PSI Markdown Administrative Decision Records

PSI MADR

Version:	MS11 [1.3.0]
Date:	2025-04-23
Reference:	PSI-MADR
Total Pages:	114

Table of Contents

List of Figures

List of Tables

1 Document Meta Information

1.1 Document Signature Table

	Name	Function	Company
Author	Christian Grubert	PSI Project Team	CGI
Author	Dafinka Srezoska	PSI Project Team	CGI
Author	Norbert Czeranka	Project Team	CGI
Author	Christine Glaesser	Liaison Manager	CGI
Author	Hendrik Oppenberg	Technical Officer	CGI
Approval	Rui Goncalves	Project Manager	SES
Approval	Wolfgang Robben	Project Manager	CGI
Checked	Pepijn Witte	Quality Assurance Manager	CGI

Table 1.1: Signature Table.

1.1.1 Document Change Record

1.1.1.1 Changes

Date	Version	author	message
2022-07-07	MS1	Wolfgang Robben	Initial version
2022-07-14	MS1.1	Wolfgang Robben	Updated to resolve RID comments
2022-09-30	MS2	Hendrik Oppenberg	Decisions taken for code patching and testing in MS2
2022-12-31	MS3	Christine Glaesser	Decisions taken for async workflows and cust. inquiry in MS3
2023-04-19	MS4	Wolfgang Robben	Decisions taken for beam handling, demonstrations, inventory API, overbooking and security in MS4
2023-07-27	MS5	Hendrik Oppenberg	Decision prepared for quality management, updated decision requirement trace, extracted roadmap

Date	Version	author	message
2023-07-18	MS5 pre	Wolfgang Robben	Decisions taken for transition to OpenAPI 3 and bundled group offering
2023-10-06	MS6	Christine Glaesser	Updated workflow for decision management, added requirement assumptions and descoping/implementation comments
2024-01-25	MS7	Christian Grubert	Descope logistics, elaborated on advanced billing, introduced overbooking, improved traceability
2024-09-11	MS8 [1.2.0]	Norbert Czeranka	Added chapters 3.27 and 3.78. Public domain release changes.
2024-12-09	MS9 [1.2.1]	Bela Mueller	Added mission API decisions and the mission templates.
2025-02-03	MS10 [1.2.2]	Wolfgang Robben	Updates for MS10 delivery
2025-04-23	MS11pre [1.2.3]	Wolfgang Robben	Advanced OAS patching, MEF convergence.

Table 1.2: DCR Table.

1.1.1.2 Source Control

Changes to this document are tracked electronically. No signature is required by the authors. The following information can prove the integrity of the document and reveal any change.

Repo	Date	Author	Branch	Hash

Table 1.3: GIT Changelog Table.

Figure 1.1: DCR QR-Code.

1.2 Documents

1.2.1 Reference Documents

Acronym	Reference	Title	Version
PSI-DL	PSI-DL	PSI CGI Document List	current MS (doc version)
PSI-RTM	PSI-RTM	PSI RTM - Requirements Traceability Matrix	see before
PSI-TAD	PSI-TAD	PSI Terms, Abbreviations and Definitions	see before
PSI-TOD	PSI-TOD	PSI Tasks and Operations Dictionary	see before

Table 1.4: Reference Documents

1.2.2 External Annexes

Reference	Title or Filename
PSI-ADR-Annex-I	PSI Markdown Administrative Decision Records - Annex I

Table 1.5: External Annexes

2 Introduction

The Pooling & Sharing Interfaces Definitions (PSID) project is an ESA co-funded effort to define a common standard for the interfaces of Pooling & Sharing Systems (PSS) for Satellite Communication (SatCom) services. A PSS is a digital platform for matchmaking (Gov)SatCom users' demands (both commercial and institutional) with (Gov)SatCom providers' offers. Bringing together multiple (Gov)SatCom providers in one platform makes the market transparent, thus allowing users to get an overview of the market and to compare different offers efficiently. Additionally, a PSS assists users with little knowledge about the (Gov)SatCom domain defining their requirements on the (Gov)SatCom services. Those two aspects combined allow for fast access to the services and an efficient usage of the available capacities. To accomplish this, a PSS steps in between the usual processes of finding a provider/supplier, requesting an offer, and ordering the desired products or services, either as a service broker or by pooling products and services from different providers and offering them as an intermediary or distributor. Subsequently, the PSS can be used to monitor the services and manage multiple missions in a single application.

Eventually, a PSS can also be used as (or manage) a community hub, i.e., a number of end users or customers with similar interest that *share* their common resources and utilize a commonly obtained *pool* of (Gov)SatCom capacities. This strategy increases the efficient usage of scarce resources further.

There are already different approaches on PSSs, that might lead to an unnecessary fragmentation of the market. Therefore, a common standard for the interfaces of a PSS is required to allow the interaction between those different PSSs and reduce the effort of (Gov)SatCom providers to offer their product and services via multiple PSSs to maximize their reach.

Such a standard needs to take care of the different interfaces involved in the aforementioned processes, i.e.,

The goal of this project is to mainly define aspect 1 and to develop a software mock-up as needed to validate the various interfaces being developed.

The PSI standard derives from the existing industry-standard "Open Digital Framework" of **TM Forum** alliance[1]. The "Open Digital Framework" is a reference framework for delivering online Information, Communications and Entertainment services to the telecom world. It empowers market participants to compete and cooperate. One of PSI's goals is to make this existing standard fit for the world of satellite communication.

The consortium for this project consists of the service & technology providers SES Techcom and CGI, as well as of the (Gov)SatCom operators SES, Hellas Sat, Hispasat, Hisdesat, and LuxGovSat, and Inmarsat being both a service & technology provider and a (Gov)SatCom operator.

[1] See https://www.tmforum.org/resources/reference/gb991-tm-forums-core-concepts-and-principles-v22-0-0/

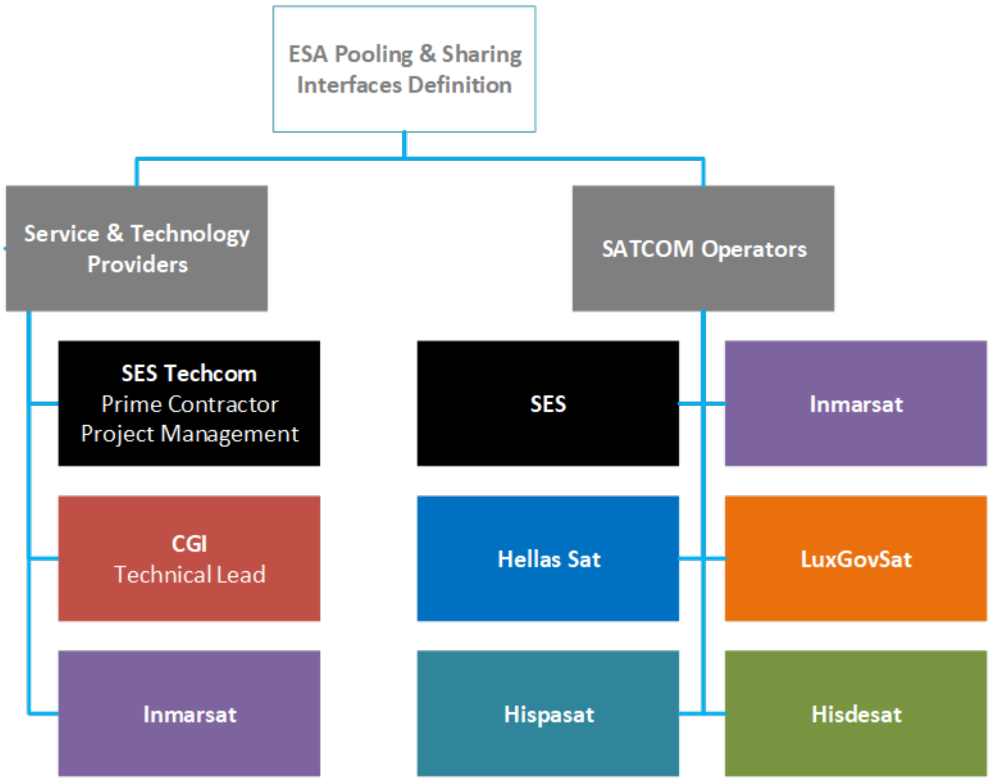


Figure 2.1: The PSI consortium.

2.1 Document Scope

This document provides a list of all architectural decision records (ADR). ADR is a lean templated approach to capture any decisions in a structured way. The template originated from capturing architectural decisions and developed to a template allowing to capture any decisions taken in project execution.

Decision may be related not only to technological or design aspects but also e.g. processes, workflows and management frameworks as well as tooling for verification and validation. Such decision will be e.g. the verification and validation approach.

This document will be updated in lockstep with the design progress to reflect all decision taken and will be final only with the last delivery. This document will be iterated/evolved in line with the agile project execution. Deprecated and superseded architectural decision records have been moved to [PSI-ADR-Annex-I] to improve the readability of the document.

The following sections heavily refer to terms, abbreviations and definitions defined in the [PSI-TAD].

2.1.1 Compiled Document

NOTE: THIS IS A COMPILED DOCUMENT [2]

This document has been compiled/generated from external sources and is not being written as-is. Therefore, any changes made within this compiled version of the document will be lost upon recompilation!

[2]Document compiled on 2025-04-23 12:37.

To make (permanent) changes, edit the respective sources directly or contact the PSID team.

2.1.2 Signature

Changes to this document are tracked electronically. No signature is required by the authors. The information in the "Source Control" chapter can prove the integrity of the document and reveal any change.

2.1.3 Development State

Current document version is `1.3.0`.

2.1.4 Release Notes

[[*TOC*]]

2.1.5 PSI Release Notes

2.1.5.1 Introduction

Welcome to the latest release of the Pooling and Sharing Interface (PSI) API!
This document outlines the new features, improvements, and important updates included in this version.

2.1.5.2 Key Highlights

The central focus of this release is the implementation of the **Mission Management ODA Blueprint**.
This component complements the mission-related APIs by providing a *reference implementation of graphical user interfaces* that help users specify their product and service requirements.

Designed with users in mind, this component uses templates to simplify mission creation and introduces a governance layer to facilitate and control the requirements collection process.
It's built as a standalone micro-frontend and can be easily integrated into existing OSS/BSS/PSS systems.

The interface includes multiple visualization modes:

Another major update in this release is the migration to **TM Forum APIs Version 5 (TMF5)**.
All APIs have been ported to the current TMF baseline.
However, TMF5 introduced some gaps in the Component Test Kit (CTK), resulting in partial test coverage for certain APIs. This limitation will be addressed once TM Forum updates the CTK.

Additionally, this release introduces **MEF-compatible APIs**, marking the beginning of convergence between MEF and TMF frameworks within PSI.
Our goal moving forward is to support both API standards in their respective areas.

2.1.5.3 What's New

2.1.5.3.1 Newly Added APIs

2.1.5.3.2 Updated APIs

2.1.5.3.3 Added Requirements

2.1.5.4 Known Limitations

2.1.5.5 Feedback and Contributions

We appreciate your input!
If you experience any issues or have suggestions, please don't hesitate to contact us.
We also encourage community contributions to help enhance PSI further.

2.1.6 Security Considerations

A security assessment is performed, if needed, on considered options to analyse their strengths and weaknesses from a security point of view. The assessment is focussing on the three most important concepts, *confidentiality*, *integrity* and *availability*.

Confidentiality is the property that information is not made available or disclosed to unauthorized entities. Data that is or is going to be protected is usually categorized according to the damage that could be done if this data was retrieved by an unauthorized entity. Usually, several methods can be implemented to ensure confidentiality, amongst those are:

Integrity refers to the maintaining of accuracy, consistency and trustworthiness of data during its lifecycle. To ensure integrity, several mitigations can be put in place:

Availability denotes the consistency and accessibility of data for authorized access. Availability includes the proper maintenance of hardware and technical infrastructure and can be increased by dedicated setups and implementations, for example:

The result of this assessment helps shape a decision, and is presented in a CIA (confidentiality, integrity, availability) table which compares all considered options.

3 List of Accepted Decisions

The following sections compile the list of management decisions that where accepted by the PSI team.

3.1 Use Java as the Backend Programming Language for the API

3.1.1 Context and Problem Statement

To implement the standardized PSI Mock-up, we need a programming language for the backend development.

3.1.2 Decision Drivers

3.1.3 Considered Options

3.1.4 Decision Outcome

Chosen option: “Java”, because compared to the other languages considered, wins in the majority of decision drivers. The most important of all is that is well known to the team and has excellent integration with both REST API and event-driven messaging, which is required for the development of the PSI Mock-up.

3.1.5 Compliance

Java will be installed in the developers environment and integrated in the setup of the PSI Mock-up.

3.1.6 Pros and Cons of the Options

3.1.6.1 Java

Pros:

Cons:

3.1.6.2 Python

Pros:

Cons:

3.1.6.3 Elixir

Elixir is a functional programming language and is an advancement of Erlang.

Pros:

Cons:

3.1.6.4 Go

Go (also called Golang or Go language) was developed by Google engineers to create dependable and efficient software. Most similarly modelled after C, Go is statically typed and explicit. It was designed by taking inspiration from the productivity and relative simplicity of Python, with the ability of C.

Pros:

Cons:

3.1.6.5 Ruby

Pros:

Cons:

3.2 Use Gradle Build Tool

Technical Story: PSI-1

3.2.1 Context and Problem Statement

The PSI prototype implementation will be using Java with Spring. We need a build tool to compile the source files to an executable.

3.2.2 Decision Drivers

3.2.3 Considered Options

3.2.4 Decision Outcome

Chosen option: "Gradle", because it has quicker recompile time than Maven. The easier extendability of Gradle could have a positive impact in the future, though it is not required now. Both compared tools are equal in other decision drivers.

3.2.5 Compliance

The prototype will be set up with Gradle as soon as the first component breakdown is done.

3.3 Use Spring Framework

3.3.1 Context and Problem Statement

The PSI prototype application will be programmed in Java. A platform is required that provides comprehensive infrastructure support for developing Java applications.

3.3.2 Decision Drivers

3.3.3 Considered Options

3.3.4 Decision Outcome

Chosen option: “Spring”, because is the oldest and most popular framework for Java. It utilises runtime Dependency Injection (DI) and Aspect-Oriented Programming (AOP) to enable highly flexible applications. There are a lot of existing extensions for web, cloud, security and other aspects. One of the extensions is Spring Boot, which features “convention over configuration” to remove the XML configurations needed before. Spring has a big community, it is a framework that is well known by the team which limits the learning time.

The other compared tools have fewer extensions, small community support and slow evolving standard.

3.3.5 Compliance

Add Spring dependencies in Gradle and necessary configuration when setting up the PSI prototype application.

3.3.6 Pros and Cons of the Options

3.3.6.1 Spring

Spring (cf. https://spring.io/) is the oldest and most popular of the compared frameworks. It utilises runtime Dependency Injection (DI) and Aspect Oriented Programming (AOP) to enable highly flexible applications. There are a lot of existing extensions for web, cloud, security and other aspects. One of the extensions is Spring Boot, which features “convention over configuration” to remove the XML configurations needed before. The extensive documentation and the “starters” enable an easy beginning, though low-level customisations can get complex.

3.3.6.1.1 Upsides

3.3.6.1.2 Downsides

3.3.6.2 Microprofile

Microprofile is a standard API for Java microservices. It is implemented by multiple products, where “Quarkus” (cf. https://quarkus.io/) by JBoss, Red Hat and IBM is the most popular one. Similar to Spring it offers Dependency Injection (DI), but does this at compile time for faster startup and lower container sizes. This also results in the possibility to create native images, which are even faster at startup. The extension ecosystem and community is smaller, but growing.

3.3.6.2.1 Upsides

3.3.6.2.2 Downsides

3.3.6.3 Dropwizard

Dropwizard (cf. https://www.dropwizard.io/) is a set of fixed libraries, which are curated and tested to be compatible. This results in a well-defined and compact environment for web development, but has drawbacks when other features are needed.

3.3.6.3.1 Upsides

3.3.6.3.2 Downsides

3.4 Use TM Forum Open APIs and SID

3.4.1 Context and Problem Statement

The Pooling and Sharing Interfaces project is based on the frameworks of TM Forum. It should be decided whether the Information Framework (SID) or the Open APIs will be used as a guideline for the implementation of the PSI data model.

Also, the implications on the licensing due to the usage of TM Forum should be considered.

3.4.2 Decision Drivers

3.4.3 Considered Options

3.4.4 Decision Outcome

Both TM Forum frameworks will be used for the PSI project, but with a different focus. For the development of the standardized interfaces, Open APIs will be used. Nevertheless, the Information Framework (SID) will be used to supplement the documentation, especially regarding the TAD.

3.4.5 Compliance

The standardized Pooling & Sharing Interfaces will follow a tailored version of the TM Forum Open APIs and data models to match the satellite communication context. The TAD refers to the ABEs of the Information Framework (SID), including some addendums to show the actual scope (excluding what is not required for this project) and to add some domain-specific information.

3.4.5.1 TM Forum SID

The Information Framework (SID) provides a common data model and data dictionary for implementing business processes in the telecommunications sector. To categorize business entities it contains the concept of domains e.g. Customer, Product, Service, Resource, Common, etc. Within each domain, further partitioning is achieved by identifying Aggregate Business Entities (ABEs). To understand the business perspective, these entities contain textual descriptions, properties and relationships depicted via diagrams.

Based on the available documentation, Eclipse Papyrus was used in attempts to generate entity and DTO classes from the data model. These were to be used to define standard-compatible request and response bodies of the data exchange between different PSS and providers. However, due to the vague description of how to generate the classes and the deep hierarchical levels of the model, the code generation was not successful.

3.4.5.2 TM Forum Open APIs

The TM Forum Open APIs were developed to meet the challenges of increasingly complex multipartner digital services and as a response to the rise of Microservices. Their data model is strongly based on the SID. The differences that exist between the two occur mostly because the SID has a deep inheritance structure which is difficult to apply in a REST API model. Therefore, the Open APIs flatten SID's data model which enables the usage of single concrete classes. This additionally relaxes the use of derived fields from the parent levels in the inheritance hierarchy and enables descoping of unused entities.

Unlike the SID, the data model of TM Forum Open APIs is available on GitHub and can be easily generated with the help of different tools such as:

The TM Forum Open APIs are licensed under Apache 2.0 License terms and conditions. According to the Apache Licensing FAQ page, if an Apache 2.0 licensed code is used and modified, the result can be used commercially, and can be distributed under a different license, but one needs to acknowledge the use of the Foundation's software e.g. by preserving the license text and any NOTICE files.

3.4.6 Scope

The influence of this document on the project's scope is significant. It outlines the decision to adopt TM Forum's frameworks, specifically the Open APIs and the Information Framework (SID), for the PSI project. The document highlights that while both frameworks will be utilized, they will serve different purposes. The Open APIs will be the primary focus for developing standardized interfaces, ensuring ease of integration and alignment with industry standards. The SID will supplement the documentation, particularly for the Technical Architecture Document (TAD). The document also emphasizes the benefits of using TM Forum's standardized interfaces, such as minimal adjustments needed for common business processes and the facilitation of integration between PSSs and service providers. Additionally, it underscores the importance of adhering to widely accepted industry standards to simplify the integration of terrestrial telecommunication services. The holistic approach of TM Forum's framework, with its comprehensive Open API component, is deemed highly applicable to the project's needs. This strategic decision is expected to streamline the development process, enhance interoperability, and ensure compliance with industry best practices.

3.4.7 Links

TM Forum Open API Licensing
Apache Licensing FAQ

3.5 Database for Prototype

3.5.1 Context and Problem Statement

The prototype should be able to do simple data storage to provide better simulation of a real PSS. For example, this will allow to actually create a resource specification and retrieve exactly this specification (instead of a dummy). To implement this we want to use a simple database backend with as less overhead as possible.

3.5.2 Decision Drivers

3.5.3 Considered Options

3.5.4 Decision Outcome

Chosen option: “MongoDB”, because

The simple structure allowed the team to adapt rather quickly, despite having nearly no previous knowledge. First CRUD instructions were already done while researching and worked immediately.

3.5.5 Compliance

The MongoDB connector for spring is included in the prototype and used for the first endpoints. Further usage is ensured by code review.

3.5.6 Pros and Cons of the Options

Capability	Postgres	MariaDB	H2	MongoDB	CouchDB
Data structure	Relational	Relational	Relational	Object-Oriented	Object-Oriented
Docker	Yes	Yes	No	Yes	Yes
Embedded	No	No	Yes	Yes	No
Spring Integration	Yes	Yes	Yes	Yes	3rd-Party
Community Support	High	High	High	High	Mediocre
Team Knowledge	Yes	Yes	Yes	No	No

Table 3.1: Decision-matrix for the prototype database

3.6 Use GitLab Flow as Branching Concept

3.6.1 Context and Problem Statement

We have to choose an appropriate branching concept for our repository.

3.6.1.1 What Is a Branching Concept

Branches are primarily used as a means for teams to develop features giving them a separate workspace for their code. These branches are usually merged back to a master branch upon completion of work. This way, features (and bug fixes) are kept separate from each other, making it easier to fix mistakes. This means that branches protect the main body of code and changes made to any given branch don’t affect other developers.

A branching strategy, therefore, is the strategy that software development teams adopt when writing, merging and deploying code when using a version control system. It is essentially a set of rules that developers follow to define how they interact with a shared codebase.

3.6.2 Decision Drivers

3.6.3 Considered Options

3.6.4 Decision Outcome

Chosen *slightly modified* GitLab Flow.

While being relatively simple, it still allows tracking releases and work on them while progressing development. This work can be picked back to development branches. The main branch is utilized to contain deployment-ready code, so code is merged into the production branch when it's ready to be released.

