Open Gateway Technical Realisation Guidelines
Version 1.0
21st November 2024

Security Classification: Non-Confidential

Copyright Notice

Disclaimer

Compliance Notice

The information contain herein is in full compliance with the GSMA Antitrust Compliance Policy.

This Permanent Reference Document is classified by GSMA as an Industry Specification, as such it has been developed and is maintained by GSMA in accordance with the provisions set out GSMA AA.35 - Procedures for Industry Specifications.

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 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. This guideline outlines the required steps, best practices, and technical specifications necessary for successful implementation and utilisation of the GSMA Open Gateway.

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. 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 (MNOs):** 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.
- **Regulatory Bodies:** authorities overseeing compliance with industry standards and regulations.

1.3 Definitions

Term	Description
3-legged Access Token	An access token that involves three parties: the Resource Owner (User), the Authorisation Server (at the Operator or Aggregator), and the client (the AP's Application). In CAMARA, 3-legged access tokens are typically created using the OIDC Authorization Code flow or Client-Initiated Backchannel Authentication (CIBA) flow.
Aggregation Platform	A platform through which the Aggregator offers the services. [1]
Aggregator	An actor who provides (or combines) services exposed by different Operators and exposes them for use to the Application Providers. [1]

Term	Description
	Note: Exposed services by the Aggregator may differ from the services provided by the Operators. Synonyms: Channel Partner
Application Provider (AP)	The provider of the application that accesses the OGW Platform. Synonym: Developer
CAMARA	An open-source project within Linux Foundation that creates, develops and tests Service APIs and other API definitions.
Consent	The agreement of a subscriber to allow the usage of their personal data. This agreement can be revoked at any time. [1]
Consent Management	Service within the Operator's domain supporting create, read, update, and delete (CRUD) operations on Application-related Consent records. The service supports also notifying (to the interest parties) when a Consent record has changed. [1]
Data Protection	Legal control over access to and use of data stored in computers.
East/Westbound Interface	The interface between instances of Operator Platforms that extends an operator's reach beyond their footprint and subscriber base. [1]
End-User	A human participant who uses the application. A customer of the Application Provider. [1] Note: The End-User is not always the Subscriber.
Enterprise Platform	An application deployment and execution platform owned by an Enterprise.
Marketplace Platform	A platform where services (and APIs) are published and offered to 3rd parties. [1]
Northbound Interface	Interface through which an OGW Platform exposes services to Applications or Aggregation/Marketplace/Enterprise Platforms
Open Gateway (OGW) Platform	A realisation of a GSMA Operator Platform (defined in [1]), providing APIs for universal access to operator networks for developers.
Operate APIs	APIs used for the business management of APIs exposed by the GSMA Operator Platform on its NBI. These APIs are defined by TM Forum for the GSMA Open Gateway context per the requirements in [5].
Operator	An entity that exposes capabilities and/or resources of their network (IT, mobile) to the Application Providers (directly or via an Aggregator). [1] Synonyms: CSP (Communication Service Provider), MNO (Mobile Network Operator)
Service APIs	APIs abstracting Telco services exposed for use by Applications or Aggregation/Marketplace/Enterprise Platforms. Service APIs are defined by CAMARA.
Southbound Interface	Connects an Operator Platform with the specific operator infrastructure that delivers the network and charging services and capabilities. [1]
Subscriber	A client/customer of the Operator, identified by a unique identifier. [1]
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.
User Equipment (UE)	Any device with a SIM used directly by an End-User to communicate. [1]

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

1.4 Abbreviations

Term	Description
AP	Application Provider
API	Application Programming Interface
BSS	Business Support System
CIBA	OpenID Connect Client-Initiated Backchannel Authentication
CSP	Communication Service Provider
EWBI	East-West Bound Interface
GDPR	General Data Protection Regulation
JSON	JavaScript Object Notation
JWT	JSON Web Token
MNO	Mobile Network Operator
NBI	North Bound Interface
OGW	Open Gateway
OIDC	OpenID Connect
OP	Operator Platform
OSS	Operations Support System
QoD	Quality On Demand
REST	Representational State Transfer
SBI	South Bound Interface
SOAP	Simple Object Access Protocol
UE	Use Equipment
XML	Extensible Markup Language

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/

Ref	Doc Number	Title
[7]		CAMARA Security and Interoperability Profile (Fall24 meta-release) https://lf-camaraproject.atlassian.net/wiki/spaces/CAM/pages/14549015/Meta-release+Fall24#Commonalities-%26-ICM In the Commonalities & ICM table, take the ICM *Public Release Tag* > browse to Code section > /documentation/CAMARA-Security-Interoperability.md
[8]		CAMARA APIs access and user consent management (Fall24 meta-release) https://lf-camaraproject.atlassian.net/wiki/spaces/CAM/pages/14549015/Meta-release+Fall24#Commonalities-%26-ICM In the Commonalities & ICM able, take the ICM *Public Release Tag* > browse to Code section > /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
[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 – API Design Guidelines (Fall24 meta-release) https://lf-camaraproject.atlassian.net/wiki/spaces/CAM/pages/14549015/Meta-release+Fall24#Commonalities-%26-ICM

Ref	Doc Number	Title
		In the Commonalities & ICM table, take the Commonalities *Public Release Tag* > browse to Code > /documentation/API-design-guidelines.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/v5.0.0

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.

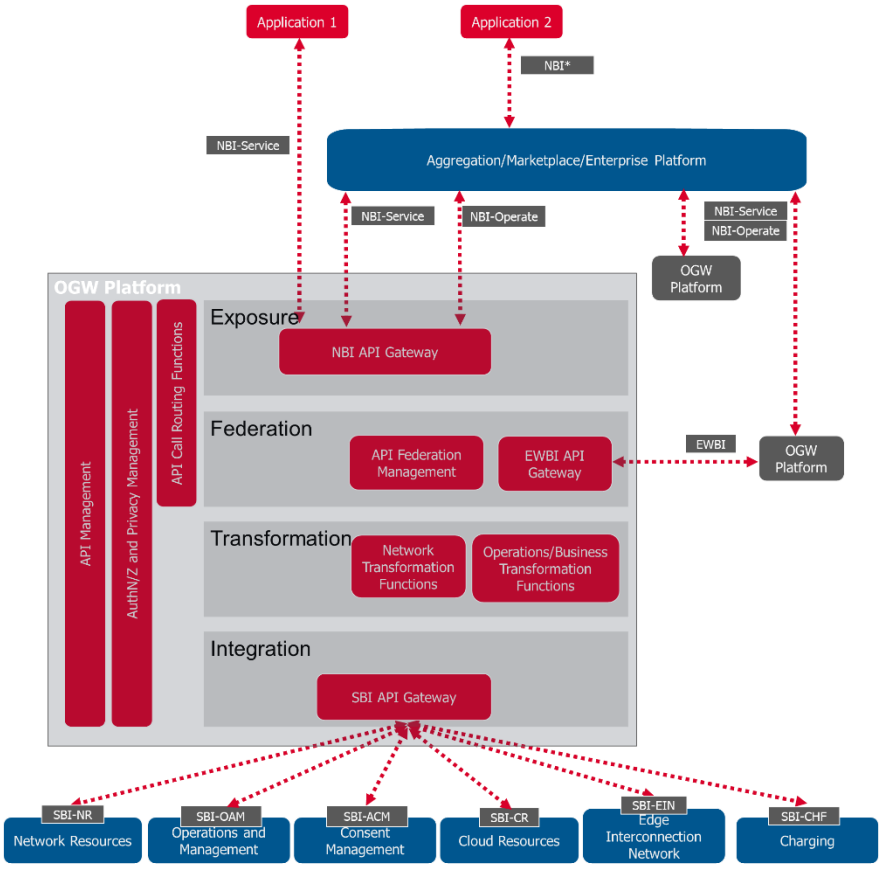

Figure 1: OGW Platform - High level architecture and functions

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 Consent Management function in the CSP domain (if allowed by local regulations)
        - Triggering Consent capture by the Consent Management in the CSP domain

- In federated scenarios, triggering Consent capture by the Consent Management function in the CSP domain of the federated partner

Note: OGW platform may relay procedures regarding Authentication / Authorisation / Identity / Privacy management to servers already in place in the CSP domain via SBI-ACM.

