openid-configuration"
                        apiRoot: "https://example.operator.com"
                'Static & Dynamic Routing Rules':
                  description: >
                    Telco Finder, if user id found in range will:

                      * Dynamic Routing for network ids 23405 and 23411 (MCC_MNC): Telco
Finder will call the WebFinger url in rule for Operator resolution

                      * Static Routing for IP ranges: Telco Finder will return operatorId and
links from rule
                  value:
                    - ipv4:
                        - '23.124.1.200/20'
                        - '34.231.2.120/22'
                      ipv6:
                        - 'ff22:0:0:ab:23:1a:346:7332/64'
                      static:
                        operatorId: "OPERATOR_ID"
                        authProviderConfiguration: "https://auth.operator.com/.well-
known/openid-configuration"
                        apiRoot: "https://example.operator.com"
                    - network:
                        - '23405'
                        - '23411'
                      dynamic:
                        authProviderConfiguration: "https://auth.operator.com/.well-
known/openid-configuration"
                        telcoFinder: “https://apis.operator.com/telco-finder/v1”

                'Static Routing Rule for IPs and MSISDN Prefixes':
                  description: 'Static Routing for IP ranges and MSISDN prefixes: Telco Finder
will return the operatorId'
                  value:
                    - ipv4:
                        - '23.124.1.200/20'
                        - '34.231.2.120/22'
                      ipv6:
                        - 'ff22:0:0:ab:23:1a:346:7332/64'
                      msisdnPrefix:
                        - '+100235'
                        - '+100333'
                      static:
                        operatorId: "OPERATOR_ID"
                        authProviderConfiguration: "https://auth.operator.com/.well-
known/openid-configuration"
                        apiRoot: "https://example.operator.com"

        '401':
          $ref: '#/components/responses/Error401Unauthenticated'
        '403':
          $ref: '#/components/responses/Error403PermissionDenied'
        '404':
```

```
          $ref: '#/components/responses/Error404NotFound'
        "500":
          $ref: "#/components/responses/Error500Internal"
        '503':
          $ref: '#/components/responses/Error503Unavailable'
        '504':
          $ref: '#/components/responses/Error504Timeout'


components:
  headers:
    x-correlator:
      description: Correlation id for the different services
      schema:
        type: string
  parameters:
    x-correlator:
      name: x-correlator
      in: header
      description: Correlation id for the different services
      schema:
        type: string
  schemas:
    StaticRouting:
      type: object
      required:
        - operatorId
        - authProviderConfiguration
        - apiRoot
      description: |
        A static routing entry
      properties:
        operatorId:
          type: string
          description: Operator identifier.
        authProviderConfiguration:
          type: string
          description: the discovery endpoint of the operator's authorisation server
        apiRoot:
          type: string
          description: the root URL of the API Gateway managed by the operator
    DynamicRouting:
      type: object
      description: |
        A dynamic routing entry
      required:
        - authProviderConfiguration
        - telcoFinder
      properties:
        authProviderConfiguration:
          type: string
          description: the discovery endpoint of the operator's authorisation server
        telcoFinder:
          type: string
          description: URL of the second level Telco Finder
    RoutingRule:
      type: object
      description: A routing entry
      minProperties: 1
      properties:
        ipv4:
          type: array
          items:
            type: string
            description: A list of IPV4 addresses.
          example: ["23.124.1.200/20", "34.231.2.120/22"]
        ipv6:
```

```
          type: array
          items:
            type: string
            description: A list of IPV6 addresses.
          example: ["ff22:0:0:ab:23:1a:346:7332/64"]
        network:
          type: array
          description: A list of network codes.
          items:
            type: string
            description: 'Network ID consisting of MCC (E.164 Country Code) and MNC, format is 
5 or 6 digits.'
            pattern: '^\d{5,6}$'
        msisdnPrefix:
          type: array
          description: A list of MSISDN prefixes.
          items:
            type: string
            description: 'Phone number prefix: MSISDN in ''E164 with +'' format.'
            example: '+10023'
    StaticRule:
      type: object
      allOf:
        - $ref: "#/components/schemas/RoutingRule"
      required:
        - static
      properties:
        static:
          $ref: "#/components/schemas/StaticRouting"
    DynamicRule:
      type: object
      allOf:
        - $ref: "#/components/schemas/RoutingRule"
      required:
        - dynamic
      properties:
        dynamic:
          $ref: "#/components/schemas/DynamicRouting"
    RoutingEntry:
      oneOf:
        - $ref: "#/components/schemas/StaticRule"
        - $ref: "#/components/schemas/DynamicRule"
    RoutingDescription:
      type: array
      description: |
        A list of routing entries
      items:
        $ref: "#/components/schemas/RoutingEntry"
    ModelError:
      type: object
      required:
        - status
        - code
        - message
      properties:
        status:
          type: integer
          description: "HTTP Status code"
        code:
          type: string
          description: "A code value within the allowed set of values for this error"
        message:
          type: string
          description: "A human readable description of what the event represent"
    Internal:
      allOf:
        - $ref: "#/components/schemas/ModelError"
```