The modification lies in the nature of PSI not having real releases. We are not delivering a solution. Therefore, we usually treat release-branches to be **delivery branches**. In PSI, we have fixed release cycles. When approaching a release cycle, feature-branches should be merged only when they are not affecting the deliverability of the main branch.

The release branches are created in front of a delivery and are used to track a release and the RIDs raised on them.

With this modification, the strategy is a chimera between TrunkBased and GitLab workflow.

3.6.4.1 Maintainer Permissions

Together with this decision, both key developers are granted maintainer access, so they can merge on their own and enforce the strategy among other teammates.

Peer-review should be performed before merge (assured by the reviewed-by feature of GitLab). In rare, exceptional cases, peer-review can be omitted. This just be justified by a comment in the merge request.

Direct push to the main branch is strictly prohibited and only allowed for AIV activities like e.g. fixing a stuck or broken pipeline.

3.6.5 Compliance

Apply the new workflow from **SPRINT S-06 22CW28-30** on.

3.6.6 Pros and Cons of the Options

3.6.6.1 GitFlow

The main idea behind the Git flow branching strategy is to isolate your work into different types of branches. There are five different branch types in total:

The two primary branches in Git flow are main and develop. There are three types of supporting branches with different intended purposes: feature, release, and hotfix.

3.6.6.1.1 Pro

3.6.6.1.2 Con

3.6.6.2 GitHub Flow

The GitHub flow branching strategy is a relatively simple workflow that allows smaller teams, or smaller applications/products that don't require supporting multiple versions, to expedite their work. In GitHub flow, the main branch contains your production-ready code. The other branches, feature branches, should contain work on new features and bug fixes and will be merged back into the main branch when the work is finished and properly reviewed.

3.6.6.2.1 Pro

3.6.6.2.2 Con

3.6.6.3 GitLab Flow

At its core, the GitLab flow branching strategy is a clearly defined **workflow**. While being quite similar to the GitHub flow branch strategy, the main differentiator is the addition of environment branches, i.e. "production" or "release" branches, depending on the situation. In PSI, these "environments" e.g. can be the AQA (agile QA) review or a delivery ("release").

Just as in the other two Git branch strategies, GitLab flow has a main branch that contains code that is ready to be deployed. However, this code is not the source of truth for releases. In GitLab flow, the feature branch contains work for new features and bug fixes which will be merged back into the main branch when they're finished, reviewed, and approved. For each release the team will create a release branch to review and approve it separately, too.

3.6.6.3.1 Pro

3.6.6.3.2 Con

3.7 Party Management API

3.7.1 Context and Problem Statement

Business interactions are executed between two or more parties, which are either natural persons ("individuals") or organizations. A party can have different roles in different interactions, for example, a service provider can also be a customer or business partner of another provider.

Therefore, PSI should support the registration of parties and their roles in the different interactions. Currently, TM Forum provides two Open APIs for managing these: Party Management API and Party Role Management API.

It should be decided whether implementing the Party Management API is sufficient for the use cases covered for PSI, or if the Party Role Management API is additionally required.

3.7.2 Decision Drivers

3.7.3 Considered Options

3.7.4 Decision Outcome

It has been decided that the Party Management API defines the roles of parties in a simpler way than the Party Role Management API. At the current stage of the project, no benefits have been identified for implementing additionally the Party Role Management API, as it will introduce unnecessary complexity.

Note: This decision is not final and if at a later stage of the project the need emerges to support the Party Role Management API, the decision could be changed.

3.7.5 Compliance

Implement the Party Management API in the PSI Mock-up as part of the referenced technical story, and support the implementation with AITF tests.

3.7.6 Pros and Cons of the Options

3.7.6.1 Party Management API

The Party Management API identifies two parties: individuals and organizations. Each of these parties have a separate API definition for the minimal CRUD operations.

In addition to the basic information that can be provided for the parties, the contact medium, credit rating, and the tax exception certificate can be given. Both individual and organization can specify a list of related parties. For each of the related parties, the role can be provided in String format which is flexible enough to take any value. A related party can be a reference to another individual or organization, without any limitations.

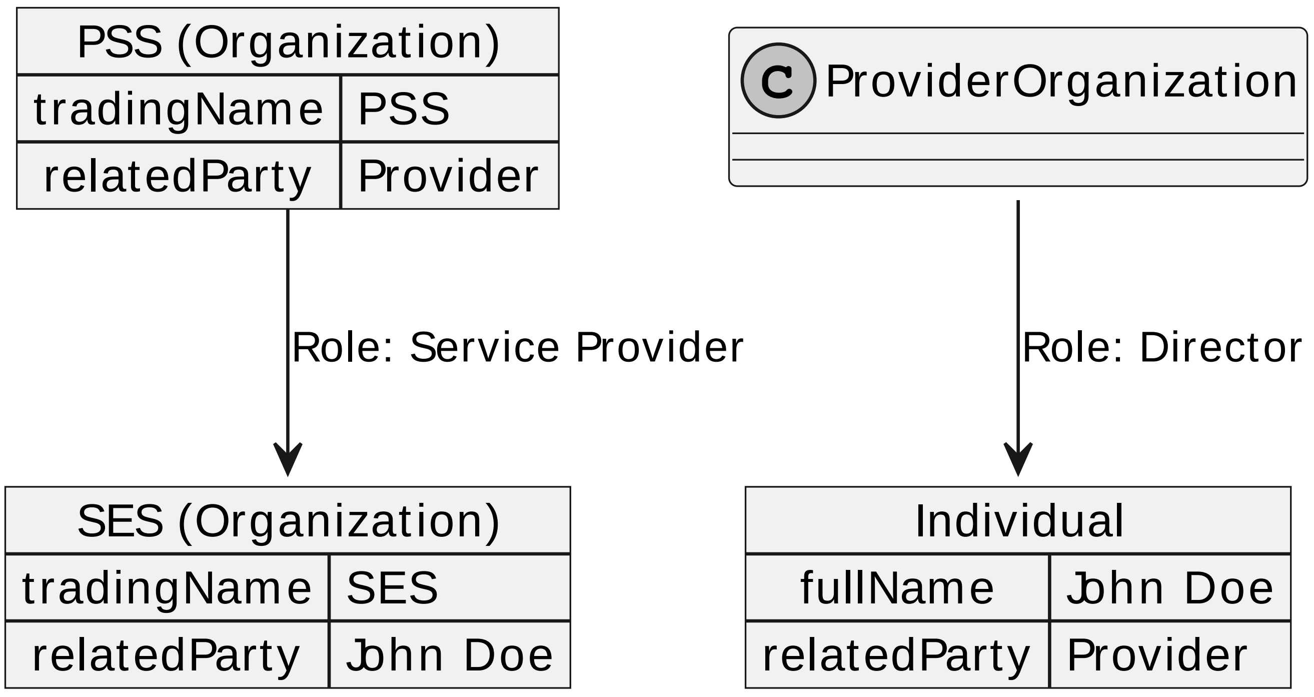


Figure 3.1: Party Management.

3.7.6.2 Party Role Management API

The Party Role Management API defines an entity named Party Role which assumes the use of the following APIs:

The Party Role entity has a mandatory Role Type property. In addition to the name of the role, the Role Type is a more complex entity containing information such as partnershipName, partnershipHref and partnershipId.

Similar to the Party Management API, the Party Role has a property for related parties where references to the Party Management API entries can be provided along with their role.

3.7.7 Implications for the Scope

The restriction to any number of hierarchical layers is dependent on the PSS implementation, and the interface only enables the hierarchical structure. Additionally, An agreement on visibility as implied by the leasing of SLA parameters requires additional parameters that this API cannot supply, making it unsuitable for the intended purpose.

However, the Party Management API enables the definition of a hierarchical structure, which allows a user organization to declare its own community of users with a maximum of three hierarchical layers.

3.8 Asynchronous Workflow Approach

3.8.1 Context and Problem Statement

Some processes between a PSS and a provider (or PSS and PSS), such as customer inquiries and orders, can take longer time to complete. For example, when a customer inquiry is created, the provider may require significant time to process and respond with an adequate product offering. Or, when a product order is placed by a customer, it can take hours to days for its state to change, e.g. from ‘inProgress’ to ‘completed’.

Additionally, while the provider processes the order, the customer may request an update to an order item, which the PSS should then propagate further to the provider. Once the provider completes the order, they should inform the PSS of the state change, and then the PSS should send a notification to the customer. This problem is perfect for publish-subscribe communication model addressed by message queues such as Apache Kafka, RabbitMQ, etc. (cf. chapter *Use RabbitMQ Message Broker* in [PSI-MADR-Annex-I])

All these processes require an asynchronous and event-driven type of communication. This kind of communication categorizes as **stateful**. The word *state* in this case refers to the condition or attributes of being at a given point in time. The RESTful APIs instead are *stateless* and can’t provide this kind of behaviour by direct means. In the RESTful synchronous approach, a client sends an HTTP request and expects a direct (prompt) response from the server. By being stateless, no client information is stored, and each request is separate and unconnected. One way would be for the client to continuously send requests to the server to check the state of the delayed response. This pattern is called *busy-waiting* or *polling*, but it is a poor solution. Besides additional network traffic, it would introduce a constant load on the server and waste of resources. It makes also hard to protect the servers against Denial-Of-Service-attacks as high-frequency polling is hard to differentiate from malicious requests.

Therefore, the aim of this decision record is to find a better approach to handle the above-mentioned asynchronous scenarios and the propagation of state changes between PSS, providers and customers.

3.8.2 Decision Drivers

3.8.3 Considered Options

3.8.4 Decision Outcome

Chosen option: *TM Forum Event Management API*, as it satisfies the decision drivers the most and represents the latest approach provided by TM Forum for handling asynchronous communication.

3.8.5 Compliance

3.8.6 Pros and Cons of the Options

3.8.6.1 Plain REST

The first and most straightforward option considered for handling asynchronous requests is the use of plain REST. It means that once the HTTP request is sent by a client, the server responds immediately with some HTTP response.

In the diagram below, an order process is depicted using this approach. The normal (solid) arrows show the HTTP requests, while the dashed arrows show the HTTP responses.

The customer places an order to the PSS with the POST method. The PSS saves the order in its data store with a unique identifier (ID) and 'pending' state. It directly sends a response back to the customer with the created order and issues a PUT request to the provider with that same order. The provider also saves the new order in their data store with the ID provided by the PSS and responds back to the PSS that the order has been created with a 'pending' state.

After some time, the provider issues a PATCH request to the PSS with the new 'acknowledged' state of the order. The PSS updates its data store record, sends a response back to the provider with the 'acknowledged' state of the order and notifies the customer.

Sometime later, the customer decides to cancel the order hence issuing a PATCH request to the PSS. The PSS directly sends a response back with the 'cancelled' state of the order to the customer and forwards the PATCH request to the provider. The provider responds back promptly that the state of the order has been updated.

One big downside of this option is that the states can easily go out of sync if one side of the communication is down/unreachable. The system that was not available cannot know with certainty which messages were lost, therefore they need to request the full state from the other party in order to re-sync. For example, they will need to implement a complex mechanism to repeat requests after re-establishing communication, restarting, failover, etc.

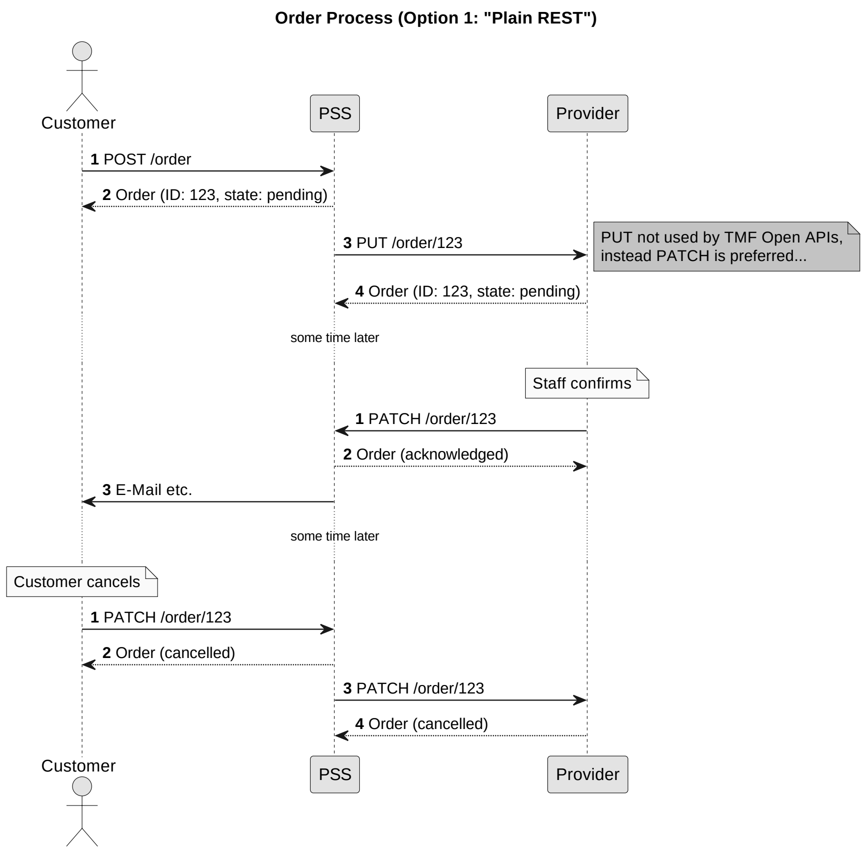


Figure 3.2: Plain REST.

3.8.6.2 TM Forum API Hub

The TM Forum's team has foreseen the need for asynchronous communication, therefore one of the approaches they offer for such requests is the publish-subscribe pattern for REST-based APIs that require eventing.

How this should be implemented is detailed in the REST API Design Guidelines document.

In summary, all the Open APIs of TM Forum contain an endpoint dedicated to register/unregister callback listeners (api/hub) and a listener endpoint where the events are published. Firstly, the listener subscribes to the /api/hub to listen to events. Then, when the state of a product order changes, a ProductOrderStateChangeEvent is raised by the system and all registered listeners in the /api/hub are notified. It is also possible to subscribe only to particular events if a subscription query is provided.
Finally, the listener can unregister from the /api/hub if they should no longer receive event notifications.

The ordering process of PSI using the TM Forum's API Hub is depicted in the diagram below.

The PSS implements the listener endpoint (e.g. `https://pss.com/ordersApi/listener`), while the provider implements the API Hub (e.g. `https://provider.com/ordersApi/hub`). The difference compared to the first diagram is that the PSS can not share the ID of the order with the provider, so the same order is saved under a different ID in the provider's data store. Then, the PSS registers its callback listener to the provider's hub towards being notified of order updates. As soon as the order is processed by the provider, a `ProductOrderStateChangeEvent` is sent to the registered listener of the PSS.

Note that the notification process works *one way* in this example, from provider to PSS. It is not possible to communicate in the inverse direction, because the provider does not have visibility to the order ID in the PSS's data store, hence can't establish a notification subscription. Therefore, the order cancellation works similar to the first option with plain REST, because there is no other way for the PSS to notify the provider.

In consequence, we differentiate between calls that are **expected** to consume time and calls that are expected to be processed instantaneously.

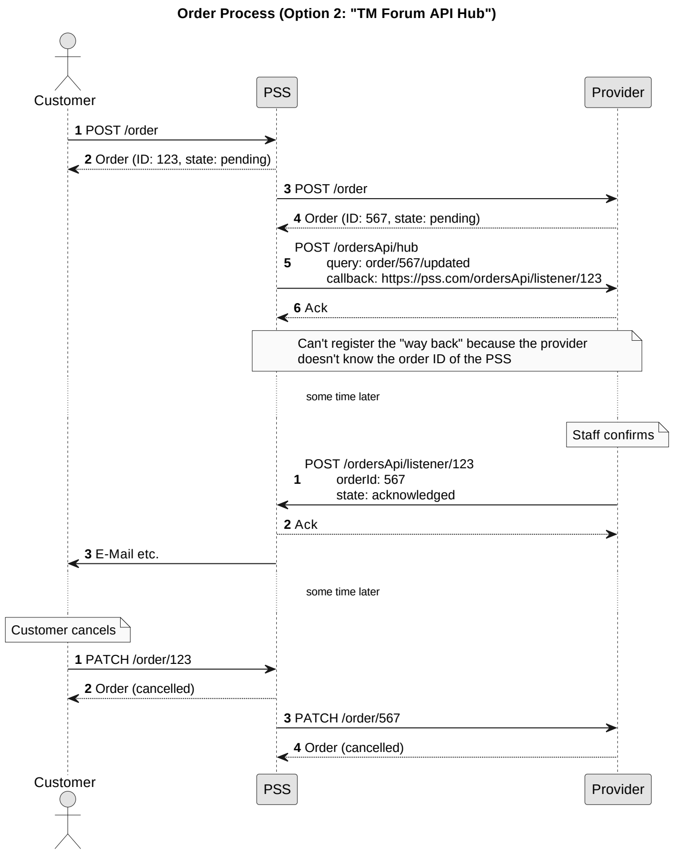

Order Process (Option 2: "TM Forum API Hub")
Customer
PSS
Provider
1 POST /order
2 Order (ID: 123, state: pending)
3 POST /order
4 Order (ID: 567, state: pending)
5 POST /ordersApi/hub query: order/567/updated callback: https://pss.com/ordersApi/listener/123
6 Ack
Can't register the "way back" because the provider doesn't know the order ID of the PSS
some time later
Staff confirms
1 POST /ordersApi/listener/123 orderId: 567 state: acknowledged
2 Ack
3 E-Mail etc.
some time later
Customer cancels
1 PATCH /order/123
2 Order (cancelled)
3 PATCH /order/567
4 Order (cancelled)
Customer
PSS
Provider

Figure 3.3: TM Forum API Hub.

3.8.6.3 Message Queues

This approach envisions that on both sides (PSS and provider) there is a message queue running, such as *Apache Kafka* or *RabbitMQ*. These are well-known complete solutions to asynchronous message exchanges with configuration options for fault tolerance and message acknowledgement already in place. For more details on message queues, refer to chapter *Use RabbitMQ Message Broker* in [PSI-MADR-Annex-I].

The diagram below shows an example setup for the ordering process. What can be spotted is that the asynchronous message exchanges don't make use of REST APIs. Only the initial POST request for the order creation from the customer to the PSS is done with help of REST endpoint and TM Forum's standard, while everything else is pure Apache Kafka/RabbitMQ message exchange.

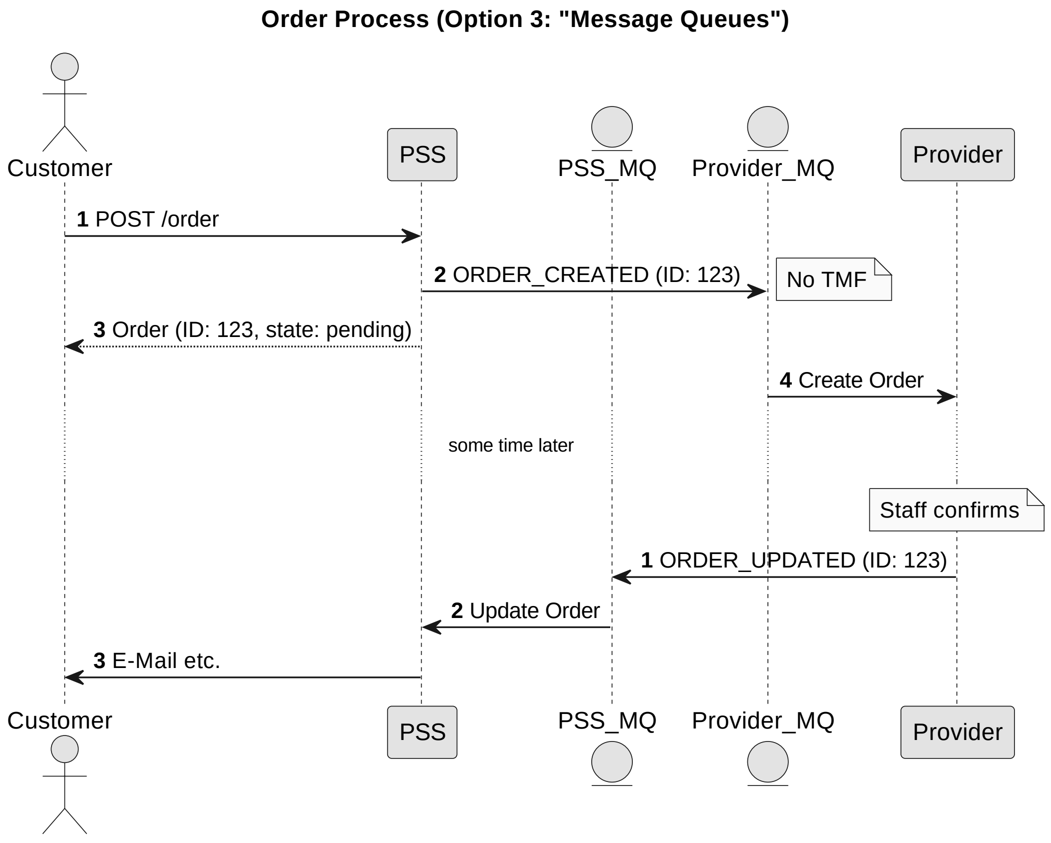

Order Process (Option 3: "Message Queues")
Customer
PSS
PSS_MQ
Provider_MQ
Provider
1 POST /order
2 ORDER_CREATED (ID: 123)
No TMF
3 Order (ID: 123, state: pending)
4 Create Order
some time later
Staff confirms
1 ORDER_UPDATED (ID: 123)
2 Update Order
3 E-Mail etc.

Figure 3.4: Message Queues.