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 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 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 within an Open Gateway system, 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:

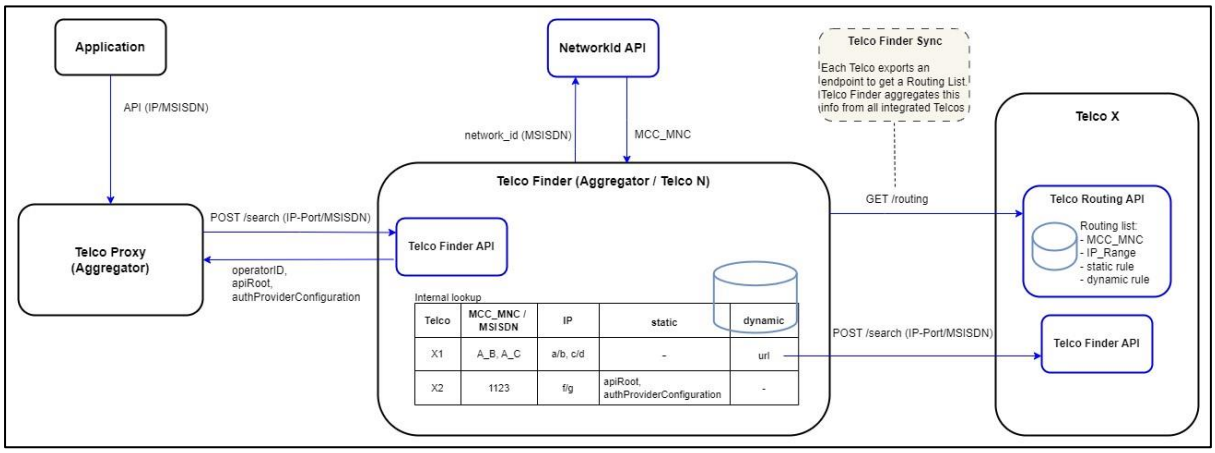

Figure 2: Telco Finder – main components

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 a 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 Operator 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:

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.

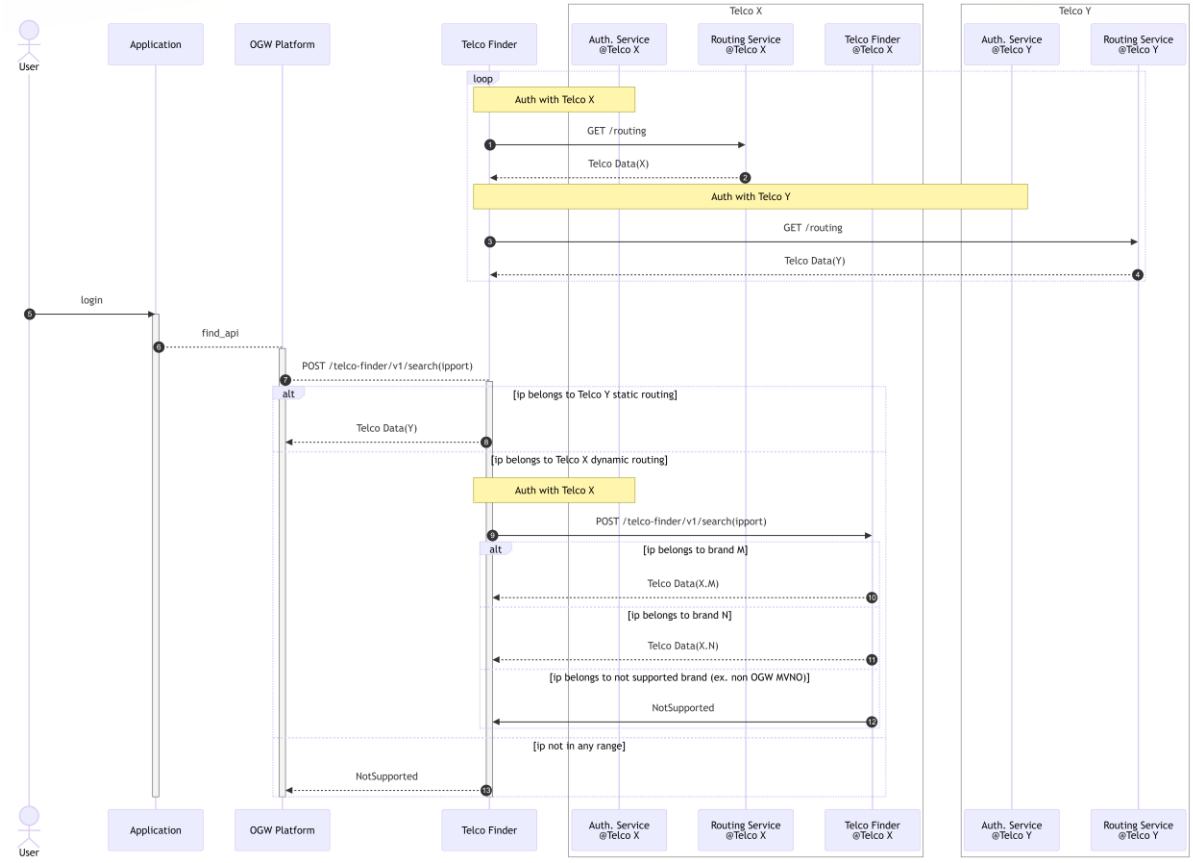

Figure 3: IP address lookup sequence diagram

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.