```
      type: object
      properties:
        code:
          type: string
          enum: [INTERNAL]
          description: "Unknown server error. Typically a server bug."
    Unauthenticated:
      allOf:
        - $ref: '#/components/schemas/ModelError'
        - type: object
          properties:
            code:
              type: string
              enum: [UNAUTHENTICATED]
              description: 'Request not authenticated due to missing, invalid, or expired
credentials.'
    NotFound:
      allOf:
        - $ref: '#/components/schemas/ModelError'
        - type: object
          properties:
            code:
              type: string
              enum: [NOT_FOUND]
              description: 'The specified resource is not found.'
    PermissionDenied:
      allOf:
        - $ref: '#/components/schemas/ModelError'
        - type: object
          properties:
            code:
              type: string
              enum: [PERMISSION_DENIED]
              description: 'Client does not have sufficient permissions to perform this
action.'
    Unavailable:
      allOf:
        - $ref: '#/components/schemas/ModelError'
        - type: object
          properties:
            code:
              type: string
              enum: [UNAVAILABLE]
              description: 'Request timeout exceeded'
    Timeout:
      allOf:
          - $ref: '#/components/schemas/ModelError'
          - type: object
            properties:
              code:
                type: string
                enum: [TIMEOUT]
                description: 'Request timeout exceeded'
  responses:
    Error401Unauthenticated:
      description: 'Request not authenticated due to missing, invalid, or expired
credentials.'
      headers:
        x-correlator:
          $ref: '#/components/headers/x-correlator'
      content:
          application/json:
            schema:
              $ref: '#/components/schemas/Unauthenticated'
            example:
              status: 401
              code: UNAUTHENTICATED
```

```
              message: 'Client not authenticated'
    Error403PermissionDenied:
      description: 'Client does not have sufficient permission.'
      headers:
        x-correlator:
          $ref: '#/components/headers/x-correlator'
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/PermissionDenied'
          example:
            status: 403
            code: PERMISSION_DENIED
            message: 'Client does not have sufficient permissions to perform this action.'
    Error404NotFound:
      description: 'The specified resource is not found.'
      headers:
        x-correlator:
          $ref: '#/components/headers/x-correlator'
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/NotFound'
          example:
            status: 404
            code: NOT_FOUND
            message: 'The specified resource is not found.'
    Error500Internal:
      description: "Server error."
      headers:
        x-correlator:
          $ref: '#/components/headers/x-correlator'
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/Internal"
          example:
            status: 500
            code: INTERNAL
            message: "Server error"
    Error503Unavailable:
      description: 'Service unavailable. Typically the server is down.'
      headers:
        x-correlator:
          $ref: '#/components/headers/x-correlator'
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Unavailable'
          example:
            status: 503
            code: UNAVAILABLE
            message: 'Service unavailable'
    Error504Timeout:
      description: 'Request time exceeded. If it happens repeatedly, consider reducing the
request complexity.'
      headers:
        x-correlator:
          $ref: '#/components/headers/x-correlator'
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Timeout'
          example:
            status: 504
            code: TIMEOUT
            message: 'Request timeout exceeded. Try it later'
```

```
  securitySchemes:
    openId:
      type: openIdConnect
      openIdConnectUrl: /.well-known/openid-configuration
servers:
  - url: "https://localhost:9091/telco-routing/v1"
```

A.3 Network ID API specification (OpenAPI Specification format)