3.8.6.4 TM Forum Event Management API

The last considered option for the asynchronous workflow approach for PSI is the new TM Forum's *Event Management API* (TMF688). Similar to how they implement the other Open APIs of TM Forum, the PSS and the provider will implement the Event Management API in addition.

That involves the definition of an endpoint for a Hub, to which callback listeners will be registered, and an endpoint of a client listener where all the events will be received. It very much resembles the TM Forum's API Hub concepts introduced in Option 2, but with the advantage that they will not be implemented for each individual API separately. This is possible because the Event Management API has defined a **common event model**, as the base for all events, which can be extended to more specific event types such as `Order Create Event`, `Inquiry State Update Event`, etc. Additionally, it supports the *events streaming concept* by introducing topics to organize events coming from different domains. That said, events can be streamed to separate topics for resources, services, products, etc. The Event Management API has been developed specifically to work with a message queue which is hidden behind the listener endpoint. This enables providers and PSSs to freely choose a message queue implementation (e.g. Apache Kafka, RabbitMQ, etc.) as long as its API fulfils the requirements. In consequence, potential existing solutions may be re-used.

The following diagram depicts one such sample implementation for the ordering process. Take note that both PSS and Provider implement the Event Management API, ensuring the state updates will be propagated in both directions: from the provider to the PSS and from the PSS to the provider. The provider and the PSS have message queues running, hidden behind the implementation of the REST endpoints of the Event Management API. When an order is placed by the customer, it first hits the REST endpoints of the Product Order API of the PSS, which then translates the HTTP request to an appropriate event and passes it to the underlying message queue towards being processed. Then, the Event Management API of the PSS serves as an interface to notify the event to the registered callback listener of the provider via a dedicated REST endpoint. The Event Management API of the provider receives the event notification and forwards it to its message queue for processing.

The main challenge, which is also a problem for the other considered options as well, is that the provider and the PSS have to implement event publishing properly. That is, they have to ensure that the events they publish end up at the correct destination. For example, a PSS should inform the *affected* provider about order state updates only - not all providers.

For this scenario, it is envisioned that both PSS and provider use the same ID for an order. If for security reasons the PSS cannot share its internal order ID, it has to generate and maintain an anonymous external order ID which will be used for the message exchange with the provider.

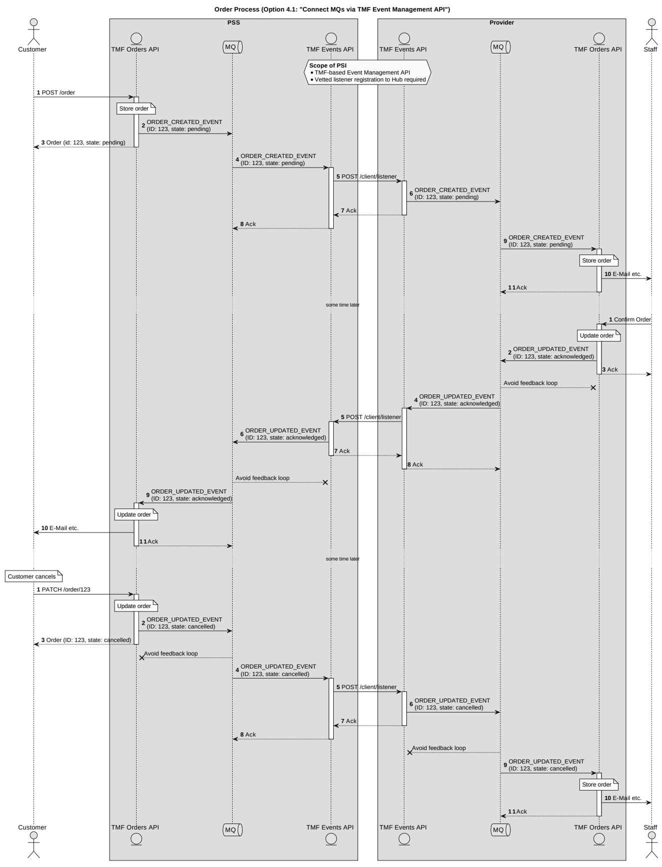

Order Process (Option 4.1: "Connect MQs via TMF Event Management API")
PSS
Provider
Customer
TMF Orders API
MQ
TMF Events API
TMF Events API
MQ
TMF Orders API
Staff
Scope of PSI
• TMF-based Event Management API
• Vetted listener registration to Hub required
1 POST /order
Store order
2 ORDER_CREATED_EVENT (ID: 123, state: pending)
3 Order (id: 123, state: pending)
4 ORDER_CREATED_EVENT (ID: 123, state: pending)
5 POST /client/listener
6 ORDER_CREATED_EVENT (ID: 123, state: pending)
7 Ack
8 Ack
9 ORDER_CREATED_EVENT (ID: 123, state: pending)
Store order
10 E-Mail etc.
11 Ack
some time later
1 Confirm Order
Update order
2 ORDER_UPDATED_EVENT (ID: 123, state: acknowledged)
3 Ack
Avoid feedback loop
4 ORDER_UPDATED_EVENT (ID: 123, state: acknowledged)
5 POST /client/listener
6 ORDER_UPDATED_EVENT (ID: 123, state: acknowledged)
7 Ack
8 Ack
Avoid feedback loop
9 ORDER_UPDATED_EVENT (ID: 123, state: acknowledged)
Update order
10 E-Mail etc.
11 Ack
some time later
Customer cancels
1 PATCH /order/123
Update order
2 ORDER_UPDATED_EVENT (ID: 123, state: cancelled)
3 Order (ID: 123, state: cancelled)
Avoid feedback loop
4 ORDER_UPDATED_EVENT (ID: 123, state: cancelled)
5 POST /client/listener
6 ORDER_UPDATED_EVENT (ID: 123, state: cancelled)
7 Ack
8 Ack
Avoid feedback loop
9 ORDER_UPDATED_EVENT (ID: 123, state: cancelled)
Store order
10 E-Mail etc.
11 Ack

Figure 3.5: TM Forum Event Management API.

The design *enables and encourages* the use of message queues, though they are *not required* on any side. It enables the re-use of existing message queue systems. The following diagram shows a black box view of the PSS and provider system to demonstrate a more simplistic implementation by using monolithic applications behind the Event Management APIs. Note that the REST endpoints are identical to the example above.

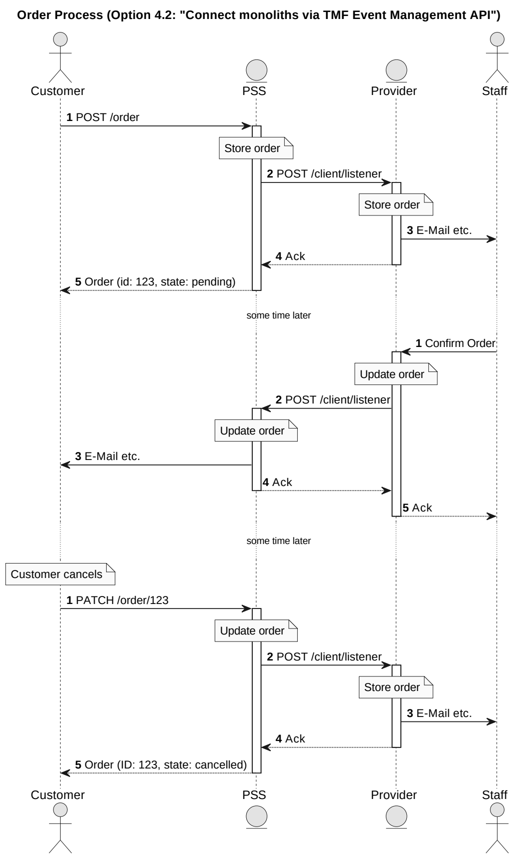


Figure 3.6: TM Forum Event Management API with monoliths.

Implementing the Event Management API will not have implications on the other APIs that are developed as part of the PSI project. The exchange of events should be considered for asynchronous communication scenarios only such as customer inquiries and orders. Nevertheless, a best practice in what scenarios event exchange should be preferred over synchronous calls is still undefined. This will be an area of research in the course of the project.

3.8.6.5 WebSockets / STOMP

The general message flow is identical to the Event API option described above, but instead of transmitting the events via independent HTTP calls it uses a permanent, bidirectional WebSocket connection: The client application initiates an HTTP request to the server and marks it to be “upgraded” to a WebSocket, which is supposed to be accepted. After that, both sides can send data at any time without additional handshakes, which enables real-time transport of huge amounts of data (compared to plain HTTP).

The server-client-model has some implications on the API and implementation:

Since WebSockets are designed to work exactly like arbitrary TCP/IP sockets (except for the initialisation), the data format for the exchange is not defined generally. A reasonable implementation is to use the *Simple Text Oriented Messaging Protocol* (STOMP[3]) protocol, which uses JSON to transport information between message brokers. This allows the reuse of our existing data structure without adjustments.

Another solution is the MBWS subprotocol[4]. The RFC drafts solutions to some problems, especially the connection recovery. Since it was never finalized and no implementation was found, this is also considered a sign of risk for

[3]https://stomp.github.io/
[4]https://datatracker.ietf.org/doc/html/draft-hapner-hybi-messagebroker-subprotocol-03

using WebSockets for this purpose.

The following diagram demonstrates the overall sequence:

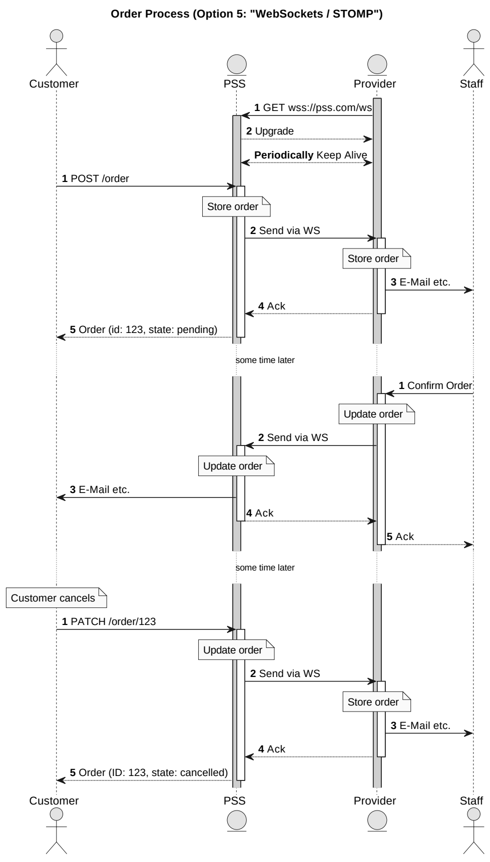


Figure 3.7: WebSocket API.

3.8.7 Security Considerations

3.8.7.1 Plain REST (Option 1)

This option is highly prone to be a victim of the integrity issue. Same order IDs are being used in the left-hand side, i.e. between the Customer and PSS, and in the right-hand side, i.e. between PSS and the Provider, of the process flow shown in the option. If an attacker is able to obtain the correct order ID, he can leverage this information to manipulate the right-hand side of the process flow.

This option is also highly prone to be a victim of the availability issue. The reason is that order states can easily become asynchronous. To prevent this, full state requests need to be considered in the implementation and need to be handled in a special manner. So, this special consideration could also be used as a door opener and exploited by the attackers to cause DoS attacks.

3.8.7.2 TM Forum API Hub (Option 2)

This option is lowly prone to be a victim of the integrity issue. The reason is that different order IDs are being used in both left- and right-hand side of the process flow. This means that, even if an attacker could be able to obtain the order ID from either side of the traffic flow, he cannot use that information directly owing to the fact that there is not a direct relationship between the order IDs of both sides.

However, this option is highly prone to be a victim of the availability issue because of the same reason mentioned in Plain REST option.

3.8.7.3 Message Queues (Option 3)

This option is highly prone to be a victim of the integrity issue due to the same reason mentioned in Plain REST option.

On the other hand, this option is lowly prone to be a victim of the availability issue, thanks to the nature of the message queues, i.e. MQs. As explained above, MQs have internal mechanisms to handle the synchronisation problems. Therefore, requesting full states is not needed, preventing the root cause of the DoS problem.

3.8.7.4 TM Forum Event Management API (Option 4)

This option is highly prone to be a victim of the integrity issue due to the same reason mentioned in Plain REST option. Nevertheless, if this option would be implemented to use different order IDs, the feasibility will be decreased to a low level. As it is found in the option description above, the use of different order IDs use could be implemented for this option.

This option is lowly prone to be a victim of the availability issue since it uses MQs. The positive effects of the MQs in terms of the availability issue is described in Message Queues option above.

3.8.7.5 WebSockets / STOMP (Option 5)

Although this option could not be directly compared within the same integrity issue context, it is assessed that this option is prone to be a victim of a similar integrity issue at medium level. The reason is, provider URLs are understood to be unique, which increases the complexity of a possible integrity-related attack from the attacker perspective.

Although this option could not be directly compared within the same availability issue context either, it is assessed that this option is prone to be a victim of a similar availability issue at medium level. The reason is that, the implementation of the fault tolerance was not found straightforward, which could expose the mechanism to DoS attacks due to the ineffective implementation of the option.

3.8.8 CIA Comparison for all Given Options

Below, a table is given summarizing the probability scores of each option regarding the CIA triad. As explained above, a three-level scale was used for scoring.

Option #	Confidentiality	Integrity	Availability
1	Low	High	High
2	Low	Low	High
3	Low	High	Low

Option #	Confidentiality	Integrity	Availability
4	Low	High (Low)	Low
5	Low	Medium	Medium

Table 3.2: CIA table for different asynchronous implementations.

3.9 Customer Inquiry Message Sequence

3.9.1 Context and Problem Statement

In the previous decision 2022-10-21 Asynchronous Workflow Approach we decided how to handle events with focus on the order management. The second API which requires asynchronous messages is the customer inquiry management, which was defined from scratch because of missing patterns in TM Forum. This leads to the need of defining the message flow between PSS and provider. Since this communication is not 1-to-1, the conditions are slightly but essentially different and require a new decision on the data exchange.

3.9.2 Decision Drivers

3.9.3 Considered Options

3.9.4 Decision Outcome

Chosen option: Common inquiry with synchronisation, because it is the more consistent option. The additional filtering of event data is considered a minor effort.

3.9.5 Compliance

Requirements are documented in PSI-REQ and PSI-TOD. An example implementation (without filtering) will be added to the mock-up.

3.9.6 Pros and Cons of the Options

3.9.6.1 Distinct Inquiry per System

In this variant, every system creates a distinct inquiry. The user creates an inquiry in the PSS, and the PSS creates a new inquiry (with different IDs and potentially different data) in every single provider system. Figure 3.8 shows the inquiries with ID 1 for the PSS and 2/3 for the providers. The initial one is created by the user, while the others are created by the PSS, which has to keep track of this association. When the inquiry is processed by the provider, the system sends an “updated”-event to the PSS. After the last provider has set the state to “completed”, the PSS can update its own state and inform the user. Depending on the implementation, it can also directly fetch and aggregate the results of the provider or wait until the user wants to view it.

Note the option of storing the association as part of the actual inquiry data structure was discussed after doing that in option 2 (Common inquiry with synchronisation). We found that this does not add much additional value to this particular setup and deliberately decided not to change this description.

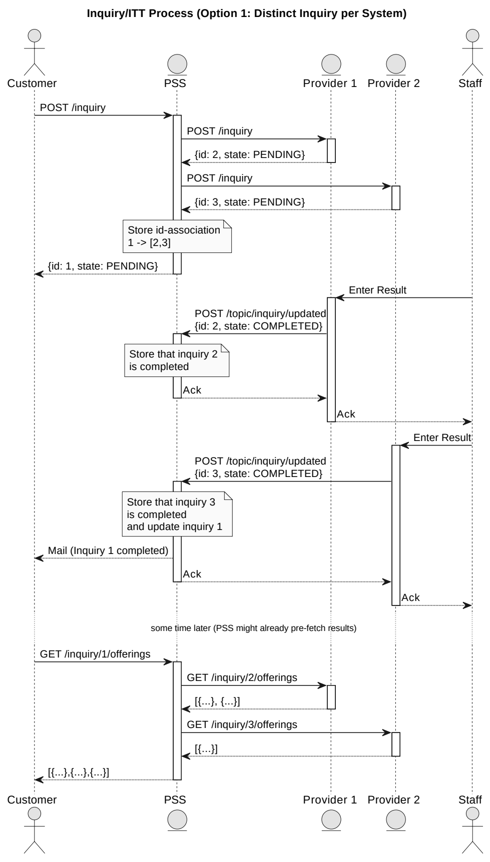

Inquiry/ITT Process (Option 1: Distinct Inquiry per System)
Customer
PSS
Provider 1
Provider 2
Staff
POST /inquiry
POST /inquiry
{id: 2, state: PENDING}
POST /inquiry
{id: 3, state: PENDING}
Store id-association 1 -> [2,3]
{id: 1, state: PENDING}
Enter Result
POST /topic/inquiry/updated {id: 2, state: COMPLETED}
Store that inquiry 2 is completed
Ack
Ack
Enter Result
POST /topic/inquiry/updated {id: 3, state: COMPLETED}
Store that inquiry 3 is completed and update inquiry 1
Mail (Inquiry 1 completed)
Ack
Ack
some time later (PSS might already pre-fetch results)
GET /inquiry/1/offerings
GET /inquiry/2/offerings
[{...}, {...}]
GET /inquiry/3/offerings
[{...}]
[{...},{...},{...}]
Customer
PSS
Provider 1
Provider 2
Staff

Figure 3.8: Distinct Inquiry per System

3.9.6.2 Common Inquiry with Synchronisation

In the second option, all systems share a common ID and synchronize the state (see figure 3.9). The inquiry is created by the user, possibly containing the list of providers they want to include (can be constructed by the PSS if not present). The PSS stores this information directly in the data structure and introduces a state for each provider. This is then propagated to the named providers using a “created”-event. It may be necessary to filter the submitted data to not disclose the other participants. The providers work on the inquiry and send an update of their own state upon completion, which is stored by the PSS (other changes are dismissed). When all providers completed their part, the overall state is set to “completed” as well and the user is informed as above. The PSS might also decide to show the intermediate results to the user and abort pending providers (which are then informed via an “update”-event as well). To prevent long delays and limiting the response time of the providers, the governance of a PSS should be able to set a *responseTime* characteristic for inquiries in the provider’s party profile. If the provider does not respond within the agreed response time, their state will be updated to “cancelled” and their product offerings will not be part of the customer inquiry results.

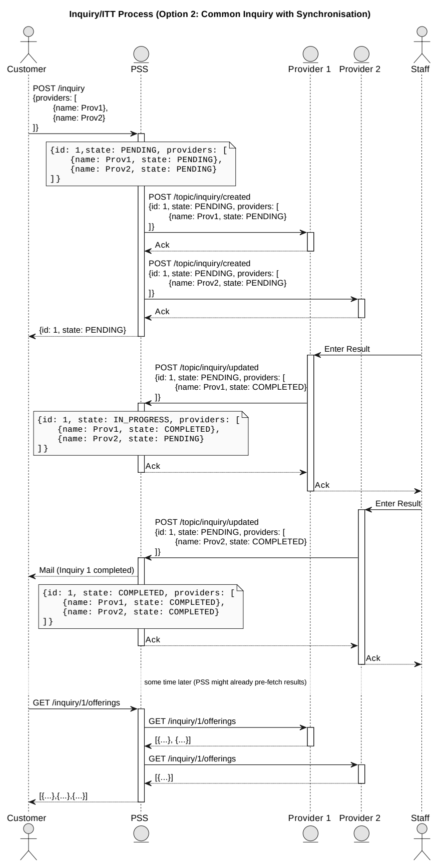

Inquiry/ITT Process (Option 2: Common Inquiry with Synchronisation)
Customer
PSS
Provider 1
Provider 2
Staff
POST /inquiry {providers: [{name: Prov1}, {name: Prov2}]}
{id: 1,state: PENDING, providers: [{name: Prov1, state: PENDING}, {name: Prov2, state: PENDING}]}
POST /topic/inquiry/created {id: 1, state: PENDING, providers: [{name: Prov1, state: PENDING}]}
Ack
POST /topic/inquiry/created {id: 1, state: PENDING, providers: [{name: Prov2, state: PENDING}]}
Ack
{id: 1, state: PENDING}
Enter Result
POST /topic/inquiry/updated {id: 1, state: PENDING, providers: [{name: Prov1, state: COMPLETED}]}
{id: 1, state: IN_PROGRESS, providers: [{name: Prov1, state: COMPLETED}, {name: Prov2, state: PENDING}]}
Ack
Ack
Enter Result
POST /topic/inquiry/updated {id: 1, state: PENDING, providers: [{name: Prov2, state: COMPLETED}]}
Mail (Inquiry 1 completed)
{id: 1, state: COMPLETED, providers: [{name: Prov1, state: COMPLETED}, {name: Prov2, state: COMPLETED}]}
Ack
Ack
some time later (PSS might already pre-fetch results)
GET /inquiry/1/offerings
GET /inquiry/1/offerings
[{...}, {...}]
GET /inquiry/1/offerings
[{...}]
[{...},{...},{...}]
Customer
PSS
Provider 1
Provider 2
Staff

Figure 3.9: Common Inquiry with Synchronisation

3.9.7 Implications for the Scope

The maximum time that is allowed to answer an inquiry is defined as a parameter in this API.

3.10 Attachment Approach

3.10.1 Context and Problem Statement

Various entities need to store attachments (PDF, DOCX, JPG etc.):

We need to define a common approach on how this binary data can be stored, shared between systems and retrieved by the related parties.

3.10.2 Considered Options

3.10.3 Decision Outcome

Define Attachment API as part of Document Management, because it separates the concern of file handling correctly, and we expect a benefit from storing metadata which justifies the slight increase of complexity.