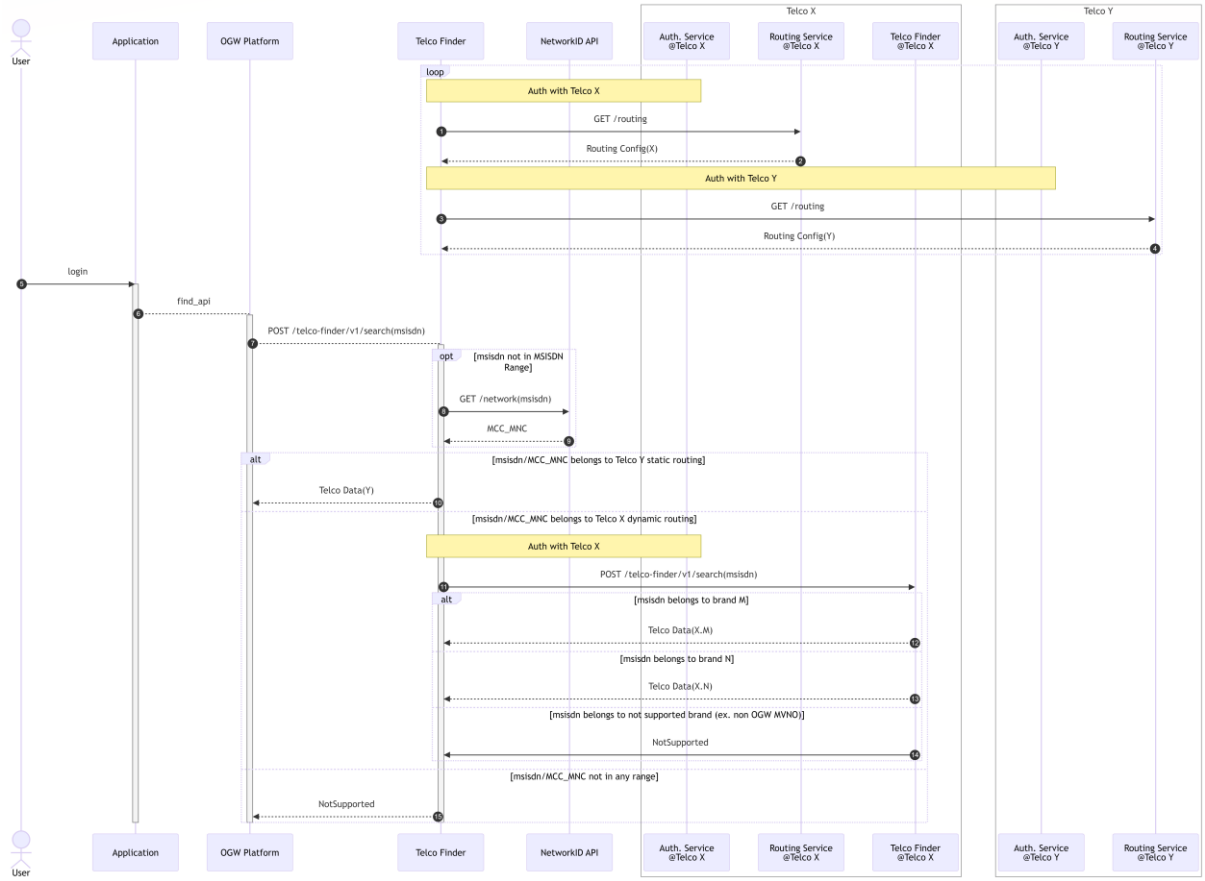

Figure 4: MSISDN lookup sequence diagram

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 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

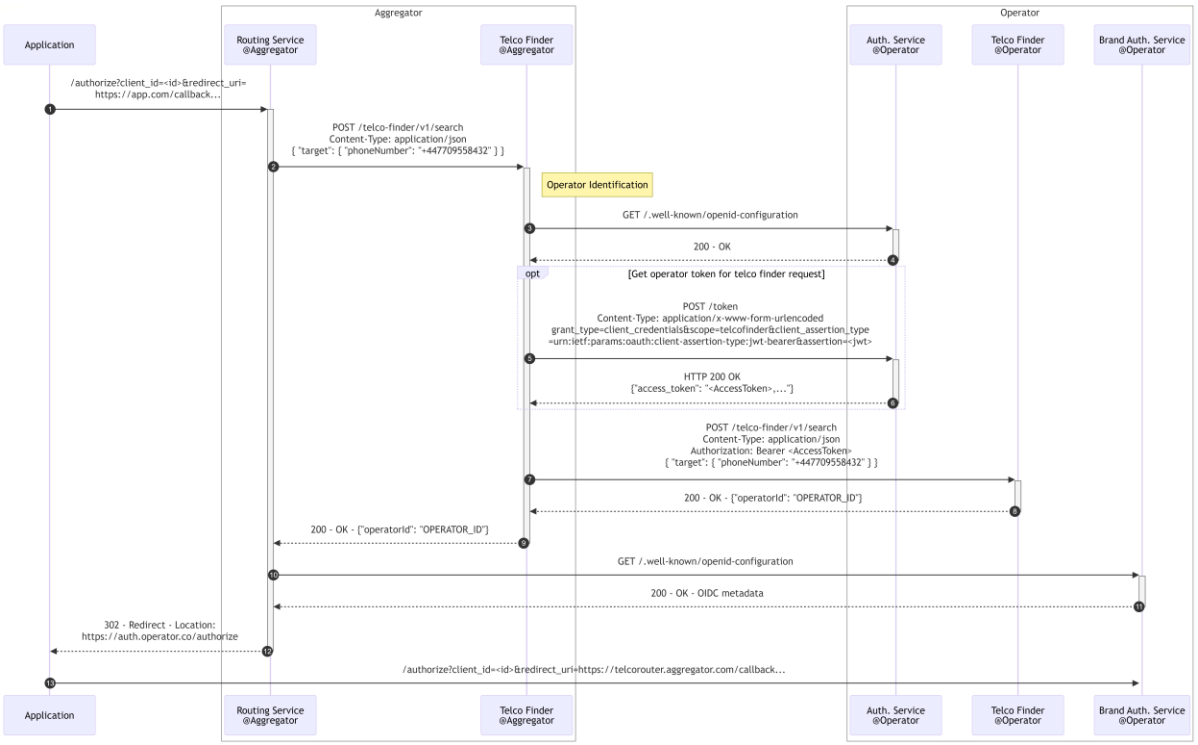

Figure 5: Multi-brand lookup

The diagram below demonstrates how the component delegates the obtaining of information related to the user from 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

2.2.2 Exposure Functions

The Exposure functions enable exposing Service APIs (to Applications or Aggregation/Marketplace/Enterprise Platforms) via the NBI-Service interface and Operate APIs (to Aggregation/Marketplace/Enterprise 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 Aggregation/Marketplace/Enterprise 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 Aggregation/Marketplace/Enterprise 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 CSP domain (SBI-ACM)
    - 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 Aggregation/Marketplace/Enterprise 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

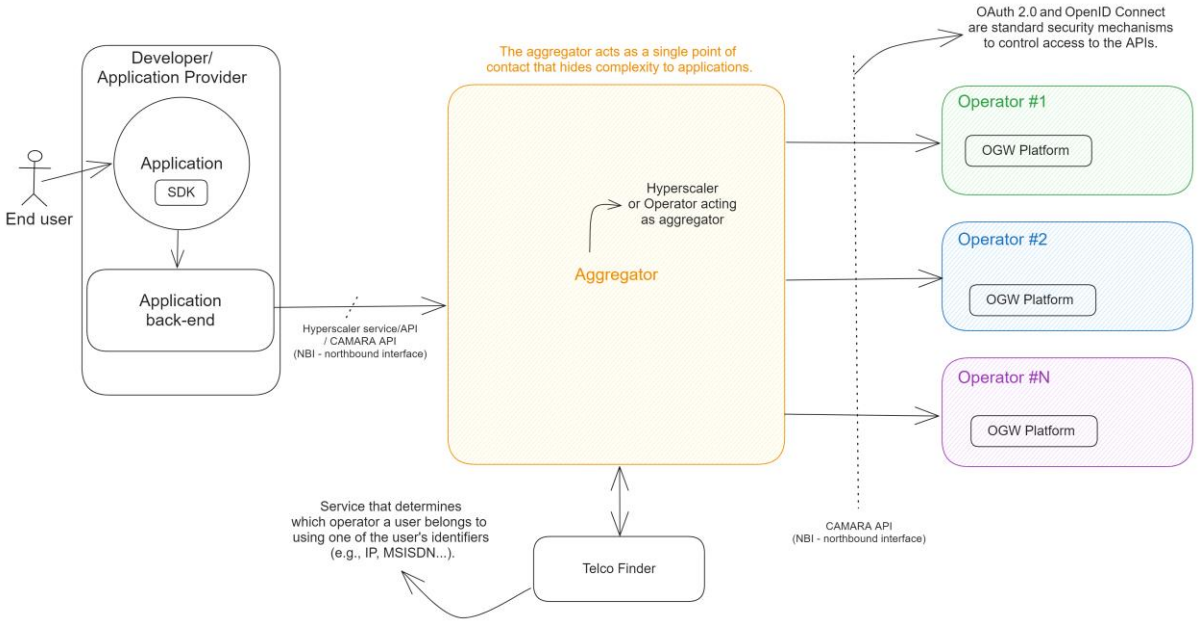

Figure 6: Aggregation model

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 (AP)**, 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 3.1.4) 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 6, 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. The Open Gateway 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 7 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.