```
openapi: 3.0.3
info:
    title: 'Network ID Resolution'
    description: "Allows to retrieve the network id (MCC+MNC) for a given mobile phone
number\n# Relevant Definitions and concepts\n\n - **Network ID**: The MCC followed by the MNC,
each phone number has only one network id.\n\nFind more information about MCC and MNC in the
[ETSI Technical Specification 123
003](https://www.etsi.org/deliver/etsi_ts/123000_123099/123003/17.10.00_60/ts_123003v171000p.p
df)\n\n# API Functionality\n This API allows the API Client to learn the specific network id
for a given mobile phone number. For example, for Open Gateway Telco Finder may be an API
Client."
    termsOfService: http://swagger.io/terms/
    contact:
        email: project-email@sample.com
    license:
        name: Apache 2.0
        url: https://www.apache.org/licenses/LICENSE-2.0.html
    version: 1.0.0
tags:
    - name: 'Network ID'
      description: 'Operations to Resolve the network code of a MSISDN'
paths:
    /resolve-network-id:
        post:
            description: 'Retrieve network id for a given phone number'
            security:
                - openId:
                    - network-id:resolve-network-id
            tags:
                - 'Network ID'
            operationId: resolveNetworkID
            requestBody:
                content:
                    application/json:
                        schema:
                            $ref: '#/components/schemas/PhoneNumber'
                required: true
            parameters:
                - $ref: '#/components/parameters/x-correlator'
            summary: 'Retrieve network id'
            responses:
                '200':
                    headers:
                        x-correlator:
                            $ref: '#/components/headers/x-correlator'
                    description: OK
                    content:
                        application/json:
                            schema:
                                $ref: '#/components/schemas/NetworkInfo'
                '400':
                    $ref: '#/components/responses/Error400NetworkIDInvalidArgument'
                '401':
                    $ref: '#/components/responses/Error401Unauthenticated'
                '403':
                    $ref: '#/components/responses/Error403PermissionDenied'
                '500':
                    $ref: '#/components/responses/Error500Internal'
```

```
                '503':
                    $ref: '#/components/responses/Error503Unavailable'
                '504':
                    $ref: '#/components/responses/Error504Timeout'
components:
    headers:
        x-correlator:
            schema:
                type: string
            description: 'Correlation id for the different services'
    schemas:
        ModelError:
            type: object
            required:
                - status
                - code
                - message
            properties:
                status:
                    type: integer
                    description: 'HTTP Status code'
                code:
                    type: string
                    description: 'A code value within the allowed set of values for this
error'
                message:
                    type: string
                    description: 'A human readable description of what the event represent'
        PhoneNumber:
            type: object
            required:
                - phoneNumber
            properties:
                phoneNumber:
                    type: string
                    description: 'Phone number for which network id is requested. MSISDN in
''E164 with +'' format.'
            description: 'Network connection information of a user provided as input context'
            example:
                phoneNumber: '+346667778880'
        NetworkInfo:
            type: object
            required:
                - networkId
            properties:
                networkId:
                    type: string
                    description: 'Network Identifier as MCC(E.164 Mobile Country Code)
concatenated with the MNC (Mobile Network Code). Format is 5 o 6 digits.'
                    pattern: '^\d{5,6}$'
                    example: '21407'
        NetworkIDInvalidArgument:
            allOf:
                - $ref: '#/components/schemas/ModelError'
                - type: object
                  required: [code]
                  properties:
                    code:
                        type: string
                        enum: [INVALID_ARGUMENT, NETWORK_ID.PHONE_NUMBER_NOT_FOUND]
                        description: 'Client specified an invalid argument, request body or
query param'
                        default: INVALID_ARGUMENT
        Unauthenticated:
            allOf:
                - $ref: '#/components/schemas/ModelError'
                - type: object
```