3.10.4 Compliance

The interface definition of the Document Management API will be updated and complemented by a mock-up implementation along with this decision.

3.10.5 Pros and Cons of the Options

3.10.5.1 Plain TM Forum APIs

TM Forum defines an `attachment` attribute in all aforementioned entities. This structure can contain either a reference via id/href to another API or a concrete object containing:

This results in 4 different scenarios: local+url, local+content, ref+url, ref+content. The examples usually combine reference and concrete object, but the hrefs point to non-existing endpoints of the TM Forum "Document Management API". Looking at the documentation of it, the reason why these endpoints don't exist are obvious:

> Attachment is not a managed resource since the binary content is assumed to be resident in a Content Management System and not created by an API – only retrieved from or referred to.

It is not clear why there is even the option to reference attachments when they are not managed. On the one hand, managing (and referencing) an object which in turn just provides a URL with minimal metadata is an unnecessary overhead. On the other, storing the content in base64 is inefficient and retrieving it as part of a JSON structure is inconvenient.

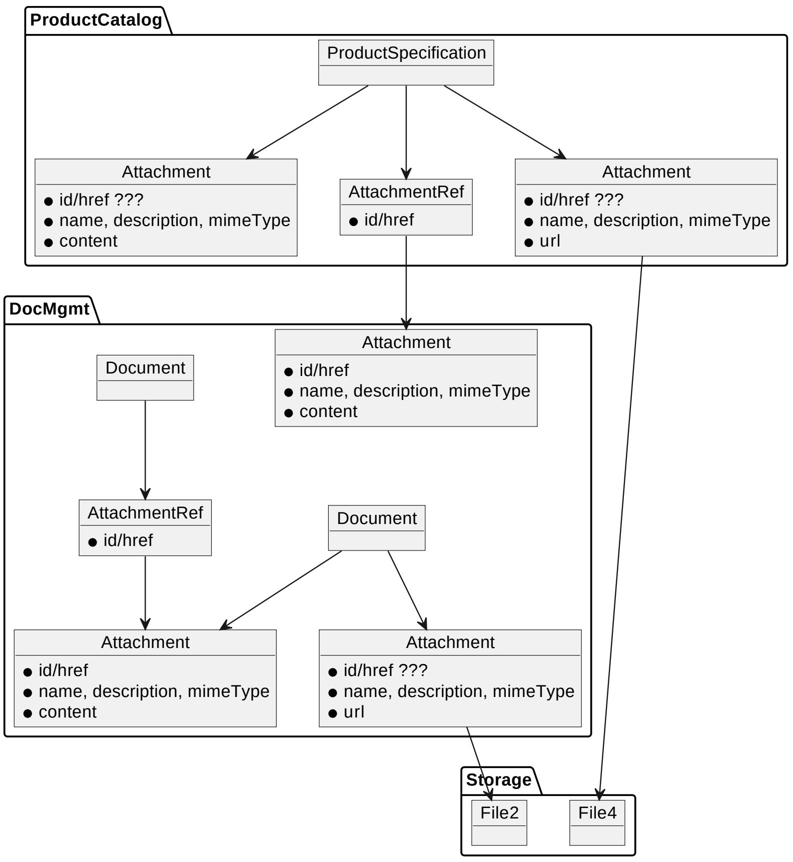


Figure 3.10: TM Forum data model for attachments

3.10.5.2 Define Attachment Endpoints in each API

Based on the problems described above, it is reasonable to remove the `AttachmentRef` as well as the base64 encoded content. This means, that all system manage attachments as part of their particular entity, but use a common storage platform (which is not defined by PSID and opaque to the user) to store the binary content. There are plenty of COTS components available, from simple local or network filesystems to cloud storage protocols like WebDAV and S3. A PSS may choose whether to allow the usage of direct links, filter them for allowed domains (e.g. allow providers to link content of their own website), implement other security measures or just prohibit it.

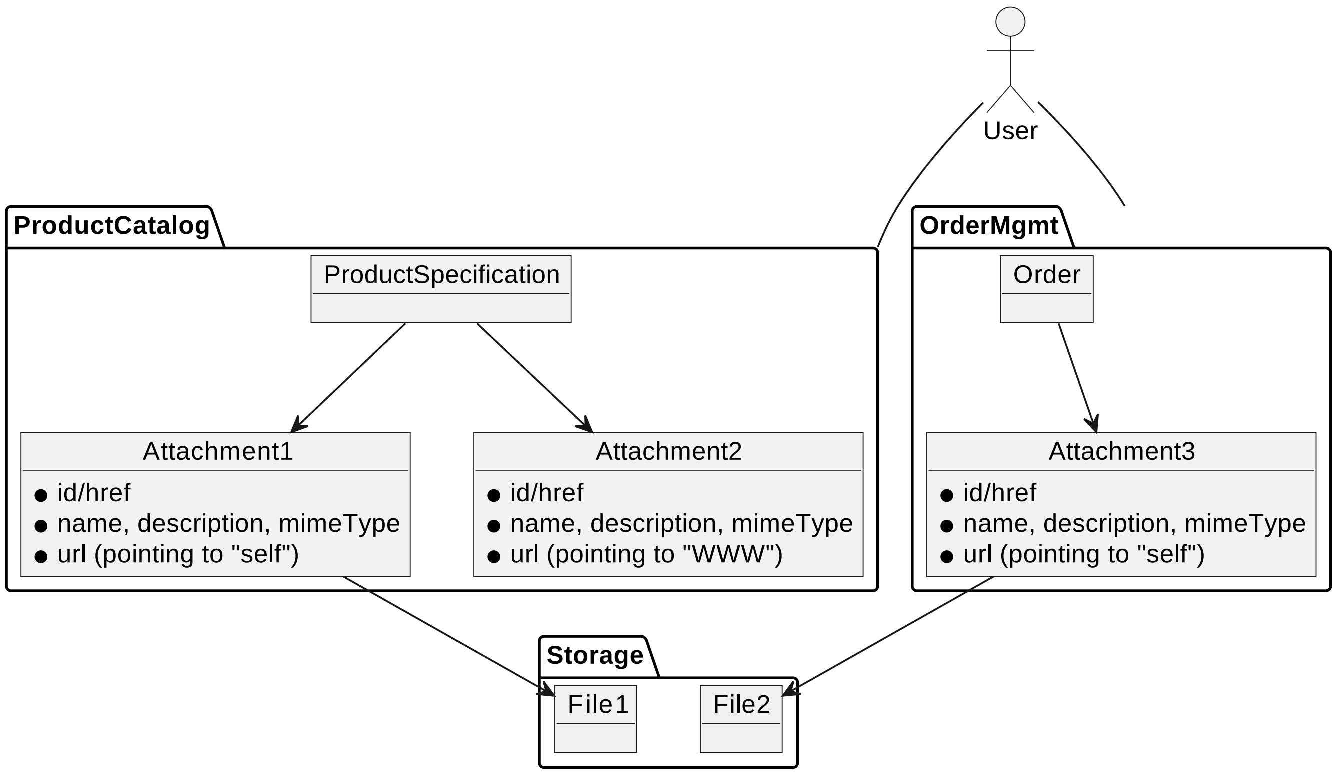

User
ProductCatalog
ProductSpecification
Attachment1
● id/href
● name, description, mimeType
● url (pointing to "self")
Attachment2
● id/href
● name, description, mimeType
● url (pointing to "WWW")
OrderMgmt
Order
Attachment3
● id/href
● name, description, mimeType
● url (pointing to "self")
Storage
File1
File2

Figure 3.11: Data model for attachments without Document Management

The structure of the endpoints would be standardized by PSID, which allows the PSS to easily implement this as a cross-cutting-concern (e.g. a library). The message sequence would be quite straight forward: After a catalog entry is created, the newly defined endpoint can be used to upload attachments to it.

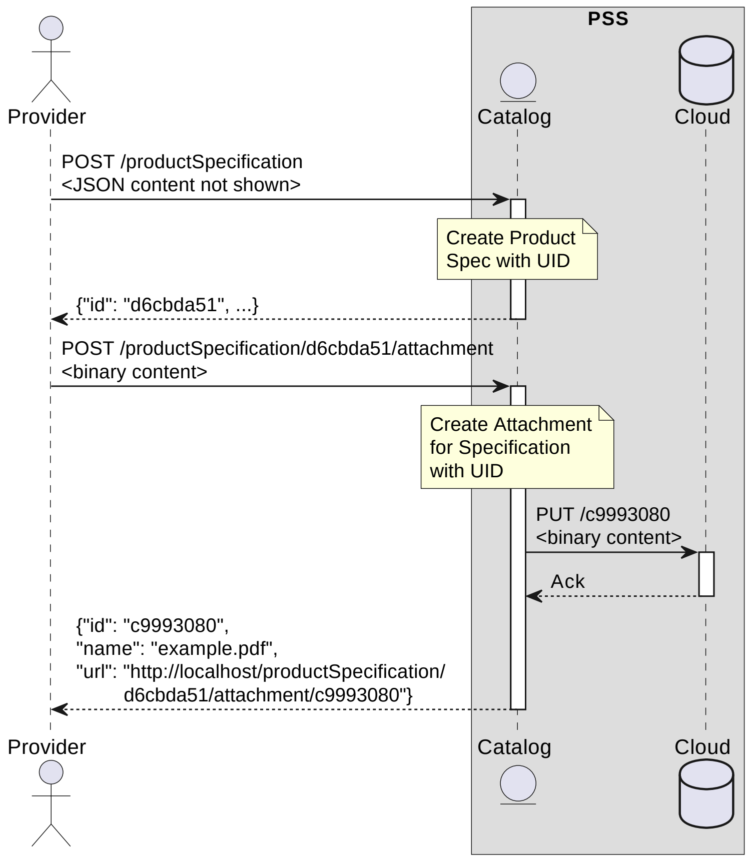

PSS
Provider
Catalog
Cloud
POST /productSpecification
<JSON content not shown>
Create Product Spec with UID
{"id": "d6cbda51", ...}
POST /productSpecification/d6cbda51/attachment
<binary content>
Create Attachment for Specification with UID
PUT /c9993080
<binary content>
Ack
{"id": "c9993080",
"name": "example.pdf",
"url": "http://localhost/productSpecification/d6cbda51/attachment/c9993080"}
Provider
Catalog
Cloud

Figure 3.12: Provider message sequence without Document Management

The (potential) customer can now browse the catalog and retrieve the attachment through the same API:

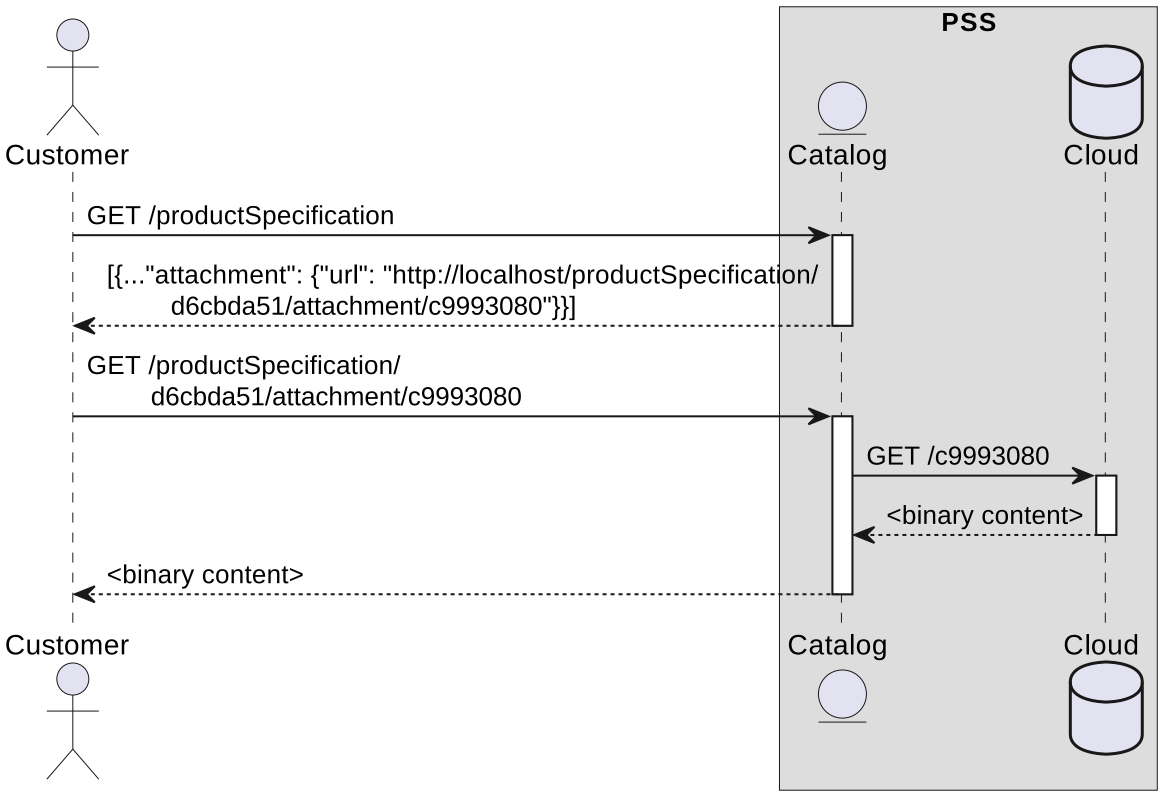

PSS
Customer
Catalog
Cloud
GET /productSpecification
[{..."attachment": {"url": "http://localhost/productSpecification/d6cbda51/attachment/c9993080"}}]
GET /productSpecification/d6cbda51/attachment/c9993080
GET /c9993080
<binary content>
<binary content>
Customer
Catalog
Cloud

Figure 3.13: Customer message sequence without Document Management

Note that some storage systems like Amazon S3 allow the creation of temporary access URLs to reduce load on the actual Catalog backend. This can be utilized by redirecting the client when they access the attachment instead of passing through all the data.

3.10.5.3 Define Attachment API as part of Document Management

To fix the problems of the previous options, one would have to define a central API that actively manages all attachments. Simply exposing the storage causes a lot of additional problems concerning isolation of tenants, especially with filename uniqueness. Additionally, defining it as part of PSID would put very hard constraints on the implementation and also diverge from the other APIs, which are TM Forum compliant.

So as a first step, a simple facade would be possible (named Attachment Management):

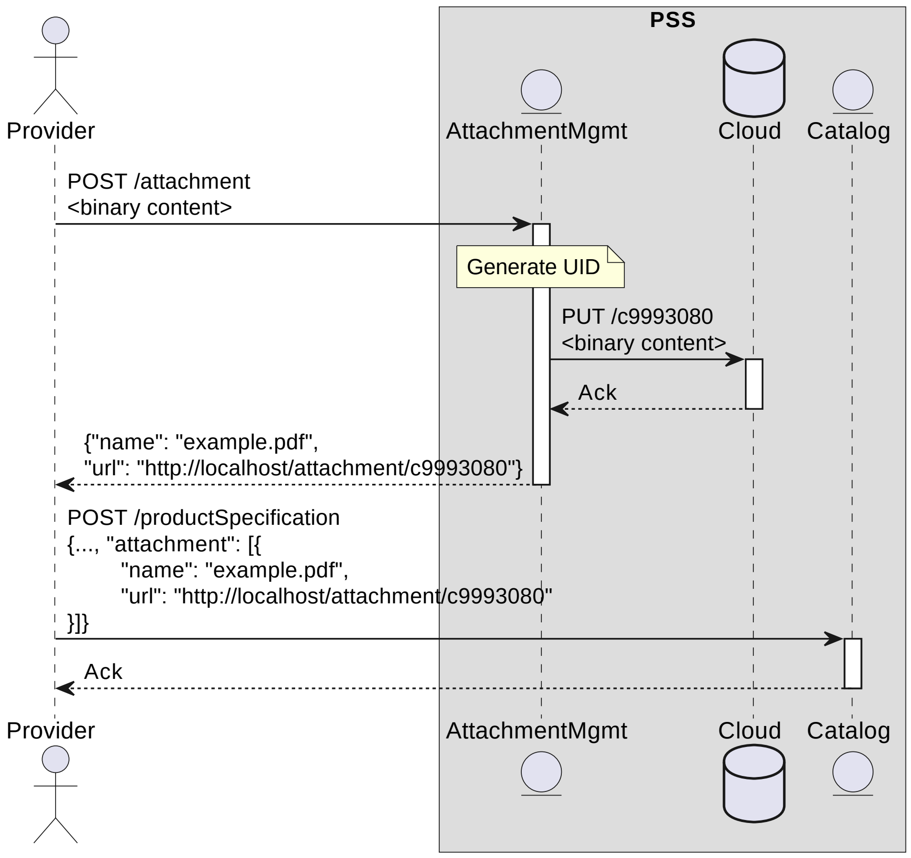


Figure 3.14: Provider message sequence with Attachment Management

Since the Attachment and Catalog APIs are now separated, there is a need to look up and manage files independently. Especially when it is uploaded but not used in any other system, there must be a way to find and potentially delete (or re-use) it. This means the Attachment API has to store metadata and provide interfaces to browse through it. Introducing this brings us back to the Document Management API of TM Forum, but enables us to make slight modifications in differentiating between the attachment entities that are part of a document and their counterparts in other subsystems:

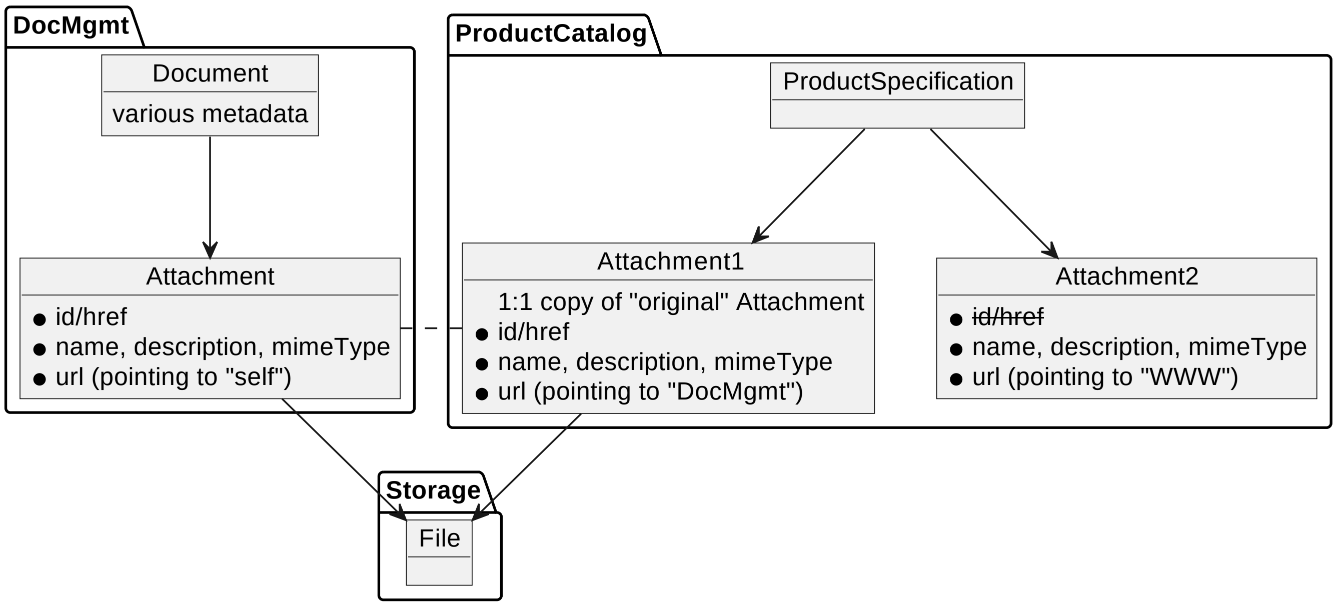

DocMgmt
Document
various metadata
Attachment
● id/href
● name, description, mimeType
● url (pointing to "self")
ProductCatalog
ProductSpecification
Attachment1
1:1 copy of "original" Attachment
● id/href
● name, description, mimeType
● url (pointing to "DocMgmt")
Attachment2
● ~~id/href~~
● name, description, mimeType
● url (pointing to "WWW")
Storage
File

Figure 3.15: Data model for attachments with Document Management

There are now only two distinct options how to use attachments: either uploading it as part of a Document in the Document Management of the PSS or by direct reference as part of another entity. A PSS may choose to prohibit the usage of direct links, filter them for allowed domains (e.g. allow providers to link content of their own website) or implement other security measures. *Not* included is the option to use external URLs as attachments to documents, standalone-attachments without documents and pure references (without direct access URL) in other entities.