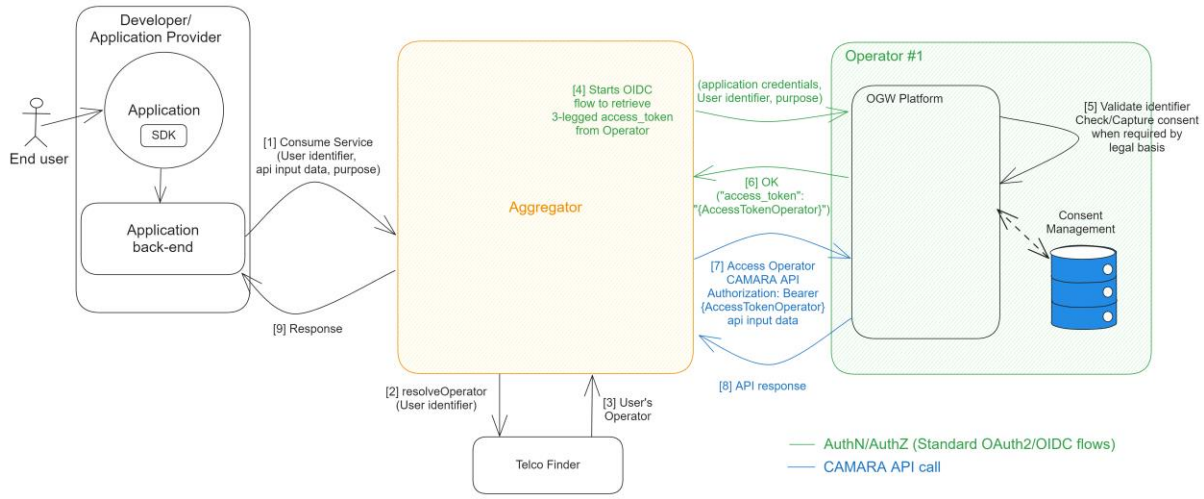

Figure 7: High level flow of the consumption of a Service API

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).

3.1.2 Detailed backend-based flow

To get into the details of consuming Service APIs, the Figure 8 describes the entire backend flow.

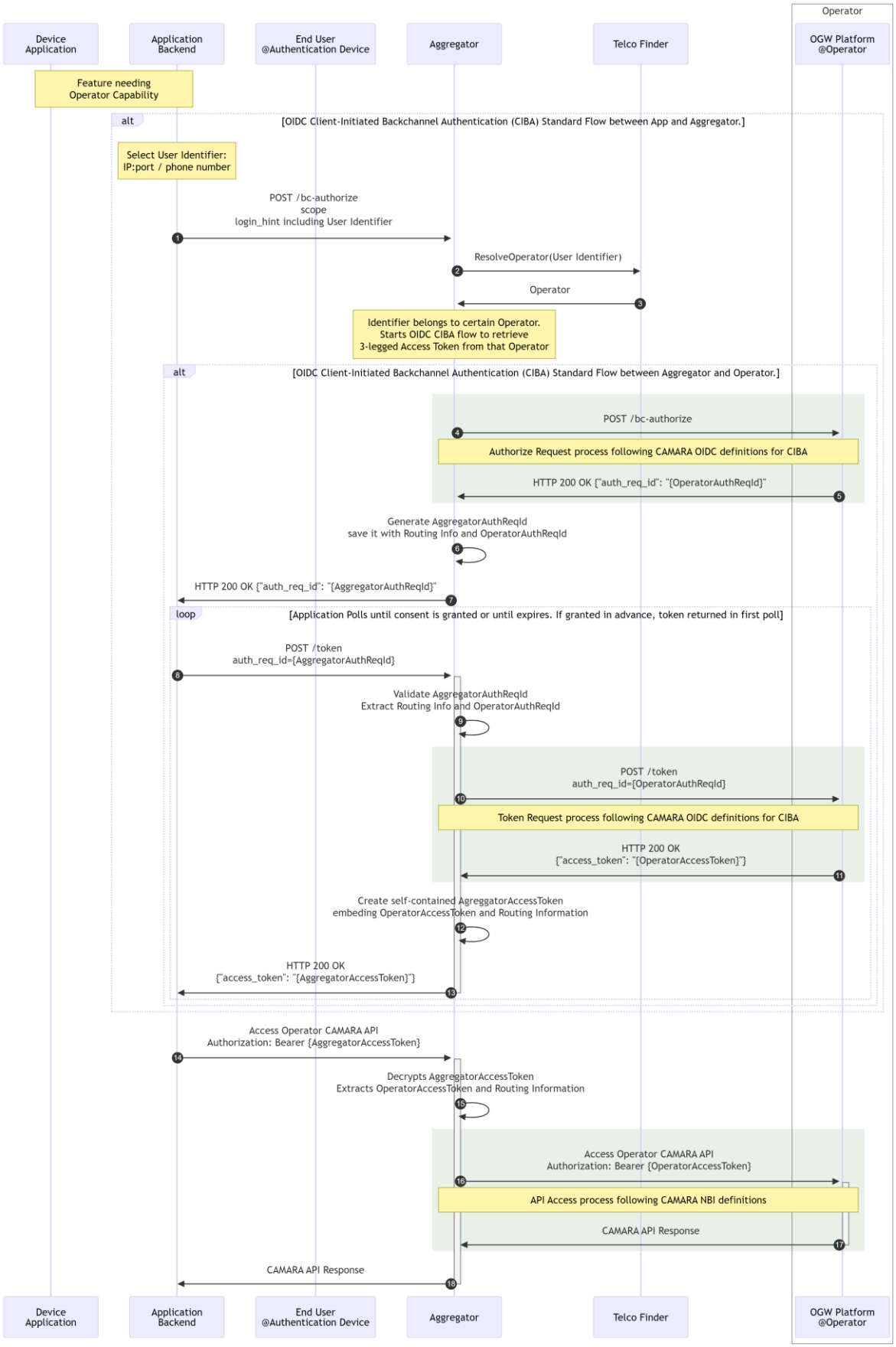

Figure 8: Detailed backend-based flow

Scenario description:

- The Aggregator exposes CAMARA APIs to the Application, access is protected by 3-legged OIDC access tokens.
- 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 3.1.4 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 to the Aggregator (in this case CIBA grant type); and an API gateway to handle API requests with issued access tokens.
    - Consent 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 authorisation request (/bc_authorize) a login_hint with a valid user identifier together with the credentials 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 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 authorise 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 Consent Management if user consent has already been given for that identifier and for the requested purpose.

When the authorisation 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 auth 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 auth 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).