```
                  properties:
                    code:
                        type: string
                        enum: [UNAUTHENTICATED]
                        description: 'Request not authenticated due to missing, invalid, or
expired credentials.'
        PermissionDenied:
            allOf:
                - $ref: '#/components/schemas/ModelError'
                - type: object
                  properties:
                    code:
                        type: string
                        enum: [PERMISSION_DENIED]
                        description: 'Client does not have sufficient permissions to perform
this action.'
        Internal:
            allOf:
                - $ref: '#/components/schemas/ModelError'
                - type: object
                  properties:
                    code:
                        type: string
                        enum: [INTERNAL]
                        description: 'Unknown server error.Typically a server bug.'
        Unavailable:
            allOf:
                - $ref: '#/components/schemas/ModelError'
                - type: object
                  properties:
                    code:
                        type: string
                        enum: [UNAVAILABLE]
                        description: 'Request timeout exceeded'
        Timeout:
            allOf:
                - $ref: '#/components/schemas/ModelError'
                - type: object
                  properties:
                    code:
                        type: string
                        enum: [TIMEOUT]
                        description: 'Request timeout exceeded'
    responses:
        Error400NetworkIDInvalidArgument:
            description: "Problem with the client request.\n\n In addition to regular scenario
of INVALID_ARGUMENT, another scenarios may exist.\n- The indicated phone_number is well-formed
but is unknown (e.g. It is a landline phone number, cannot obtain any network id associated to
the mobile phone number, ...) (\"code\": \"NETWORK_ID.PHONE_NUMBER_NOT_FOUND\", \"message\":
\"Indicated phone_number is well-formed but is unknown\")"
            headers:
                x-correlator:
                    $ref: '#/components/headers/x-correlator'
            content:
                application/json:
                    schema:
                        $ref: '#/components/schemas/NetworkIDInvalidArgument'
                    examples:
                        response:
                            value:
                                status: 400
                                code: NETWORK_ID.PHONE_NUMBER_NOT_FOUND
                                message: 'Indicated phone_number is well-formed but is
unknown'
        Error401Unauthenticated:
            description: 'Request not authenticated due to missing, invalid, or expired
credentials.'
```

```
            headers:
                x-correlator:
                    $ref: '#/components/headers/x-correlator'
            content:
                application/json:
                    schema:
                        $ref: '#/components/schemas/Unauthenticated'
                    example:
                        status: 401
                        code: UNAUTHENTICATED
                        message: 'Client not authenticated'
        Error403PermissionDenied:
            description: 'Client does not have sufficient permission.'
            headers:
                x-correlator:
                    $ref: '#/components/headers/x-correlator'
            content:
                application/json:
                    schema:
                        $ref: '#/components/schemas/PermissionDenied'
                    example:
                        status: 403
                        code: PERMISSION_DENIED
                        message: 'Client does not have sufficient permissions to perform this
action.'
        Error500Internal:
            description: 'Server error.'
            headers:
                x-correlator:
                    $ref: '#/components/headers/x-correlator'
            content:
                application/json:
                    schema:
                        $ref: '#/components/schemas/Internal'
                    example:
                        status: 500
                        code: INTERNAL
                        message: 'Server error'
        Error503Unavailable:
            description: 'Service unavailable. Typically the server is down.'
            headers:
                x-correlator:
                    $ref: '#/components/headers/x-correlator'
            content:
                application/json:
                    schema:
                        $ref: '#/components/schemas/Unavailable'
                    example:
                        status: 503
                        code: UNAVAILABLE
                        message: 'Service unavailable'
        Error504Timeout:
            description: 'Request time exceeded. If it happens repeatedly, consider reducing
the request complexity.'
            headers:
                x-correlator:
                    $ref: '#/components/headers/x-correlator'
            content:
                application/json:
                    schema:
                        $ref: '#/components/schemas/Timeout'
                    example:
                        status: 504
                        code: TIMEOUT
                        message: 'Request timeout exceeded. Try it later'
    securitySchemes:
        openId:
```

```
            type: openIdConnect
            openIdConnectUrl: /.well-known/openid-configuration
    parameters:
        x-correlator:
            name: x-correlator
            in: header
            schema:
                type: string
            description: 'Correlation id for the different services'
servers:
    - url: 'https://localhost:9091/network-id/v1'
```

Annex B Document Management

B.1 Document History

Version	Date	Brief Description of Change	Approval Authority	Editor / Company
1.0	21st Nov 2024	New PRD OPG.10	ISAG	Diego Preciado Rojas / Nokia
2.0	27th May 2025	Update implementing OPG.10 CR1002	ISAG	Diego Preciado Rojas / Nokia
3.0	23rd Oct 2025	Update implementing OPG.10 CR1003	ISAG	Diego Preciado Rojas / Nokia
4.0	19th Feb 2026	Update implementing OPG.10 CR1004	ISAG	Diego Preciado Rojas / Nokia
5.0	17th Jul 2026	Update implementing OPG.10 CR1005	ISAG	Diego Preciado Rojas / Nokia

B.2 Other Information

Type	Description
Document Owner	Operator Platform Group
Editor / Company	Diego Preciado Rojas / Nokia

It is our intention to provide a quality product for your use. If you find any errors or omissions, please contact us with your comments. You may notify us at prd@gsma.com

Your comments or suggestions & questions are always welcome.