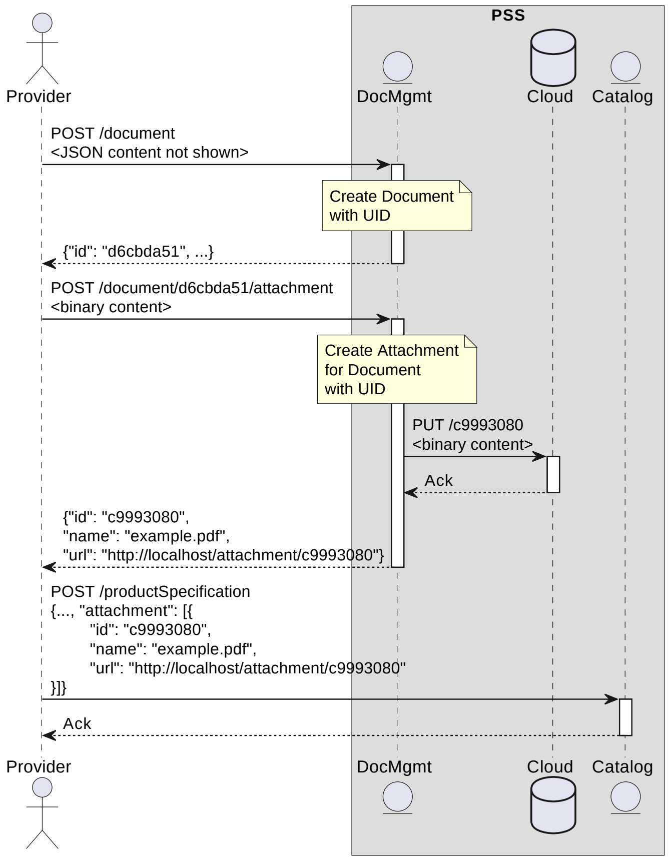


Figure 3.16: Provider message sequence with Document Management

To create an attachment, the provider now has to first create a document containing metadata about it. After that, they use the document ID to upload attachments to it (the Document Management can decide where to store the binary data). The Document Management creates the URL where the attachment can be downloaded by the user (or other related parties). All management of the attachment (name changes, deletion) is then be done as a sub-resource to the document, meaning the endpoints in `/attachment/{attachmentId}` are purely for binary access, while `/document/{documentId}/attachment/{attachmentId}` allows access to the JSON structure that is also used in other systems.

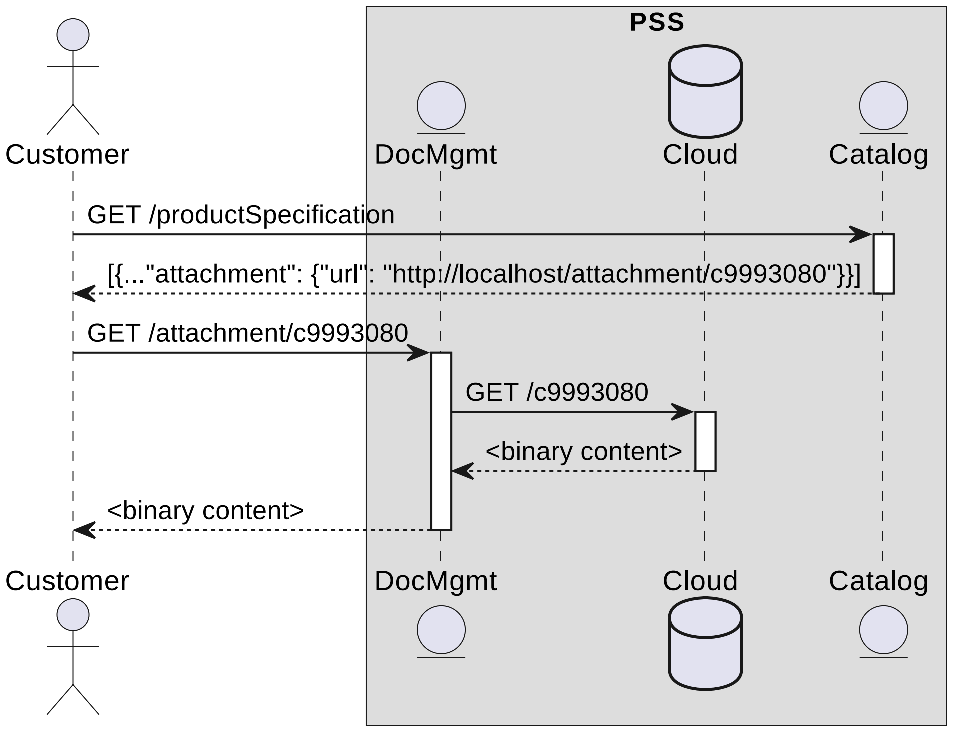


Figure 3.17: Customer message sequence with Document Management

Note that some storage systems like Amazon S3 allow the creation of temporary access URLs to reduce load on the actual Document Management. This can be utilized by redirecting the client when they access the attachment instead of passing through all the data.

3.11 Beam Handling

3.11.1 Context and Problem Statement

The PSS needs to know where a service can be delivered, which implies knowledge of the involved beams, especially their respective footprints and EIRP values. But beams are not always fixed, so we need to find a way to handle this.

3.11.2 Decision Drivers

3.11.3 Considered Options

3.11.4 Decision Outcome

Chosen option: “Allow definition of beam and service boundaries”, because this information is just enough for coarse matchmaking and simple compared to other options.

Accordingly, the PSID will *not* include time-dependent beam geometries, which would highly increase the complexity of matchmaking and doesn’t solve the problem of steered beams and confidential information. Also, *no* new workflow to periodically update beams is specified, which would require some unnecessary effort on the provider side and doesn’t cover the case of confidential information. In other words, the beam prediction and allocation will only be handled by the provider and not by the PSS. PSS governance and provider can agree to update dynamic information through existing Catalog and Inventory APIs if required, for example when the PSS is actively managing committed resources.

This leaves the following two extremes for defining beam resources:

As an intermediate use case the API also allows providing exact beam information without having pre-configured services, which were included in the above sample for better understanding. Note that the already introduced GeoJSON data format is able to describe “holes” for those areas, which can be used to exclude certain areas from a bigger polygon.

3.11.5 Compliance

ICD and TAD will be updated to explicitly show this behaviour.

3.11.6 Implications for the Scope

The creation of the listing of geographical regions is not part of the API, but a PSS-internal process. However, the footprint is *one* source this information can be taken from.

Beam handling is a PSS internal process, while the definition of GeoJSON zones are available in the interface definition. Thus, the footprint is *one* source of information for defining a region, e.g. as pool.

The interface enables the exchange of footprints in any desired precision agreed between PSS governance and provider. This information can be used by the governance to derive the total set of regions where services are offered and display it to the user. However, this is not part of the API, but a PSS-internal process.

The GeoJSON data format is able to describe “holes” within areas, which can be used to exclude certain areas from a bigger polygon.

Implementing GeoJSON data format allows, from API perspective, defining mission areas. How this is done, however, is a PSS-internal process.

The coverage of beams is one way to allocate a geolocation to a resource, service, or product. An easy way to make a first approach for mapping, is to compare a service request’s localization with the available satellite footprints. However, this is not part of the API, but a PSS-internal process.

During the lifecycle of a satellite, its footprint(s) may or may not change, even continuously. Pushing updates when required is the easiest way to manage these changes.

Pooling based on the geographical region requires resources, services, and products to have a location or an area where they can be deployed.

The footprint is an easy and reasonable way to do this. However, this is not part of the API, but a PSS-internal process.

3.12 Security Scope

ID: ADR019 * Status: :accepted: * Deciders: @dvs @rgs @vmr @wr @cg @daf @cgr @ncz @rsperb * Date: 2023-03-15 * Version: 1.1 * Category: Security

3.12.1 Context and Problem Statement

The scope of security aspects that shall be covered by the API definition is to be decided.

3.12.2 Decision Outcome

As for other aspects we strongly divide between “the API definition” (the main output of this project) and “the implementation” of the API (a real PSS or provider system). Thus, this decision is mainly considering **where** security is handled, not **if**.

For PSI this means that the interface must be interpreted as *one* building block of an overall solution. The solution must be enabled to allow strong encryption and meaningful authentication. Thus, the interface must not block such techniques. However, the interface cannot and should not enforce the usage of such techniques. They are considered part of the domain solution, not part of the interface definition. The interface will (only) strongly suggest in the documentation the application of such techniques.

This is in line with TM-Forum’s approach[5].

3.12.2.1 Authentication

Authentication is the process of validating the identity of a requests source. In HTTP-based communication it is usually implemented by credentials (username, password, second-factor) stored in a server-side session or a JWT[6], API-Keys or a PKI[7]. It is performed via one or more requests before the actual API access, and transported in a separate HTTP-Header field and therefore has no effect on the transported data. Such methodology can also be used to introduce a correlation ID tracking, e.g. for auditing purposes. It was agreed to exclude it from the scope of the API definition, but add it as a prerequisite into the documentation with a reference to the TM Forum REST API Guidelines, which suggest the use of OpenAuth 2.0.

As per requirements, additional accreditation of participants will be performed, which leads to even higher trust in the legitimacy of authenticated requests. The actual process is considered to be outside the project scope, as it is performed before the API can be used.

Authentication of the server towards the client on the other hand is an integral part of the mandatory SSL protocol. It will not be defined how the PKI is implemented, as this depends on the concrete security level of the participants. Therefore, no additional information will be provided.

3.12.2.2 Authorization and Data Visibility

Authorization is closely related to the above discussed authentication: It is the process of determining whether the authenticated entity has access to a source. Because of the close relation, it is not possible to define this without implying an authorization method. The documentation will be extended with guidelines to implement this, for example using role based access control (RBAC).

Especially for a multi-provider PSS, but also for a provider handling data from multiple customers, it is very important to prevent data leaking between parties. This again is the responsibility of the implementation and is already integrated into the phrasing of requirements. The interface definition describes the structure of data that can be exchanged, while the implementation must decide which entities (and attributes) are actually exposed to which authenticated party.

[5]https://www.tmforum.org/resources/specification/tmf630-rest-api-design-guidelines-4-2-0/
[6]JSON Web Token (see http://jwt.io/)
[7]Public Key Infrastructure

3.12.2.3 Transport Encryption

All data that is exchanged between systems has to be encrypted while it is transported. As already stated in the ICD, this is accomplished by using SSL with TLS v1.2 or higher. Additional layers like VPNs, custom PKI and certificate-pinning are possible, but out of scope for this project.

Another possibility is to use the JWE[8] technology to encrypt data end-to-end. This allows obfuscating the data even for reverse-proxies or other middle boxes. There is currently no known data-exchange where it seems reasonable to enforce this. Since the used data format can easily be adapted for this approach, some information on the subject will be added to the ICD.

3.12.2.4 Data at Rest

Securing data at rest (on hard drives, databases etc.) from unauthorized access and data loss is out of the scope of the interface.

3.12.2.5 Jamming

Jamming attacks could be conducted on single channels as well as multiple channels. The attack might be automatically mitigated on the provider's side by failover. Although anti-jamming measurements are to be implemented on the provider's side, a PSS should be enabled to handle information related to jamming attacks or anti-jamming mitigations, if provided by the providers. Thus, the API should enable the forwarding of such information. The actual enforcement of anti-jamming measurements has to be done by the governance of a PSS.

3.12.2.6 Other Attack Vectors

The API could be misused by attackers, e.g. via denial-of-service (including brute-force on authentication) or replay attacks. This cannot be prevented by the API definition, but by the security design of the implementation. It is therefore out-of-scope for this project.

3.12.3 Compliance

As a first step, the above-mentioned possibilities will be added to the ICD with useful links to external sources. Depending on the extent, it may be decided to extract it to a new document later.

Additionally, TOD and REQ are to be checked and augmented with references to these new chapters of the ICD.

3.12.4 Implications for the Scope

The accreditation and vetting processes can be performed on data provided via the interface. The interface itself can only define its formatting, but not its content. The business processes necessary to implement accreditation and/or vetting are - in principle - enabled by the PSI, but cannot be enforced.

The PSS is enabled to map the given securityLevels to any classification system. The mapping itself is considered a PSS-internal process.

[8]JSON Web Encryption (see http://jwt.io/)

As discussed above, the interface must be interpreted as *one* building block of an overall solution. The solution must be enabled to allow strong encryption, meaningful authentication, and other security measures. Thus, the interface must not block such techniques. However, the interface cannot and should not enforce the usage of such techniques. They fall into the domain solution and not the interface.

As discussed above, the API could be misused by attackers. This can not be prevented by the API definition, but by the security design of the implementation. It is therefore out-of-scope for this project.

This requirement applies to the overall solution, not the interface definition in particular. Thus, compliance with the requirement cannot be enforced, but the interface definition enables a PSS to be verified against the requirement.

The interface cannot enforce any security measures on data at rest, therefore, this candidate requirement is considered to be implemented only partially.

PSS internal mitigation approach against spoofing. Considering P&S_164, the interface cannot and must not enforce the usage of specific techniques, thus, the requirement is considered to be out of scope for the interface definition.

PSS internal process. Whether this requirement is fulfilled or not depends on the concrete PSS connected to the interface and cannot be enforced by the interface.

An isolation of parties within the system requires input via the interfaces, e.g., with the Party Management API, the Party Role Management API, etc. However, the PSS itself needs to implement a corresponding isolation based on this input. We consider this as a security feature that is enabled but not enforced by the PSI.

The data model implied by the TM Forum Open APIs and adapted in PSI does not differentiate between users and providers on the party level. If the PSS is considered to be working similarly to a digital marketplace, this does make a lot of sense. The difference between user and provider is basically their role, i.e., if they buy or sell products. A customer can, in principle, have both roles.

The interface allows setting the link status (nominal, reduced, downgraded, broken) via API to spot faulty links. Thus, the interface is to be understood as an enabler for security aspects, not enforcing such for a PSS connected to it.

The interface is developed with state-of-the-art designs and is based on a common industry standard. The interface is to be understood as an enabler for security aspects, not enforcing such for a PSS connected to it.

Whenever a specific technology is considered for usage together with the PSI, the corresponding decision record lists a summary of the probability scores of each option regarding the CIA triad. The interface is to be understood as an enabler for security aspects, not enforcing such for a PSS connected to it.

There are only parts of this candidate requirement to be applied to an interface definition, because most of them can only be applied to a PSS or in some cases to the implementation of the interface. As discussed above, this is out of the scope of this project. Nevertheless, whenever applicable, the security aspects are analysed based on common security standards.

As described above, the interface enforces transport layer encryption using SSL with TLS v1.3 or higher. However, data at rest and its security are out of scope of this project.

As the chosen approach allows sending a correlation ID within the separated header, the interface definition enables auditing processes.

3.13 Inventory and Stock API

3.13.1 Context and Problem Statement

As stated in the decision on Overbooking (V2.0), we need to exchange information about concrete resources somewhere. These have to contain details about the actual set of characteristics (in contrast to the variations of specifications) and the time-dependent booking state. Additionally, we have to consider some variety:

3.13.2 Considered Options

3.13.3 Decision Outcome

Inventory and Stock API are to be used for different purposes. Further investigation on the Product Offering Qualification API is put into the project backlog for now.

The inventory API will be used to track the state of resources. For *all* resources there will be a master-entity, either in the provider system (for on-demand resources) or the PSS (committed resources). Every order will create a sub-entity in the PSS (optionally also in the provider system) that is assigned to the customer for the given timespan. Depending on the implementation, the resource can be further subdivided, e.g. if it is shared with another user or is resold by a service provider.

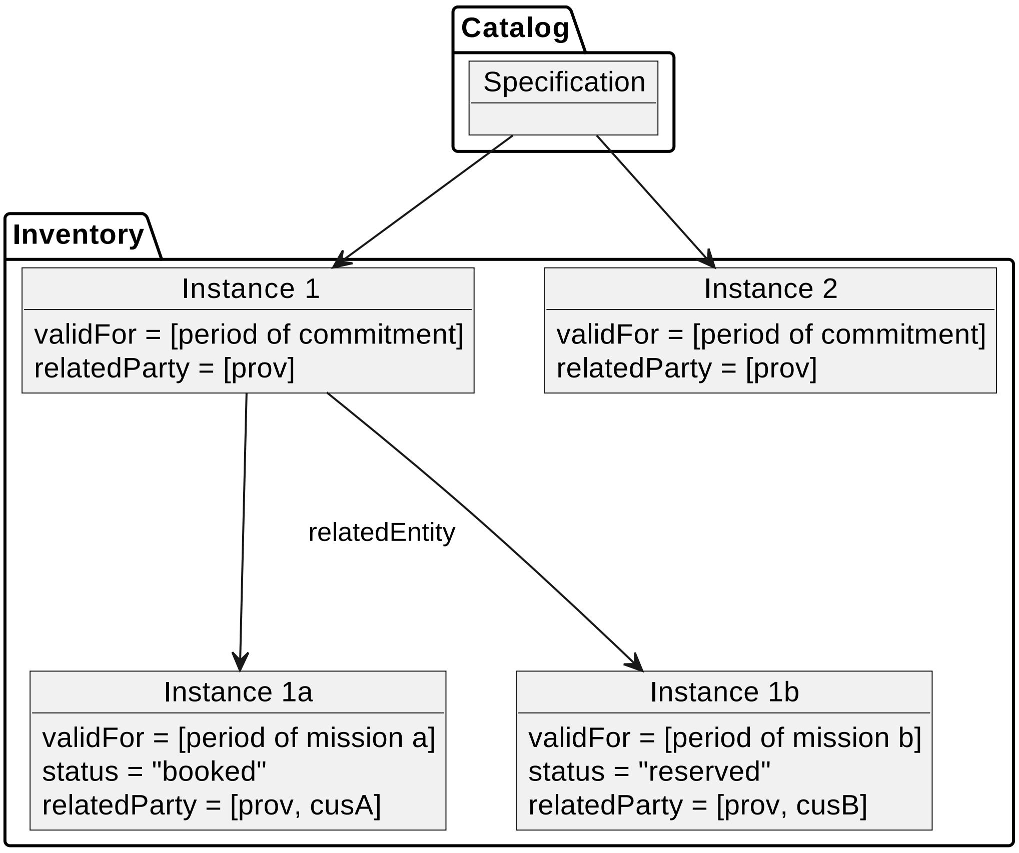

Catalog
Specification
Inventory
Instance 1
validFor = [period of commitment]
relatedParty = [prov]
Instance 2
validFor = [period of commitment]
relatedParty = [prov]
relatedEntity
Instance 1a
validFor = [period of mission a]
status = "booked"
relatedParty = [prov, cusA]
Instance 1b
validFor = [period of mission b]
status = "reserved"
relatedParty = [prov, cusB]

Figure 3.18: Inventory Data Model

Physical resources (e.g. terminals) are bookable when there is no sub-entity in the inquired timespan. For logical resources (e.g. bandwidth) the implementation has to sum up the assigned amount (MHz or Mbit), subtract it from the available amount and compare it with the inquired value.

The “Check Product Stock” part of the Stock API will be used to allow an automatic check of availability for on-demand resources. The field `availabilityDate` will be extended to support time ranges.

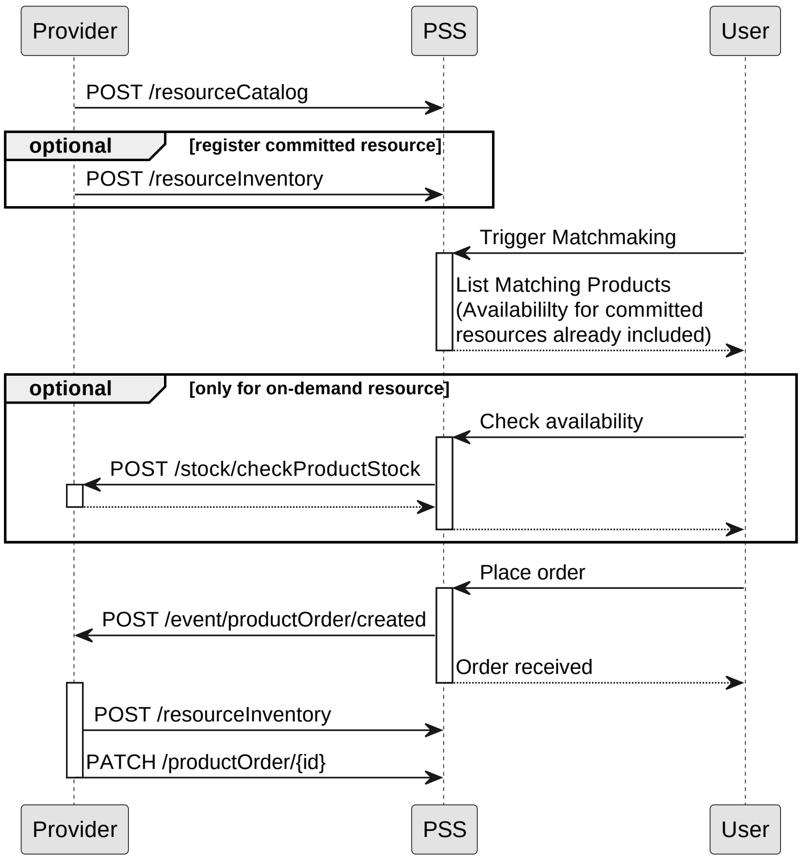


Figure 3.19: Message exchange with inventory and stock

3.13.4 Compliance

The description of overbooking will be added to the TAD. The interfaces will be defined in the ICD and mock-up.

3.13.5 Pros and Cons of the Options

3.13.5.1 Inventory APIs

The Inventory APIs manage concrete instances of products, services and resources. Each entry is derived from and linked to a specification (which is managed by the catalog).

For *on-demand* resources, the PSS would have to request the availability from the provider. Using only this API would mean to basically open the whole inventory towards the PSS and let it look up the requested resources.