3.1.3 Detailed frontend based flow

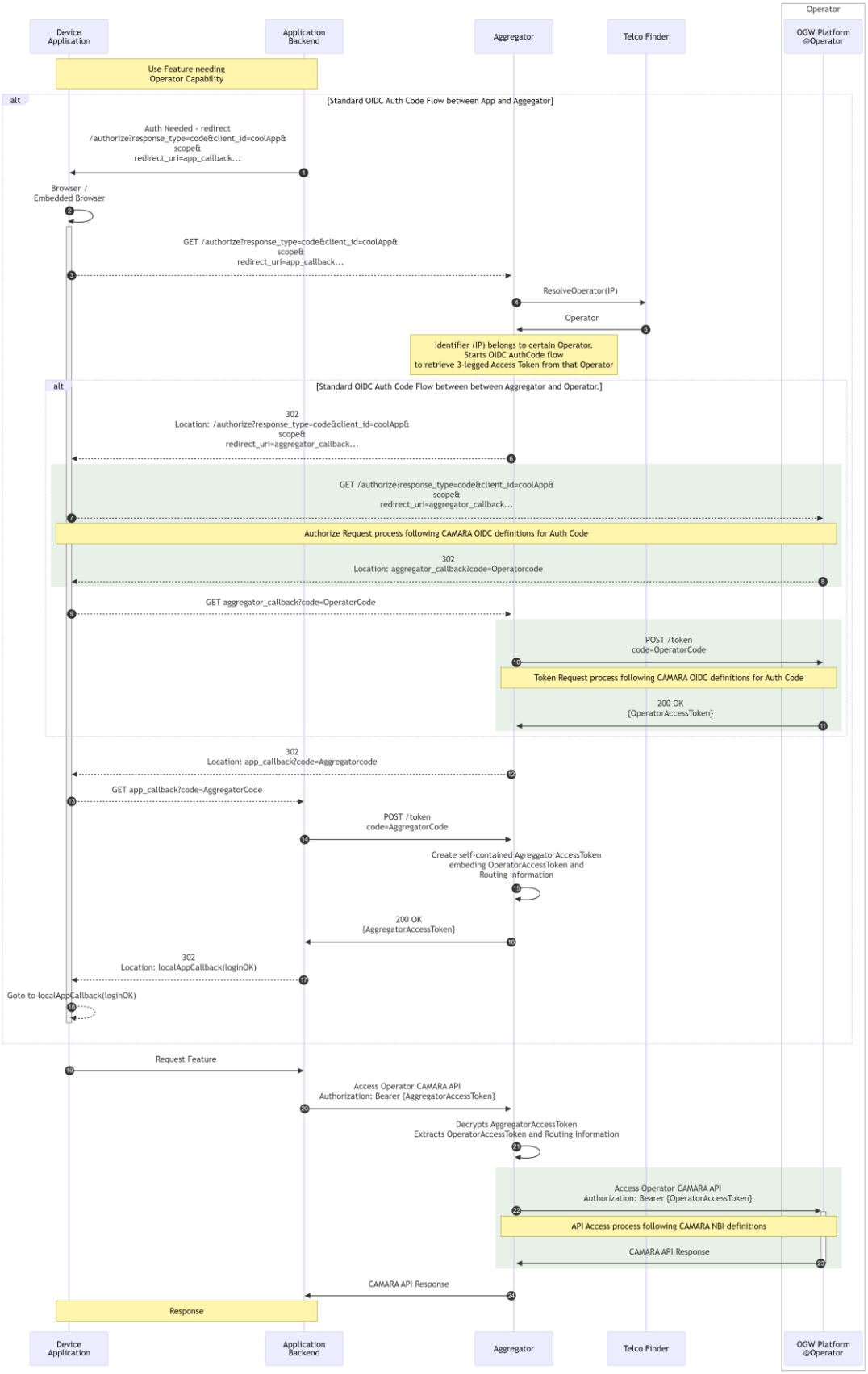

Figure 9: Detailed frontend based flow

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 (Steps 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 Consent Management if user consent has already been given for that identifier and for the requested purpose.

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).

3.1.4 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.

3.1.4.1 Enhanced Service API backend-based flow

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].

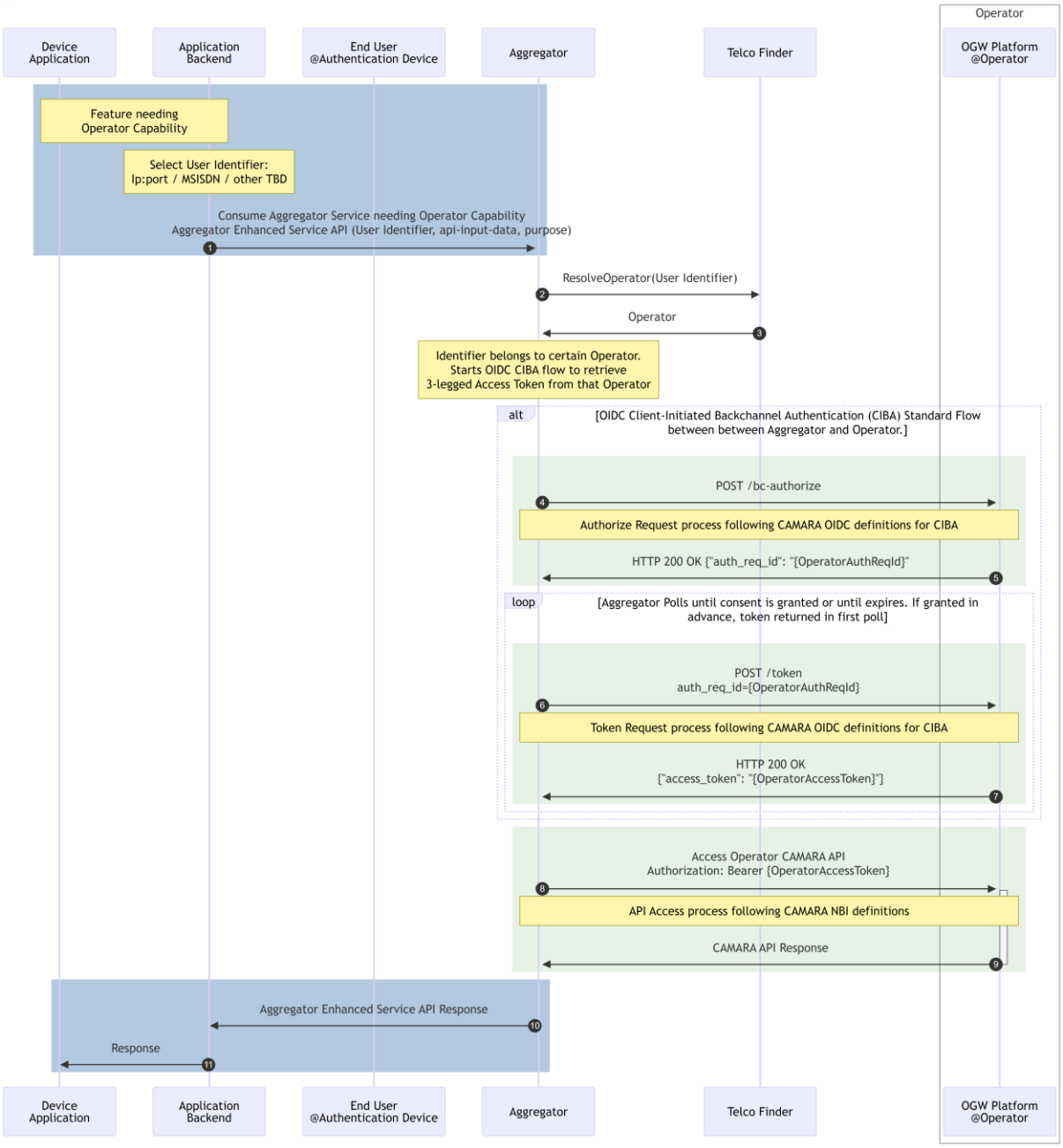

Figure 10: Enhanced Service API backend based flow

3.1.4.2 Enhanced Service API frontend-based 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].

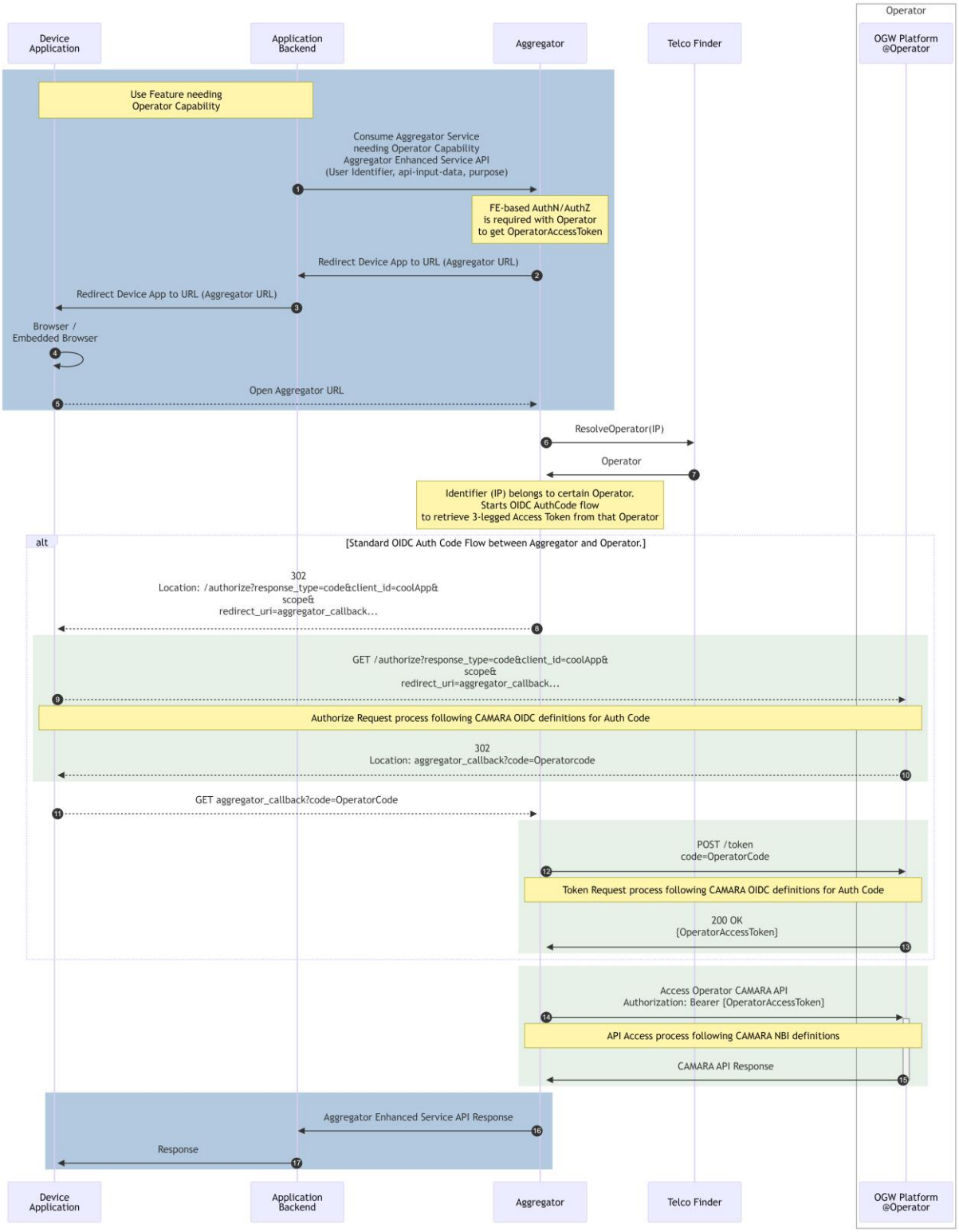

Figure 11: Enhanced Service API frontend-based flow

3.1.5 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.

3.1.5.1 Request a Refresh Token in frontend-based flow

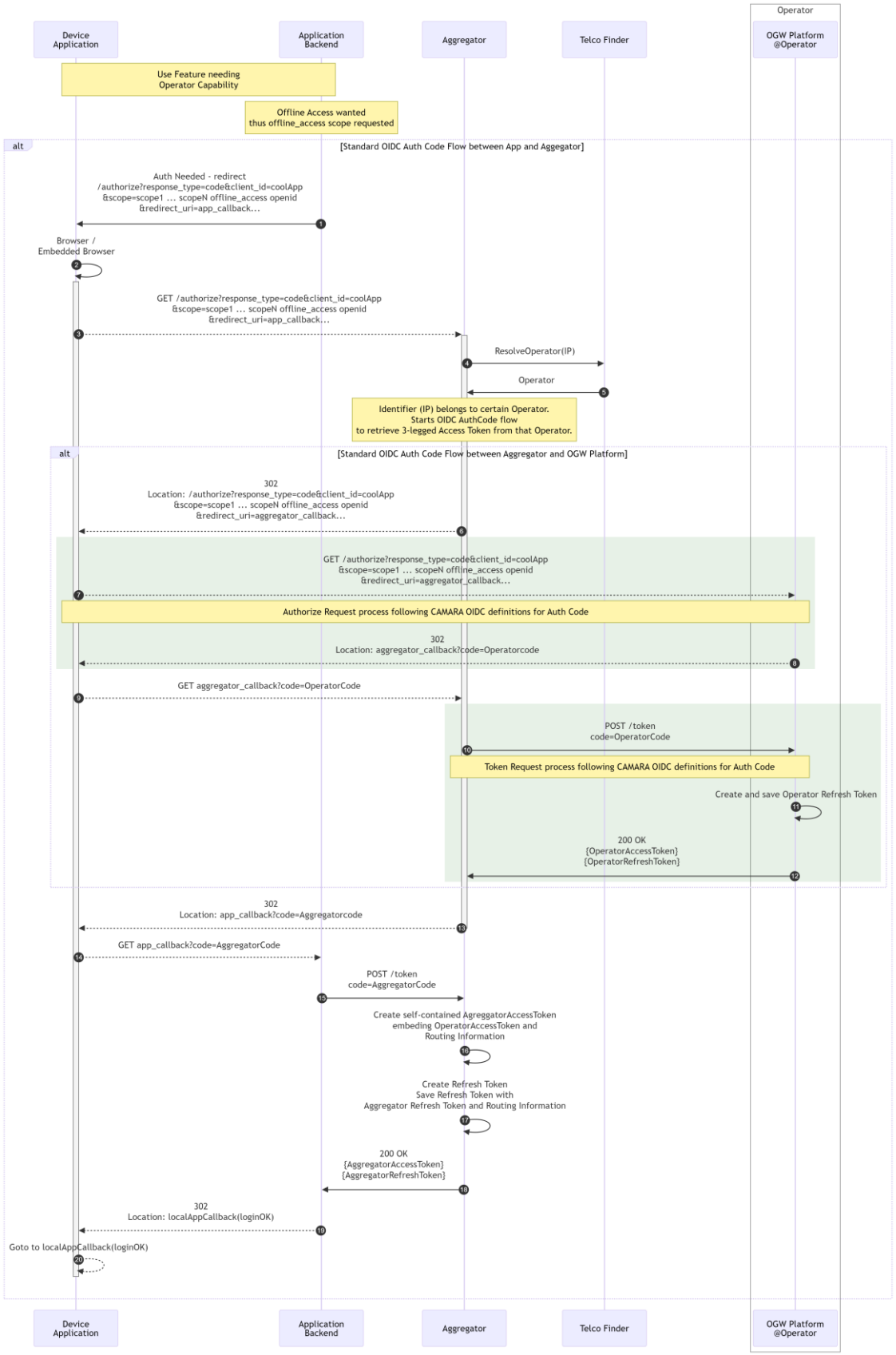

Figure 12: Request a Refresh Token in frontend-based flow

Scenario description:

- The same considerations that apply to general call flows in section 3.1.3 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 3.1.3.