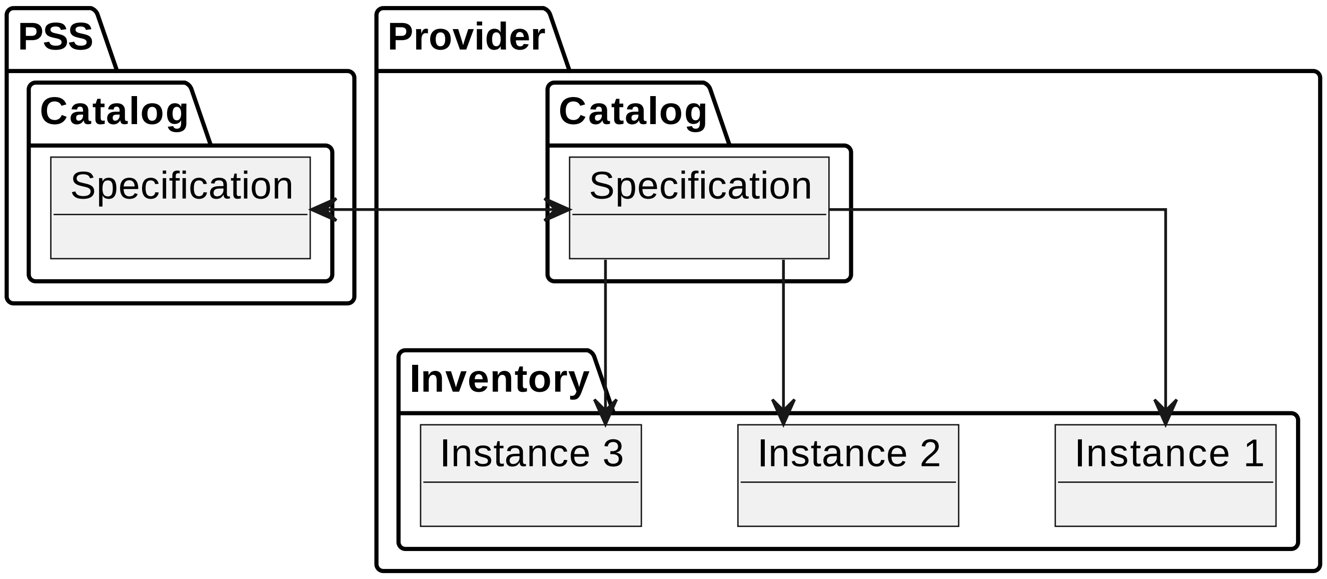


Figure 3.20: Inventory of On-Demand Resources Before Order

When the order is accepted, the provider would also create the resource in the PSS inventory. The resource-copy lives for the duration of the mission and is marked invalid afterwards. This lifecycle is applicable for physical and logical resources alike.

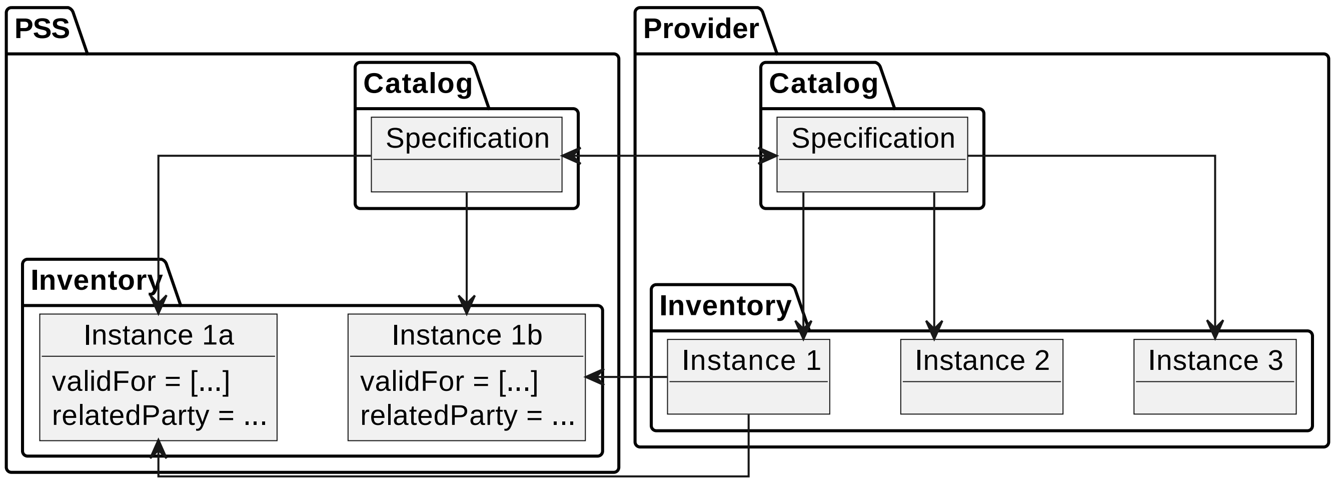


Figure 3.21: Inventory of On-Demand Resources After Order

To *commit* a resource, it would be created in the PSS inventory before any order is placed. Therefore, the availability check can be performed internally in the PSS without additional data exchange. The order is still sent to the provider system as for all other products, which allows the assignment of an alternative resource and even rejection in exceptional cases. It is up to the SLA defined between the governance and the provider how strict this is handled.

Since the resource now already is in the PSS inventory, we'd have to define more sophisticated lifecycles:

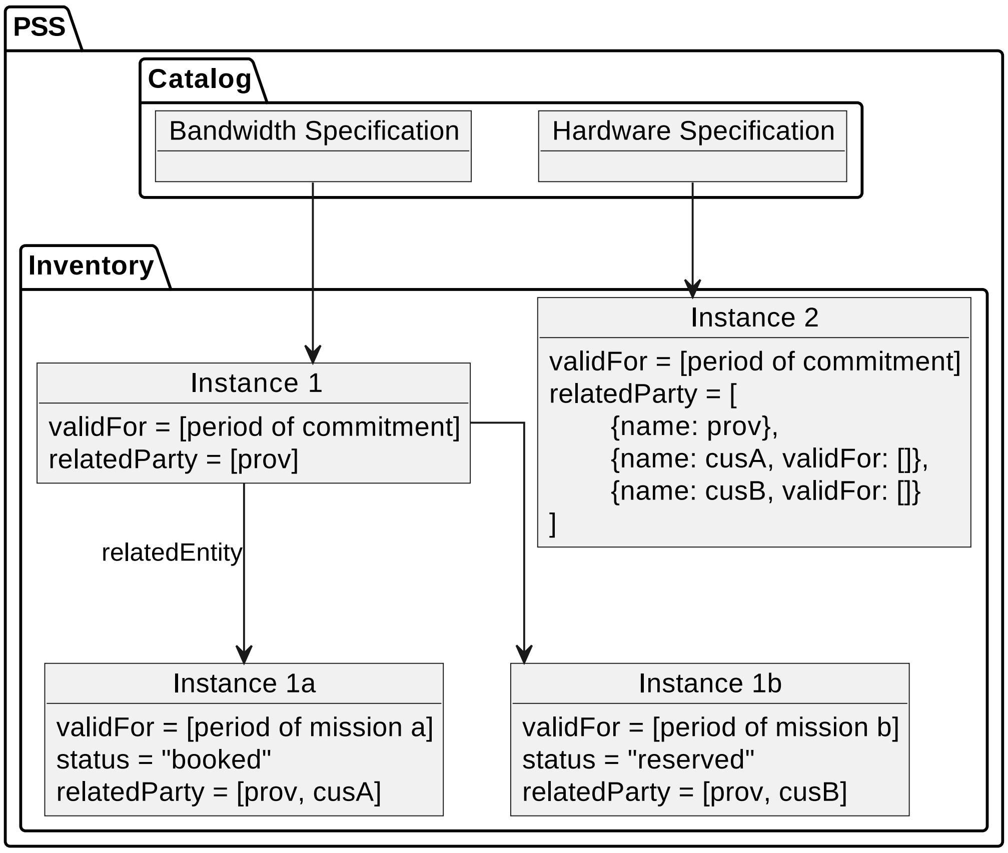

PSS
Catalog
Bandwidth Specification
Hardware Specification
Inventory
Instance 1
validFor = [period of commitment]
relatedParty = [prov]
Instance 2
validFor = [period of commitment]
relatedParty = [
{name: prov},
{name: cusA, validFor: []},
{name: cusB, validFor: []}
]
relatedEntity
Instance 1a
validFor = [period of mission a]
status = "booked"
relatedParty = [prov, cusA]
Instance 1b
validFor = [period of mission b]
status = "reserved"
relatedParty = [prov, cusB]

Figure 3.22: Inventory of Committed Resources After Order

3.13.5.2 Stock API

The Stock API is an additional layer above the inventory to manage a physical stock. It is designed to allow a standardized query if a product is available in a store or warehouse. From a business perspective, it can also be used to automatically restock those below a threshold. Theoretically, it can be used as a standalone API without an exposed inventory.

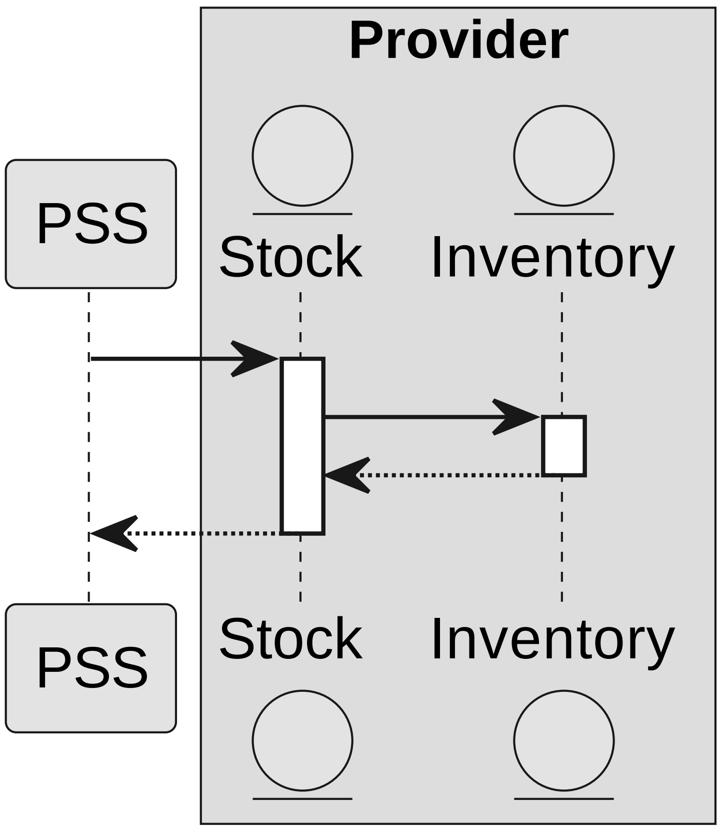


Figure 3.23: Stock Query Sequence

As long as the PSS is not considered a real reseller with its own warehouse, it doesn't make sense to implement this API there. The question of "who is using a resource when" cannot be answered by the given endpoints and data model. It could be used to query for on-demand resource availability from the provider, but the nature of the API makes it unwieldy to manage bandwidth. For this purpose we would include the specific endpoints for the operations "Check Product Stock" or "Query Product stock" in the PSID.

Feature	Query	Check
Instant Response	✓	✓
Async Response	✓	✓
Availability Date	✓	✓
Amounts	✓	✓
Characteristics	✓	✓
Multiple Items		✓
Allow Alternatives		✓
Multiple Results	✓	

Table 3.3: Feature Comparison of the Query and the Check Endpoint.

The field `availabilityDate` would have to be extended to support time ranges. The `place` fields might be removed, but could also be kept as optional. Overall, both APIs answer similar questions:

Other endpoints of actual stock management can be used by the provider internally as defined by TM Forum.

3.13.5.3 Product Offering Qualification API

The Product Offering Qualification API is another task-based API like the Stock API and uses similar patterns. It is intended for checking commercial eligibility of a product offering. Since this should already been checked as part of the inquiry process, it can currently be de-scoped. It may be considered again to check change requests for eligibility.

3.13.6 Implications for the Scope

Matchmaking is considered a PSS-internal process. The prioritisation of committed resources over others is not part of the interface.

The catalog and inventory APIs are not relevant for dynamic provision of resources, only for their definition.

Committed resources need the above discussed special handling and differentiation between the catalog and the inventory. Therefore, the API considers both.

Until an actual order is placed, on-demand resources only need a catalog entry, but no inventory entry. Therefore, the API considers both.

The inventory API foresees a period of validity for a committed resource.

The inventory API foresees the division of resources into sub-resource and further leasing of those.

3.14 Handling of Partial Matches

3.14.1 Context and Problem Statement

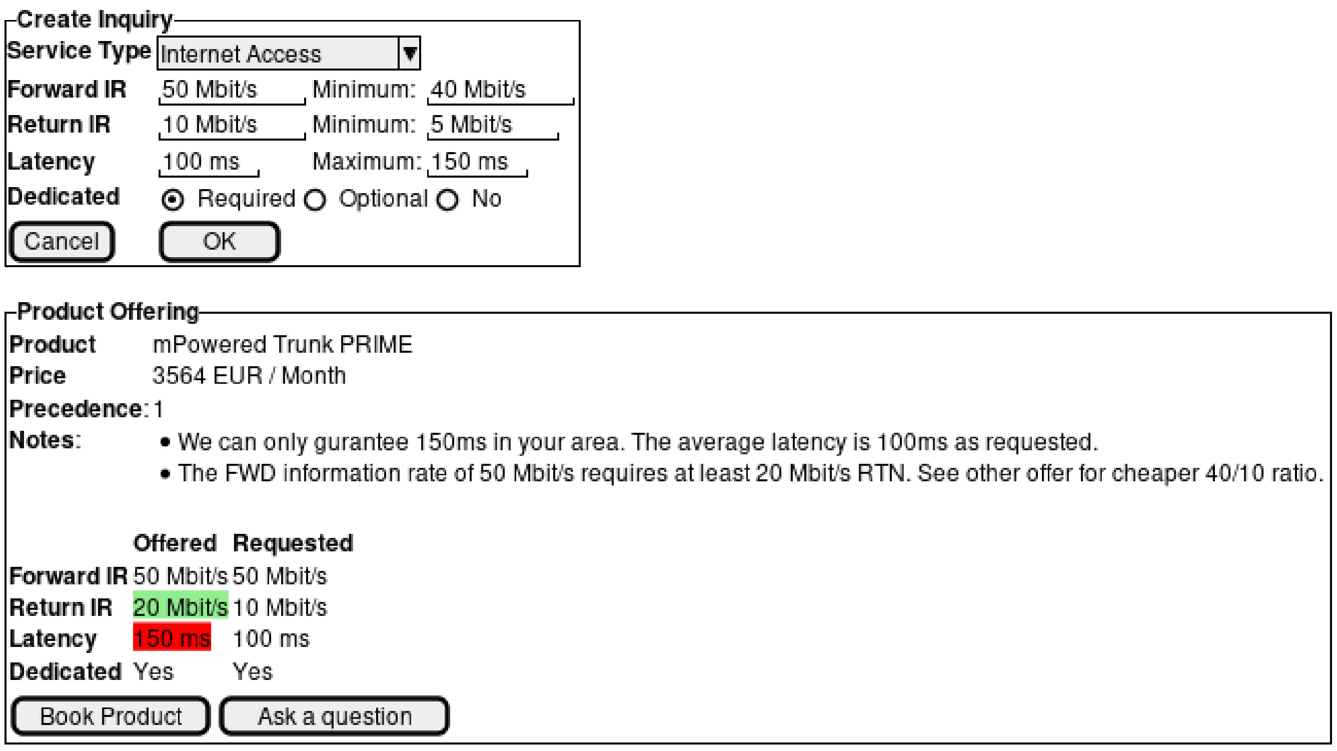
Inquiries and their responses are not always precise: A customer may want to have a specific characteristic value (e.g. 20 Mbit/s or 100ms latency), but it may not be feasible or available. This could be due to technical reasons (e.g. the frequency band or equipment does not allow for it), lacking resources in the requested timeframe or economic reasons (e.g. a little change can reduce the price a lot). The customer is often not aware of these limitations when issuing the inquiry. We have to decide how to handle these cases in the API, considering automatic and manual processes on both sides.

3.14.2 Decision Drivers

3.14.3 Considered Options

3.14.4 Decision Outcome

Option 3 was chosen for both aspects. This gives maximum flexibility to the Inquiry API, while allowing simplifications on the UI level. On the other side, the provider can justify deviations and add other notes.



Create Inquiry

Service Type	Internet Access		
Forward IR	50 Mbit/s	Minimum:	40 Mbit/s
Return IR	10 Mbit/s	Minimum:	5 Mbit/s
Latency	100 ms	Maximum:	150 ms
Dedicated	[✓ Required] Optional No		

Product Offering

Product mPowered Trunk PRIME
Price 3564 EUR / Month
Precedence: 1
Notes:

- We can only gurantee 150ms in your area. The average latency is 100ms as requested.
- The FWD information rate of 50 Mbit/s requires at least 20 Mbit/s RTN. See other offer for cheaper 40/10 ratio.

	Offered	**Requested**
Forward IR	50 Mbit/s	50 Mbit/s
Return IR	20 Mbit/s	10 Mbit/s
Latency	150 ms	100 ms
Dedicated	Yes	Yes

Figure 3.24: Final Inquiry Form with Response (Wireframe)

Note that in the wireframes, the “dedicated” attribute is depicted as boolean. This serves as a simplified example and does not imply it to be boolean in the implementation. The example data used in testing in fact does model this as MIR and PIR integers.

3.14.5 Compliance

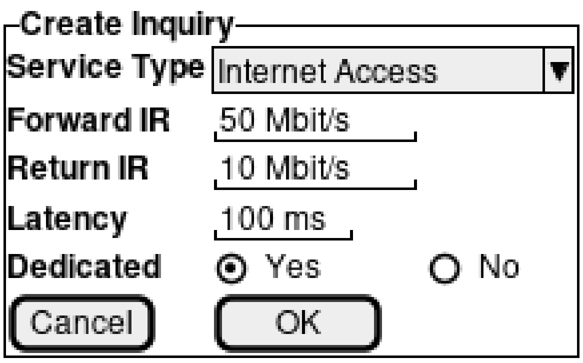
Include new fields in documentation (ICD, TOD), mock-up implementation and tests.

3.14.6 Pros and Cons of the Options

3.14.6.1 For Inquiries

Sending only target values is easier for the customer and what they would usually do. It requires implicit behaviour of the provider and therefore could lead to more iterations. For example, they (person or algorithm) have to make assumptions how much deviation is acceptable and which characteristic has higher priority, e.g. “low latency is more important than high information rate”.

Please take into account, that a PSS could provide HMIs that allows customers to define their requirements in a less technical manner, like “We need an internet connection that has to support up to 10 video streams and 5 VOIP connections in parallel”. It is assumed, that “video streams” and “VOIP connections” are not a unit for internet connections that providers usually support. Instead, it is up to the PSS application to transfer these extended requirements into accepted technical units that can then be sent as a customer inquiry to external matchmakings.

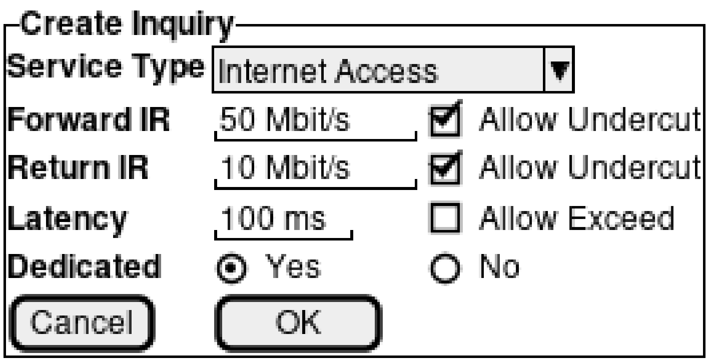


Create Inquiry

Service Type	Internet Access ▼	
Forward IR	50 Mbit/s	
Return IR	10 Mbit/s	
Latency	100 ms	
Dedicated	[✓ Yes]	No
Cancel	OK	

Figure 3.25: Plain Inquiry Form (Wireframe)

Boolean flags solve the problem of possible inaccuracies in the provided results and possibly associated iterations, since they allow specifying beforehand whether deviations are accepted. The question of default would arise, because the customer may not voluntarily forgo on a characteristic.

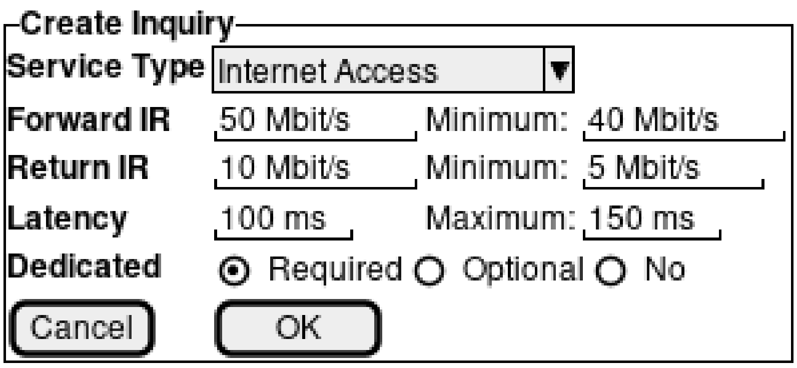


Create Inquiry

Service Type	Internet Access ▼	
Forward IR	50 Mbit/s	☑ Allow Undercut
Return IR	10 Mbit/s	☑ Allow Undercut
Latency	100 ms	☐ Allow Exceed
Dedicated	[✓ Yes]	No
Cancel	OK	

Figure 3.26: Inquiry Form with Deviation Flags (Wireframe)

The most verbose approach is to define minimum, maximum and target values for each characteristic. It allows the matchmaking algorithm (or provider) to search in concrete boundaries without assumptions. This puts a burden on the customer to define these, though they are expected to use a guiding HMI which could ease this. If the UI foresees different levels of expertise, it may fall back to the visuals from above for beginners and provide reasonable limits for the API by itself. With this option, the customer is still able to define unachievable service requests.