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 3.1.3.

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).

3.1.5.2 Request a Refresh Token in backend-based flow

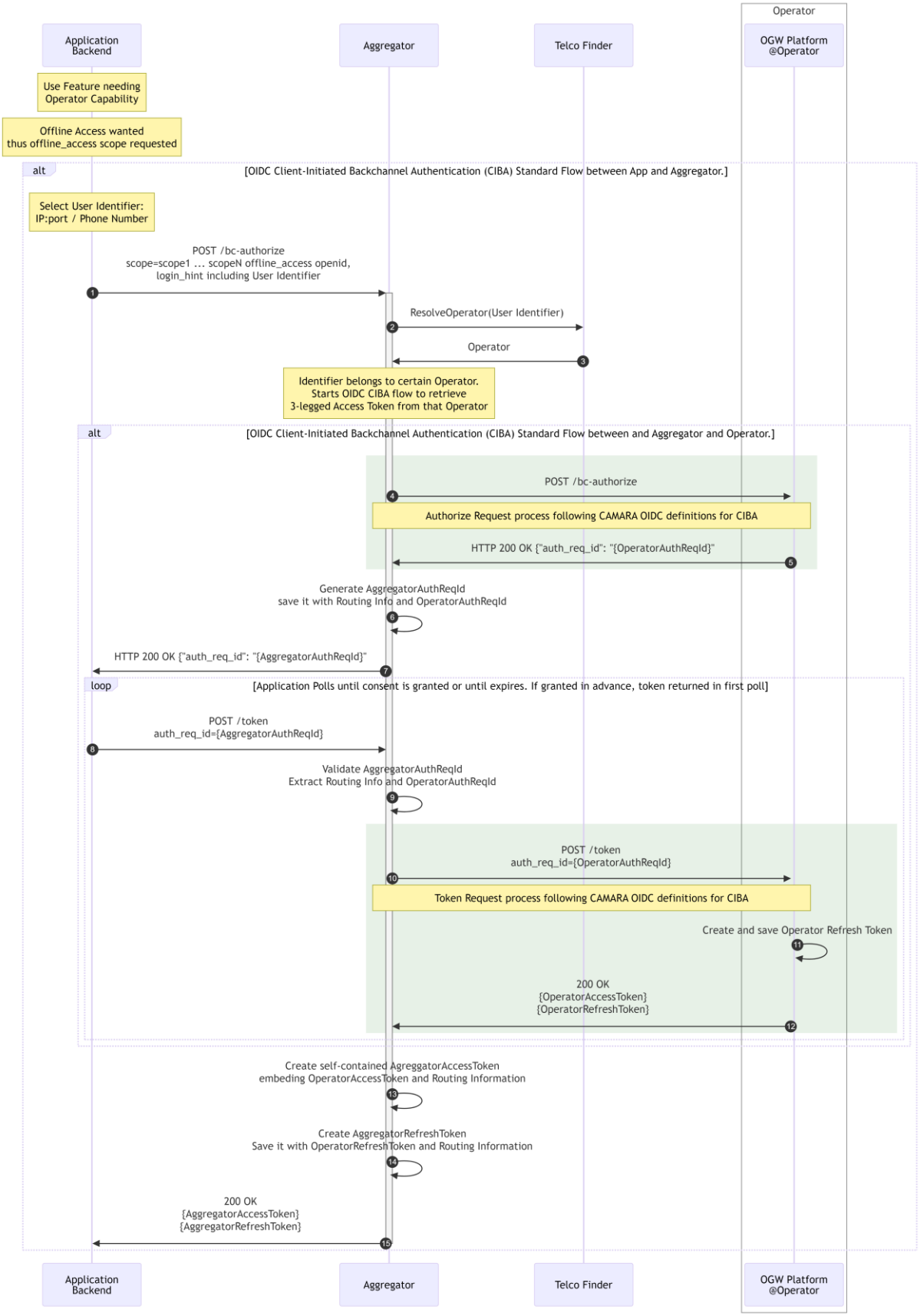

Figure 13: Request a Refresh Token in backend-based flow

Scenario description

- The same considerations that apply to general call flows in section 3.1.2 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 3.1.2.

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 3.1.2.

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 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).

3.1.5.3 Refresh token flow

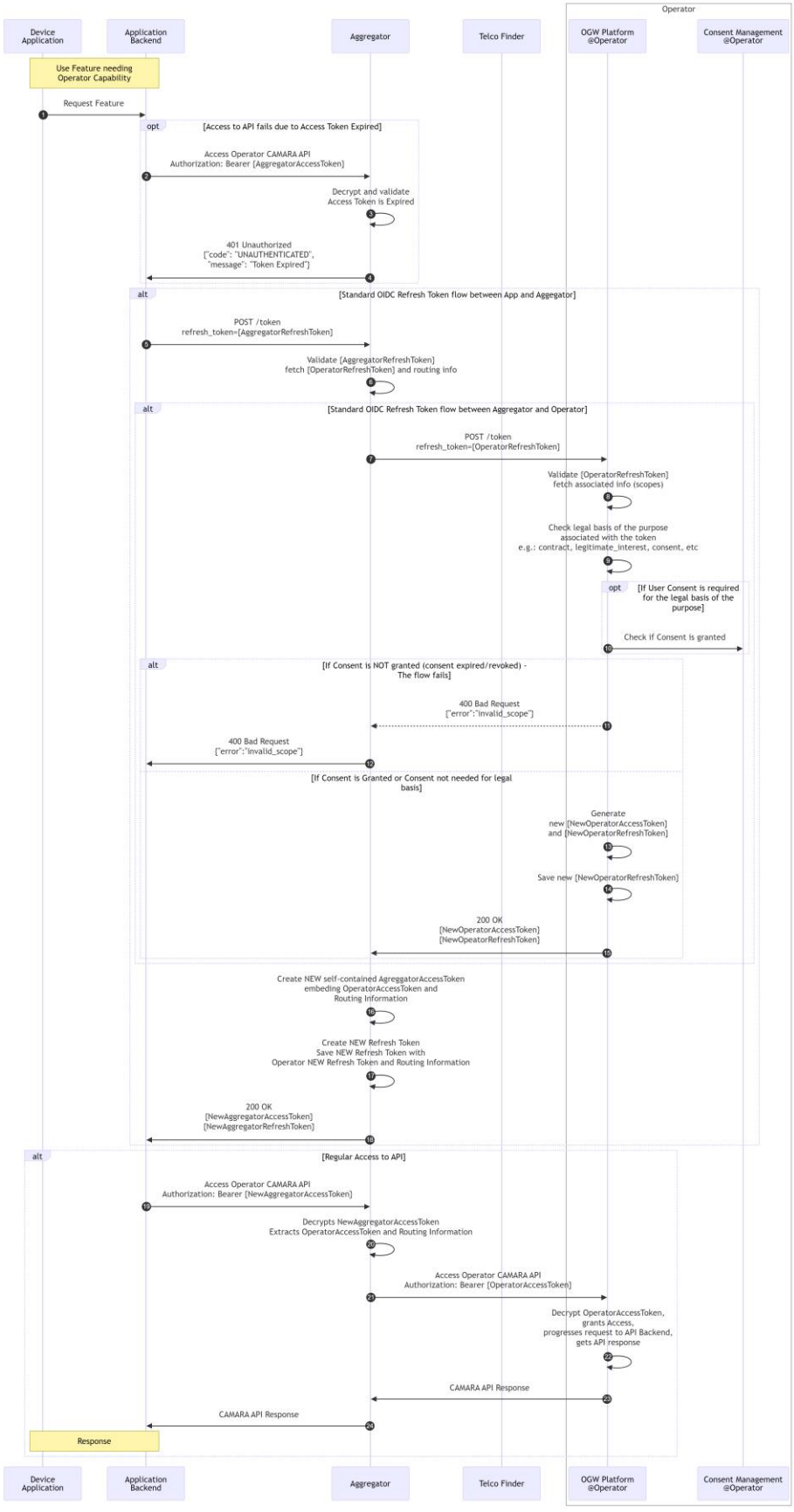

Figure 14: Refresh token flow

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.

3.2 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.

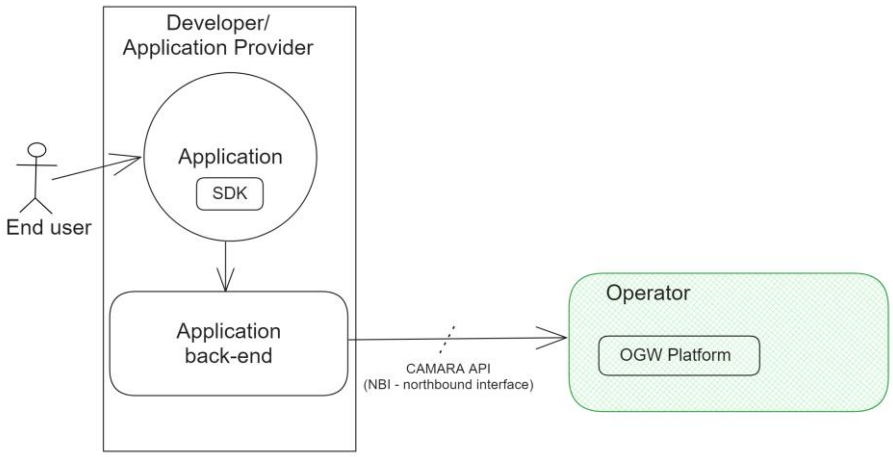

Figure 15: Direct Integration Developer – Operator (Single Operator)

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 APIs using the CAMARA standard NBI, ensuring a clear and consistent integration process.
- Known endpoint: since there is only one Operator, the endpoint for 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.

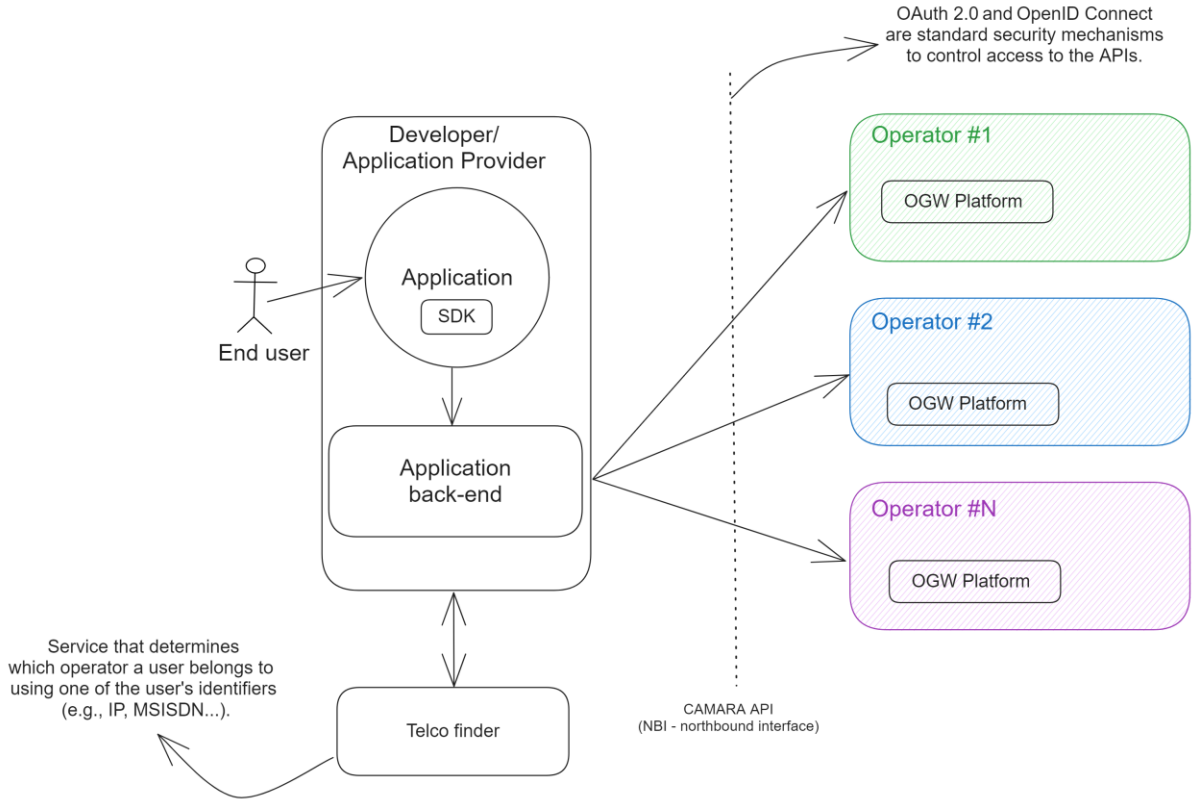

Figure 16: Direct Integration Developer – Operator (Multiple Operators)

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 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 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 API calls can be simplified.

4 Use cases and Operational User Stories

The Operator Platform as GSMA Open Gateway 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 includes a comprehensive set of Operation, Administration, and Management (OAM) capabilities that streamline API management, resource provisioning, and service delivery. These capabilities ensure the OGW platform supports critical business operations like onboarding Application Providers, managing 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 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.

4.1 Integration to the OGW platform

Channel Partners are an ideal go-to-market for Operators seeking to sell their 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 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 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.

4.2 Developer / Application Provider management

4.2.1 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.

4.2.2 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.

4.2.3 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.

4.2.4 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.

4.3 Application management

4.3.1 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.

4.3.2 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.

4.3.3 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.

4.3.4 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.

4.3.5 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.

4.3.6 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.

4.4 Order management

4.4.1 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.

4.4.2 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.

4.4.3 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.

4.5 Catalogue Management

4.5.1 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

4.5.2 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.

4.6 Usage Monitoring

4.6.1 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.

4.6.2 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.

4.7 Billing and Payment

4.7.1 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).

4.7.2 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.

4.7.3 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.

4.7.4 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.

4.7.5 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.

4.7.6 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.

4.7.7 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.

4.7.8 Historical Data and Reporting

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

4.7.9 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.

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                                      #
################################################################################
```

```
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

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.