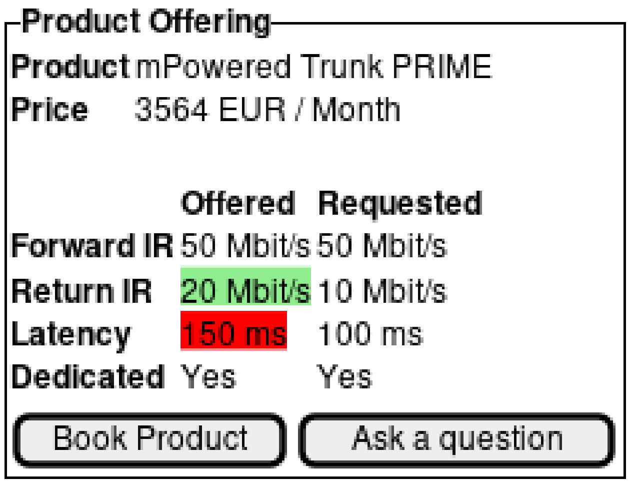
Create Inquiry

Service Type	Internet Access ▼		
Forward IR	50 Mbit/s	Minimum:	40 Mbit/s
Return IR	10 Mbit/s	Minimum:	5 Mbit/s
Latency	100 ms	Maximum:	150 ms
Dedicated	[✓ Required]	Optional	No
Cancel	OK		

Figure 3.27: Inquiry Form with Min/Max Inputs (Wireframe)

3.14.6.2 For Responses

The trivial option is to not include additional information in the interface. This means the provider (or their system) can select a partially matching product and just return it. The customer (or the HMI that displays the result, see 3.28) then has to find the differences. It could be argued that they would have to do this comparison anyway.

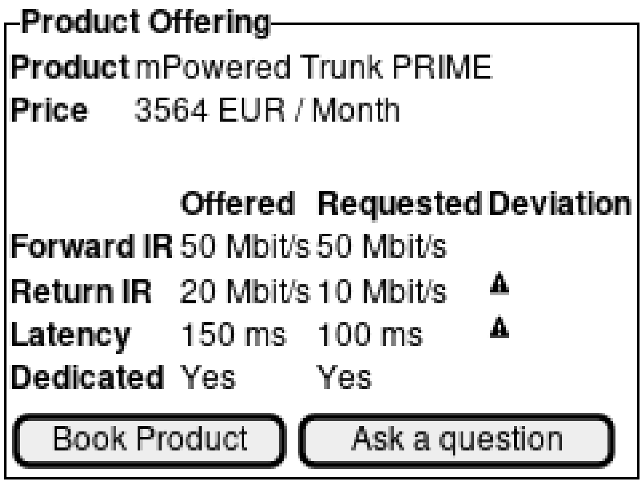


Product Offering

Product	mPowered Trunk PRIME	
Price	3564 EUR / Month	
	Offered	**Requested**
Forward IR	50 Mbit/s	50 Mbit/s
Return IR	🟩 20 Mbit/s	10 Mbit/s
Latency	🟥 150 ms	100 ms
Dedicated	Yes	Yes

Figure 3.28: Plain Product Offering with Comparison

The provider could flag the characteristics that diverge from the inquiry to help with the comparison. This would make it explicit, but requires significant changes in the API, because the catalog APIs can not be directly reused any more. It seems unreasonable, because the algorithm to find the differences is not too much different from comparing the characteristics directly. Also, the question of trust arises, because the provider could decide something is not a “real” deviation, and therefore skip the flag, while the customer sees it essential. This would then lead to the conclusion that a full check has to be performed, making the flag useless.

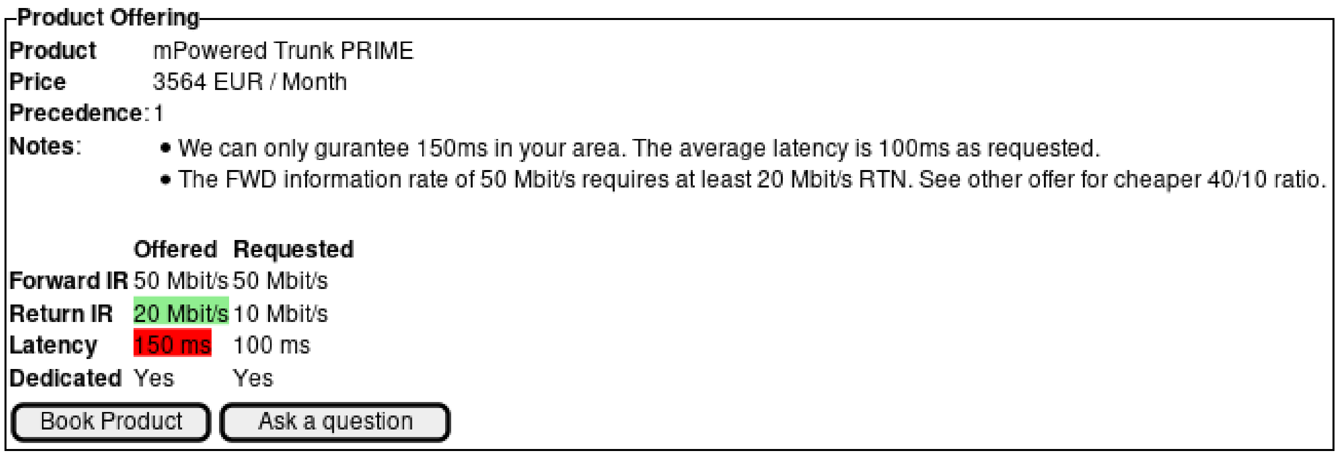


Product Offering

Product	mPowered Trunk PRIME		
Price	3564 EUR / Month		
	Offered	**Requested**	**Deviation**
Forward IR	50 Mbit/s	50 Mbit/s	
Return IR	20 Mbit/s	10 Mbit/s	⚠
Latency	150 ms	100 ms	⚠
Dedicated	Yes	Yes	

Figure 3.29: Product Offering with Boolean Indicators

The API could be extended with a free-text field on the level of product offerings. These would enable the provider to state an overall reason why the offer is included despite not exactly meeting the requested characteristics. Adding it to every characteristic is rarely useful because of interdependencies. For algorithm-based responses this field should be considered optional, because it can’t be expected to always return a concise reason (though possible under circumstances). Additionally, a new `precedence` field would allow the PSS to respect the ordering of the providers when sorting the list of results. As stated above, the HMI may still do a side-by-side comparison and highlight the differences.


Product Offering

Product	mPowered Trunk PRIME
Price	3564 EUR / Month
Precedence:	1
Notes:	• We can only gurantee 150ms in your area. The average latency is 100ms as requested. • The FWD information rate of 50 Mbit/s requires at least 20 Mbit/s RTN. See other offer for cheaper 40/10 ratio.

	Offered	**Requested**
Forward IR	50 Mbit/s	50 Mbit/s
Return IR	20 Mbit/s	10 Mbit/s
Latency	150 ms	100 ms
Dedicated	Yes	Yes

Figure 3.30: Product Offering with Free-Text Explanation

3.14.7 Implications for the Scope

The interface is not responsible for how a PSS handles inquiries and how the internal priority of resources matching the inquiries is set.

The interface is not responsible for a PSS-internal process like regional pool load balancing. The form neither of an inquiry nor of the offers in response has any influence on this.

Selection of offers is done within the PSS. However, the interfaces required for the communication to place an order are defined as part of the *product order management* API, see [PSI-TOD-03-02].

The interface has no influence on the capacity of a PSS to manage any amount of missions.

The phrase *eligible offers* leaves room for interpretation. Therefore, the interface will enable the governance to allow different approaches to define what is *eligible*. That is, some parameters might require a perfect match to be accepted, while others allow for ranges or are not restricted at all.

The interface to send an inquiry is not restricted to the user, but the governance can use it as well if the system implements the corresponding processes.

If a provider wants to send multiple offers as response to an inquiry, they can link the offers to each other. However, the PSS is responsible for showing, sorting, and filtering the offers for the user in a meaningful way, including if the offer perfectly or only partially matches the inquiry. An additional, optional field for the precedence of the offers from the provider's point of view is considered to improve this process.

The form of inquiry and response allow the PSS to compare them directly and highlight mismatching parameters. Additionally, the optional *notes* field can be used to add explanations for mismatches. Including this process in the interface by a boolean was considered but rejected due to the potential misuse.

The interface definition allows, via the form of inquiry and response, comparing parameters in different offers. Thus, this candidate requirement is considered as implemented by definition.

The *order management* API only allows the booking of complete offers, see [PSI-TOD-03-02].

The additional *notes* field in an inquiry response can be used by providers to provide a reason for accepting or rejecting the user's inquiry.

Additionally, the comparison allows users to inspect advantages one offering might have compared to another one.

3.15 Overbooking (V2.0)

3.15.1 Context and Problem Statement

The terms *overbooking* and *contention* are used ambiguously depending on where the user is coming from, i.e., it has a different meaning in the candidate requirements than in the context of (Gov)SatCom service provision. For the PSID project, we define overbooking in chapter Decision Outcome. This ambiguity leads to misunderstandings when discussing the topics and defining the scenarios in which it may occur.

For ESA, *overbooking* is a logical action that takes place at the moment a service is provisioned. If a service has been overbooked, contention may happen during operation as soon as multiple services are competing for the same resource. *Overbooking* is seen as a common practice for shared services.

For CSPs, shared services are subject to *oversubscription*. If a CSP makes the deliberate choice to offer more capacity than there is available, they say the services are subject to *overselling*. *Overbooking* is used by CSPs as a general term to describe situations that *might* lead to a contention situation.

The terms are used just slightly differently as they imply different contexts. *Oversubscription* or *overselling* represent a business view of the situation. These terms imply that potentially some elements of the service may not be fulfilled, if contention happens. *Overbooking* in contrast relates to a resources view of the situation, referring to the missing amount of resources to fulfill all requests.

However, a precise definition is required to omit further misunderstandings in the description of the processes.

The situation under discussion is depicted below in 3.31. For the sake of simplicity in the chosen example, *Product Offering A* corresponds to a committed L1 service, i.e., it offers only internet access via an IP trunk and the customer is responsible for the remote site connection including hardware. Since the service is committed, there is an instance *Service 1* in the inventory corresponding to the service specification *ServSpec*. The service specification itself is mapped with the *Product Specification A*, which is offered via the *Product Offering A*. However, the same scenario can be applied to any kind of service. Note that not all parameters are depicted.

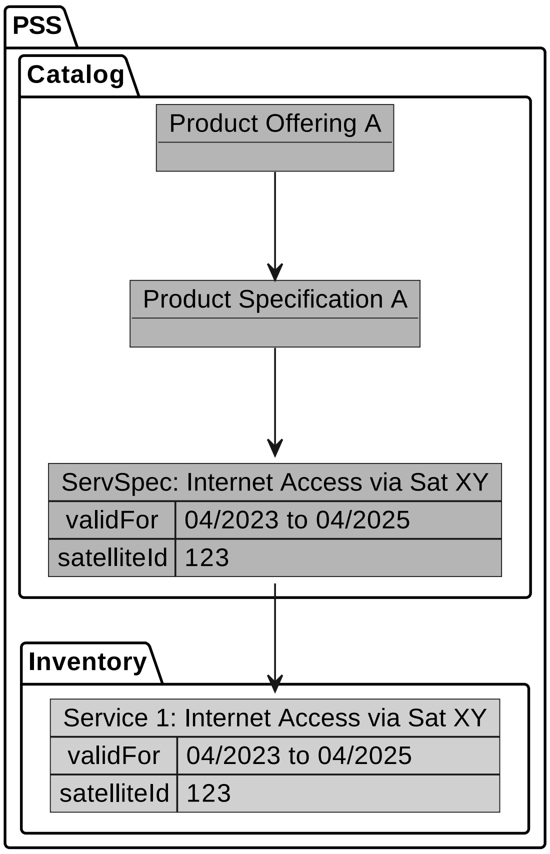


Figure 3.31: Exemplary product offering tree of an L1 service used to discuss *overbooking* and *contention*.

The process as depicted in 3.32 and described in the following *can* lead to oversubscription, overbooking, and contention situations, depending on more details of *Service 1*, the priorities of *Customer A* and *Customer B*, as well as the implementation details of the PSS.

Customer A sends an inquiry in January for internet access between February and December to the PSS and gets *Product Offering A* as a result from the PSS. *Customer A* then decides to order the product. Thus, the PSS creates a sub-instance of *Service 1*, i.e., *Service 1.1*. In February, the service is started, and *Customer A* is notified about the activation by the PSS. In March, *Customer B* enters and sends a similar inquiry for internet access between June and September to the PSS that would give the same *Product Offering A* as a result. Depending on the capacity of the resources behind this offering, the types of services and resources involved, and the implemented processes, different scenarios can arise. The PSS can either reject the inquiry, i.e., if the service is not available, there is no way to offer it a second time, or offer it. The details of the different scenarios are discussed below.

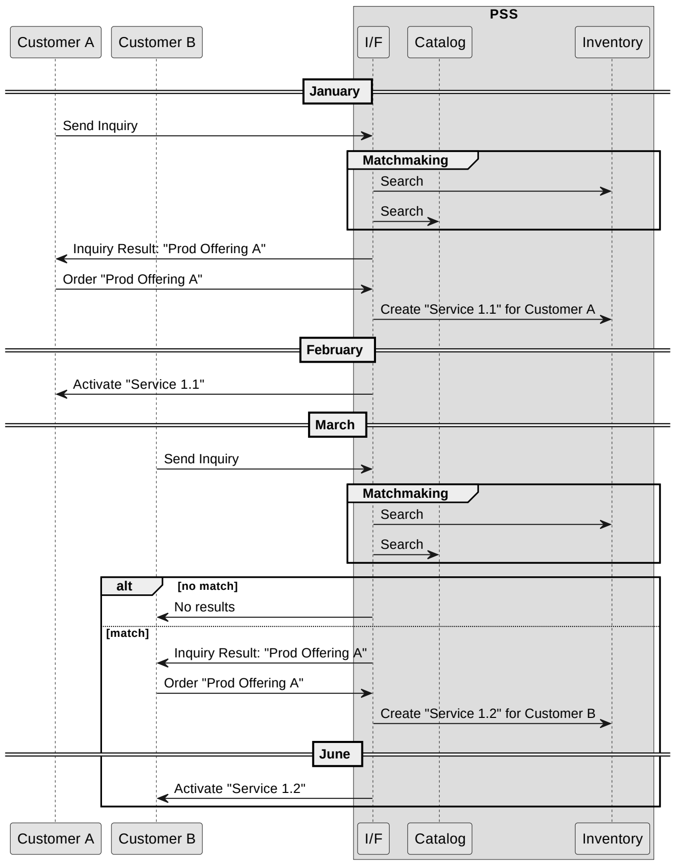


Figure 3.32: Basic process that can lead to *oversubscription*, *overbooking*, and *contention*.

The process is depicted from another point of view in 3.33 with more details on the inquiries, the product offering tree, i.e., the underlying product specification, service specification and service instance. Note that this depicts a committed service, therefore, there is already an instance in the inventory from which the instances for *Customer A* and *Customer B* are derived. For non-committed services, the instances are derived directly from the service specification in the catalog.

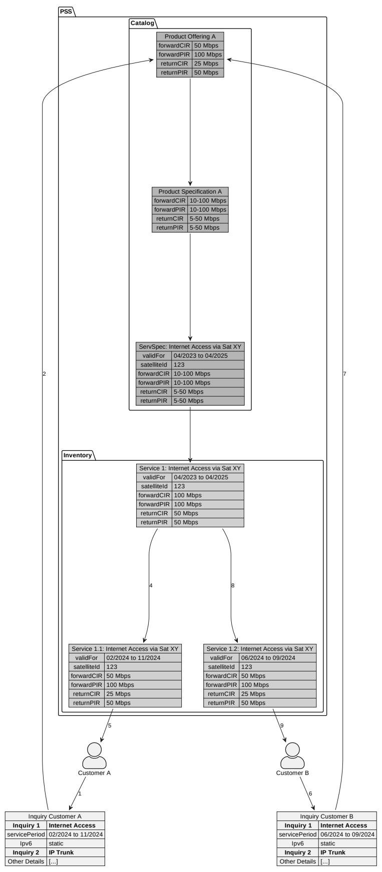


Figure 3.33: Combined view of the product offering tree and the basic process that can lead to oversubscription, overbooking, and contention.

Additionally, this discussion is tightly coupled to the terms *Committed Information Rate* (CIR) and *Peak Information Rate* (PIR) which are distinguished for managed services (L1, L2, and L3). The CIR defines the guaranteed capacity that is contracted to be available at any time for a given service. In contrast, the PIR gives the maximal available capacity that might be available sometimes for that service. For each service the PIR can be as high as the total available capacity on the resources implementing the service. That is, the extreme case that all services have a PIR equal to the total available capacity at the same time is totally valid, although they cannot make use of it at the same time. With CIR and PIR, two different types of services can be defined:

The definition on shared and dedicated services can be extended to all kinds of items, i.e., shared and dedicated resources as well as shared and dedicated products.

This decision record shall collect the different scenarios where any party might use the terms *oversubscription*, *overbooking*, and *contention* and define a common wording for this project.

3.15.2 Decision Drivers

3.15.3 Decision Outcome

The following definitions have **no influence** on the design of the API and can be used to define proper processes for the PSS implementing PSI:

Note that overbooking and overselling are similar concepts in that they both involve selling more than can be immediately accommodated or delivered, but the specific context and reasons for doing so can vary.

3.15.3.1 Oversubscription

Oversubscription is the basis for all non-dedicated telecommunications services and can be used to maximize the usage of a shared common resource. In general, it is defined by the ratio between the allocated capacity per customer and the available capacity per customer. Therefore, the oversubscription model assumes that statistically only a fraction of the customers will attempt to utilize their allocated capacity (e.g. bandwidth) simultaneously. A common example is the internet connection on the mobile phone: There is a maximum guaranteed up- and download speed, however, if there is a peak in the demand for a specific location (e.g., a sport event) only a reduced capacity is available for each individual.

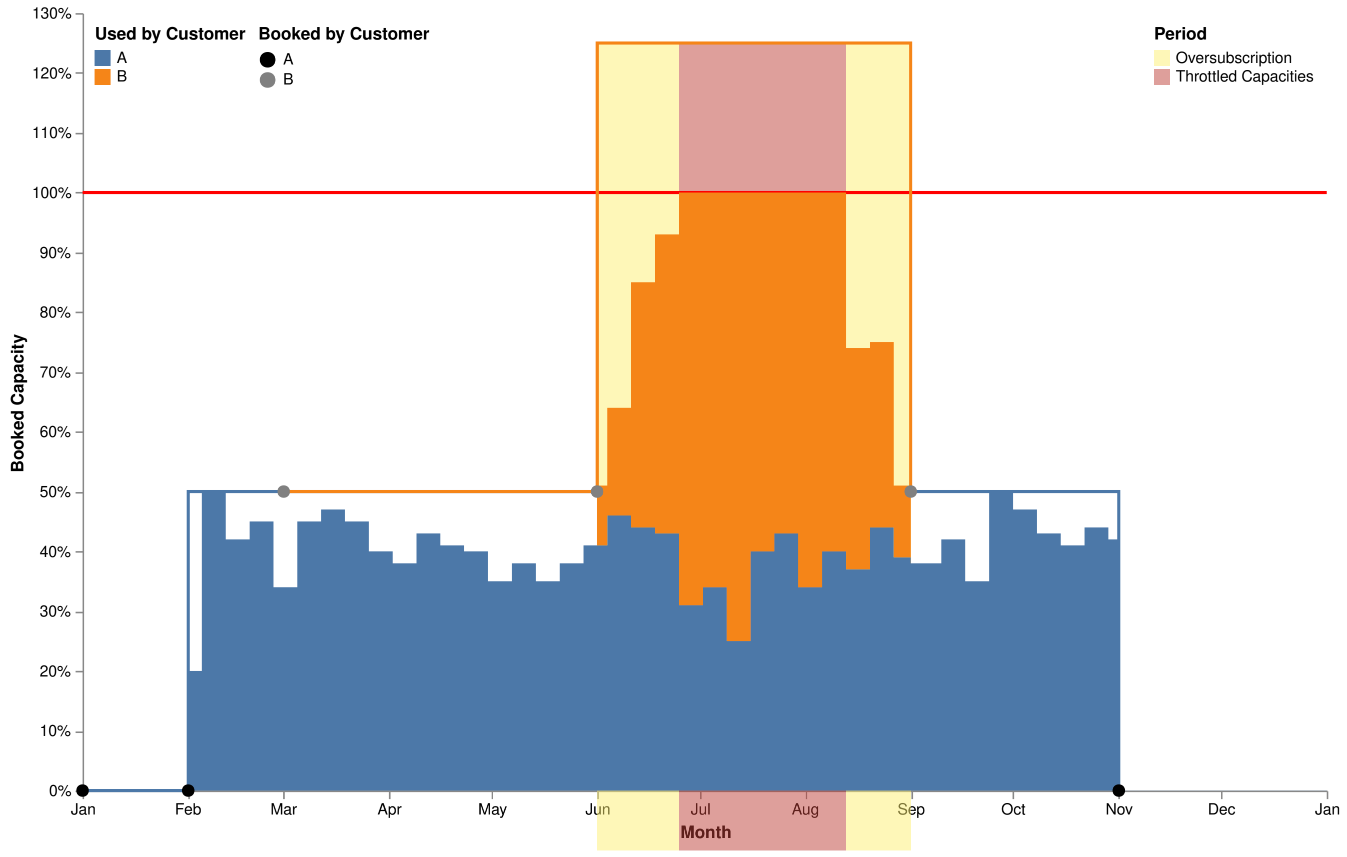


Figure 3.34: Oversubscription.

In 3.34, *Service 1* can be shared among multiple customers, e.g., it is a pre-configured iDirect service, where the underlying resource can manage up to 10Mbits download and 5Mbits upload. *Customer A* books the service from February to November with a CIR of 25% of the total capacity and a PIR of 50% of the total capacity. The PSS implements this as *Service 1.1*. *Customer B* then books the service from June to September with a CIR of 50% of the total capacity and a PIR of 100% of the total capacity. The sum of the PIRs is now 150%. This results in a throttled capacity, i.e. the bandwidth that was foreseen to be needed cannot be distributed as initially requested. However, there is never a violation of the SLAs. For example, if *Customer A* does not use any of their guaranteed capacity, *Customer B* can use it completely.

3.15.3.2 Overselling, Overbooking, and Contention

Overbooking and overselling are terms often used interchangeably, but they can have slightly different meanings depending on the context. In general, both concepts involve selling more of a product, service or resource than can be accommodated or delivered, but the specific usage can vary by case.

There are services that do not have a CIR defined in their SLA, e.g., services offered by Starlink or OneWeb that offer only a PIR, i.e., a connection with a data rate up to a specific amount. These services are better described by *overselling* than *overbooking*. Another example is a company selling more licences of a software than they can effectively support with customer service and technical assistance. If too many instances of such a product are sold, the underlying resources are stretched out across multiple customers and the services cannot be provisioned *efficiently*.

Overbooking refers to a provider offering more instances of a resource, service, or product (collectively, any *item* in a catalog or inventory) than there are available in stock. This is not a problem until the items actually need to be provided, i.e., a system can allow overbooking as a process, but cannot work properly once it needs to provide them. If this is the case, a contention situation arises.

For example, an airline can sell more tickets for a plane than there are seats available (overbooking). This is not a problem until more passengers want to board the plane than there are actual seats (contention). As long as not all passengers show up to the flight, overbooking as a process is allowed and leads to no problems. However, as soon as more passengers want to board than there are seats on the plane, the system fails and mitigation for the overbooking needs to be initiated, e.g., some passengers need to get alternative flights offered.

Note that the contention situation on planes are sometimes called *overselling*. We will use the word *contention* here instead to omit ambiguity.

In case of communication services, overbooking is usually referring to the CIR, and is not taking the PIR into account. This is illustrated by the following example, which assumes a service with 100 Mbit/s CIR available. Note that the example below does not necessarily need to refer to the CIR of a service. E.g., for L0 services, the allocated fractions of bandwidth (identified by start and end frequency) would be used instead.

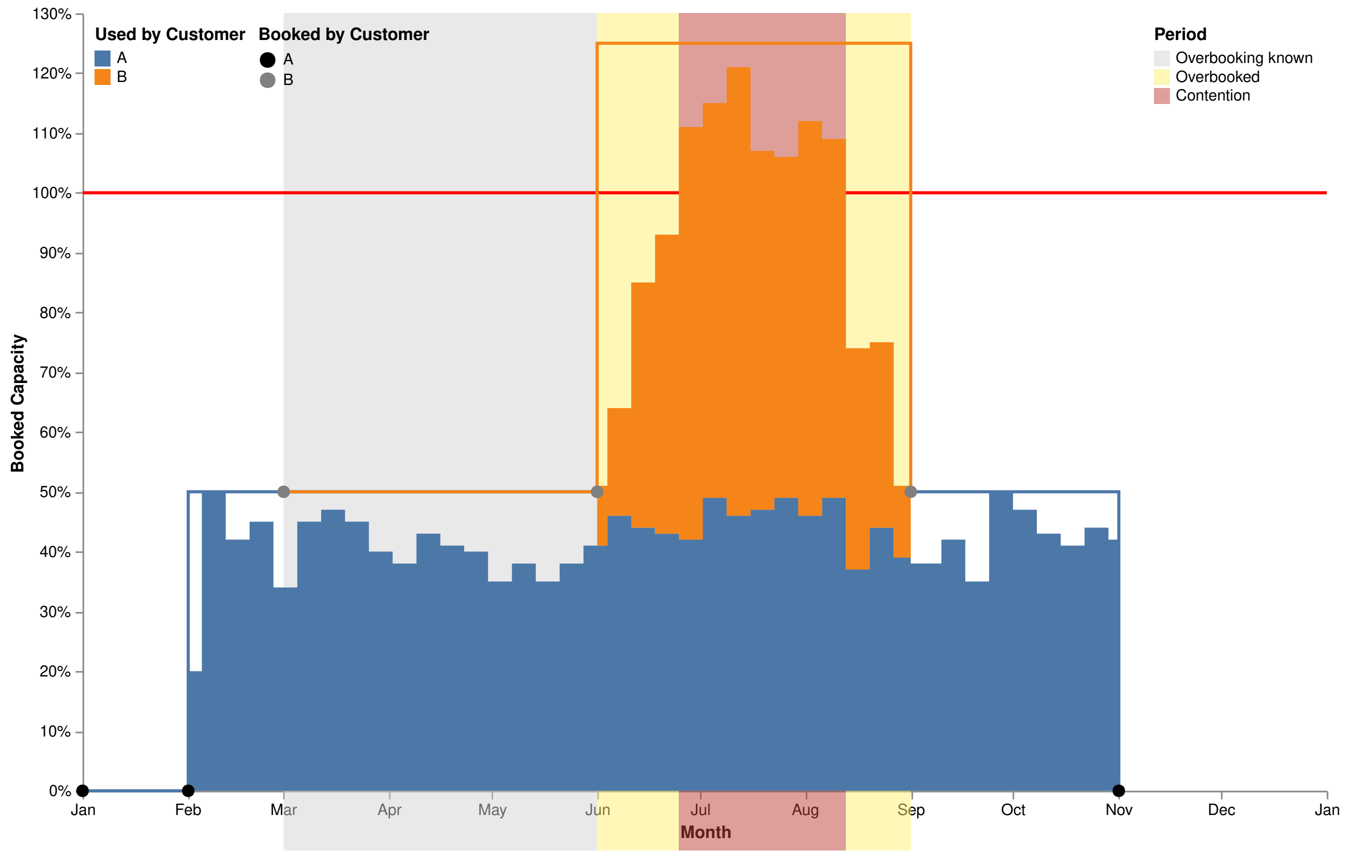


Figure 3.35: Overbooking and Contention.

In 3.35, *Service 1* cannot be shared among multiple customers, i.e., it is a dedicated service. *Customer A* books the service from February to November with a CIR of 50% of the total capacity. The PSS implements this as *Service 1.1*. In March, *Customer B* then books the service from June to September with a CIR of 75% of the total capacity.

Between March and June, the service is overbooked, i.e., the PSS knows that there is an upcoming contention situation and has time to account for this, e.g., by ordering more capacity. If the PSS fails to do so or decides not to handle the overbooking, a contention situation arises, as the guaranteed capacity is at 125% of the total capacity. This contention situation lasts for the whole time *Service 1.2* is provided.

3.15.3.3 Priority and Pre-emption

In general, overbooking as a process is a business decision to take a calculated risk, i.e., not being able to provide all items offered. The main reason to allow overbooking is to maximize the return on investment by maximizing the usage of the available capacity of an item. However, overbooking and resulting contention situations might be a result of *force majeur*, e.g., a natural catastrophe can destroy parts of the communication infrastructure and reduce the total capacity. Independent of the reasons for overbooking and/or contention situations, i.e., if it is an expected or unexpected situation, the API needs to implement a way to handle mitigation strategies.

Depending on the case, there are different strategies that can be implemented when an overbooked system becomes contended.

For dedicated items that cannot be degraded, there are basically only two solutions:

Note that it does not matter if the item was sold or reserved and how the state was achieved. Cancellation of an order might result in penalties, bad reputation, and other negative consequences. For example, it may also involve logistical problems when hardware is already shipped, which go beyond the scope of this decision.

For all other items, there is the additional option to offer a degraded service, e.g., with lower bandwidth to some or all customers. Note that the degradation of service can be combined with the cancellation of services, e.g., when a transponder of a satellite fails.

Depending on the product that is involved, the PSS and/or provider can implement priority for the customer, i.e., customers with higher priority have a higher probability to keep their service active. Priority can be implemented on different levels:

For example the first setup for priority is implemented in terrestrial networks in cases of emergency, in the US with the *Nationwide Wireless Priority Service* and in the UK with the *Mobile Telecommunication Privilege Access Scheme*. Communication priority is also implemented on planes with the SATVOICE protocol, which grants the different categories of communication different priorities, i.e., in descending order emergency, operational (high), operational (low), and non-operational. The last setup for priority can be implemented e.g. as Value-Added Services.

Based on the priority, some services may be subject to pre-emption. That is, if a customer with higher priority needs to access a service that is already booked, they might have the right to take over some or the whole capacity.

In 3.36, the contention situation discussed above is resolved by reducing the capacity of *Customer A* for the duration of *Service 1.2* of *Customer B* to only 25% of the total capacity. This only requires a change request by the PSS to be sent to the provider to implement this and a notification towards *Customer A*, which is already part of the API. This can only be implemented if the SLAs allow for such a degradation, e.g., based on priority.

Note that this requires the same data processes as for the defragmentation described in the MADR Bandwidth (De-)Fragmentation.

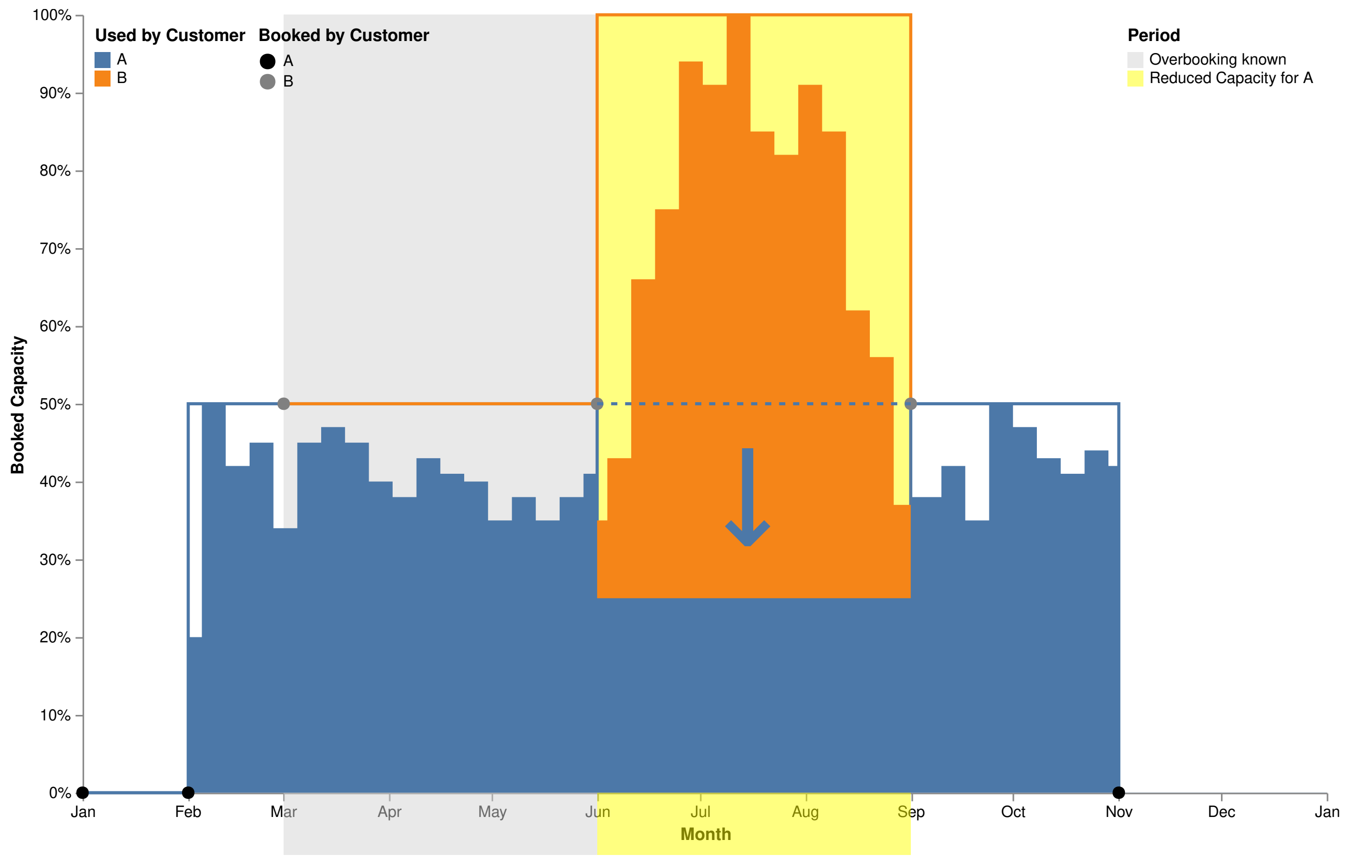


Figure 3.36: Pre-emption.

However, all these mitigation strategies can be implemented in a PSS and do not require any additional options in the API.

3.15.3.4 Influence on Pools and Pooling

Assume there are two pools, *Pool A* and *Pool B*, that share a common product offering tree like the on depicted in 3.31. Furthermore, assume that *Customer A* has only access to *Pool A* and *Customer B* has only access to *Pool B*. Again and as described above, both want to order the same service (which they can as the product offering is accessible to both). Then, this is the same situation as it would be without pools and all the discussions above apply without any change. This applies to both committed and non-committed items: For a non-committed resource specification, a resource instance is created when the corresponding offer is booked. For an offered resource, a sub instance of this resource instance is created. In both cases, that new instance is no longer part of the pool(s) but considered *private* to the customer. The resource specification or offered resource instance can be checked against its children for availability, even in the case of pre-emption. Therefore, the API does already support pools and pooling and needs no further change.

Note that this assumes that the PSS can assign the same product offering tree to multiple pools. This does make a lot of sense, though. For example, the PSS governance can decide to have a regional pool for the northern part and another regional pool for the southern part of Africa. Services on a GEO satellite with a footprint that covers Africa as a whole could (and should) be assigned to both pools, although booking in one region might reduce the availability in the other. This is subject to demand and supply management as well as forecasting and has no influence on the APIs.

3.15.3.5 Conclusion

Overbooking can only be managed by the governance of a PSS if the items that are subject to overbooking are committed to the system. Non-committed items are assumed to be handled almost exclusively with the PSS acting as a service broker. However, to manage overbooking, the items need to have corresponding entries in their respective inventory. If a committed resource is configured in such a way that the relevant parameters can be monitored in the inventory, the governance can implement processes to allow overbooking from within the PSS for those items as well as add corresponding mitigation strategies.

The possibility of overbooking an item needs to be confirmed during the accreditation process. To do so, the governance shall create a list of KPIs that can be monitored within the PSS, e.g., machine-readable parameters of the SLS/SLA, the cancellation rate of reservations, etc. The governance can enforce this by marking those parameters as, e.g., *mandatory for overbooking* in the respective templates, available via the APIs PSID633 and PSID634 or via the corresponding JSON schema.

Note that the governance has full control over the templates and thus the parameters they want to define to allow overbooking. The flexibility of the APIs enables the governance to do so, but does not enforce any process here. However, it is recommended to include this in the accreditation process of the items if overbooking is considered.

Similar to non-committed items, overbooking cannot be managed by the governance if the items are collected via a distributed matchmaking as explained in the [PSI-TAD], e.g., in the chapter on inquiries. However, the governance can implement an accreditation process that allows parties that are able to do distributed matchmaking to offer overbooked items. The processes and contracts necessary to do so are not subject of the interface, but require additional business processes.

When allowing to offer overbooked items via distributed matchmaking, the governance needs to monitor the impact of overbooking on their customers, e.g., failure or degradation of services due to a contention situation. Therefore, this process has to be based on the ability to measure certain KPIs. Again, this can be configured by the governance and enforced by an accreditation process. Thus, the interface is not involved besides offering the option to implement such a set of parameters.

3.15.4 Compliance

The description of oversubscription, overbooking, and contention will be updated in the [PSI-TAD].

Configuration changes will be included in the case study, hinting at the possibility that overbooking and the resolution of contention situations can be a reason to trigger them.

Additionally, the transmission of KPIs in the scope of distributed matchmaking needs to be taken into account as part of implementing the APIs for *Service Quality Management*.

3.15.5 CIA Comparison for All Given Options

Not applicable.

3.15.6 Implications for the Scope

The candidate requirement assumes that a product can only be offered in multiple regions by duplicating it in the corresponding regional pool. However, the current design of pools does not necessitate this, as a product offer or specification and its underlying services and resources can be assigned to multiple pools. Pools are assumed to function as follows: A product specification can be included in one or more pools. Every service specification and resource specification required to implement the product specification must be in the same pool(s) to ensure visibility. This is also true for service specifications based on a resource specification. Conversely, a resource specification can be in more pools than the service specification and the product specification that depend on it.

Product offers can only be in a pool where all product specifications they combine are available. If a product offer is committed to the system and available in multiple pools, the instances of the resources and/or services can be restricted to fewer pools than their respective specifications. For a committed product offer, this means that all specifications and instances have to be in that particular pool. Therefore, overbooking caused by booking the same product from multiple pools at the same time cannot occur.

Overbooking on a global or multi-regional scale is complex and occurs at the service level and on the provider side. Thus, it only applies to a PSS that acts as a Communication Service Provider (CSP) and only for specific services. Examples where geographical overbooking can occur include global services like Inmarsat Global Xpress or SES Skala Global Platform, roaming services for ships where the position and thus the available footprint is unclear, footprints that cover multiple time zones, and provider-side capacity pooled services that can use multiple satellites and/or beams.

Overbooking, as discussed in this MADR, can only be managed by the provider, as it requires an overview of the congestion factor and satellite resource monitoring, both pieces of information that are most likely not willingly shared with a PSS. Overbooking occurs on two levels: the commercial level, where monetary risks need to be evaluated and managed, and the technical level, where capacity and available resources need to be monitored and managed. Both include expectation management for the customers and are commonly part of the Service Level Agreements (SLAs).

Some services can allow overbooking, but some exclude this option. For example, customer-based services have a fixed, committed data rate that cannot be overbooked. Pooled services, on the other hand, might allow for it if they make use of a congested capacity pool. There is one option to allow a PSS to actually manage overbooking: If it acts as a provider and books a capacity with an activation service. Then, the provider can introduce limiting factors, such as the number of customers that can book this (this could be restricted by the hardware, e.g., iDirect Hubs with a maximum of ten users). Additionally, the provider must allow the PSS governance to do overbooking on this service.

Internal matchmaking with respect to a priority system: How resources are assigned cannot be specified on the interface level.

This candidate requirement defines two scenarios: First, “overbooking” of the peak information rate (PIR), which can be defined as oversubscription. Oversubscription is a common model in both, terrestrial telecommunication (contended services) and in (Gov)SatCom (shared services). Second, “overbooking” of the committed information rate (CIR), which means that the PSS promised more bandwidth than available. This can be defined as overselling, which needs mitigation. Oversubscription can lead to overselling, if the sum of the CIR exceeds the total available capacity. In principle, both scenarios need to be handled in the PSS internally. However, the interfaces are capable to support these processes, i.e., the interfaces can handle the transmission of the required data.

The interface allows including CIR and PIR in the respective service specifications, and thus defines the *contention ratio* for each of them. This specification needs to flow down into the corresponding SLS and SLA. However, this is out of scope of the interface definition, as it can only allow to include the required parameters. Enforcing the contractually agreed contention ratio means that any given product in the contract defines, if overbooking (either as *overselling* for *dedicated services* or as *oversubscription* for *shared services*) is allowed or not. The PSS can only enforce this in two ways: First, if the underlying items are committed or owned by the PSS, the governance has control over them and can manage overbooking itself. Second, if the underlying items are non-committed, the governance can only manage this by the accreditation process of the corresponding products, i.e., allow a provider to overbook a given product line or not. In summary, this candidate requirement is implicitly implemented via these options. Nevertheless, enforcing some PSS internal process is out of scope for this project.

The *contention ratio* for *shared services* can be over one, as this is the usual business case utilizing the concept of *oversubscription*. For any other product, the PSS is responsible for calculating this based on the number of committed resources and booked products. The interface can only allow the communication and transfer of the data required to implement this kind of monitoring. The contention ratio is a derived value that is not required for any process involving external communication. This requirement is related to the governance interface only. However, the [PSID688 Event Management] can implement a notification event that informs the governance when an instance of a resource, service, or product is booked multiple times.

From a PSS's perspective, a product offering or product specification can never be overbooked due to the fact that it is assigned to multiple pools. Therefore, all other discussions on overbooking do apply. The interface enables the PSS to implement a corresponding process.

The calculation of the overbooking ratio/contention ratio is a PSS internal process that does not require any interface. The interface allows to transmit the data required to do this calculation, i.e., the *committed information rate* (CIR) and the *peak information rate* (PIR) of *shared services* in case of *oversubscription* and for every product in case of *overselling*. However, to do this calculation, the PSS needs to store all the information on a product where oversubscription (and overselling) are possible, as the state of being overbooked accumulates over time and is not tracked by the stateless interface. The interface does not store any information and therefore is not capable of such a calculation. However, the information provided via the interface enables the PSS to implement this process.

This candidate requirement basically states that a PSS shall be able to offer *shared services* that utilize *oversubscription*. This is enabled by the interface, but not enforced, since this is a PSS internal process.

The calculation of the overbooking ratio/contention ratio of an oversubscribed product is a PSS internal process that does not require any interface. The interface does not store any information and therefore is not capable of such a calculation. However, the information provided via the interface enables the PSS to implement this process.

Assuming that "subscribed contractual option" refers to *shared services*, this candidate requirement states that a capacity of such shared services shall be manageable by its respective community, i.e., the shared ratios, CIR, PIR, and how the capacity is chunked shall be in the responsibility of the community itself, e.g, with an administrative board. In that case, *oversubscription* can indeed be managed by this administrative board, and even if *overselling* happens, this is a problem of the community, not of the PSS or the provider, as this is only managed internally. This means, that a PSS shall be able to automatically forward configuration files to a hub that is providing shared service, e.g., to configure an iDirect Hub. Such a file can be exchanged via the [PSID667 Document Management].

The calculation of the community overbooking ratio/contention ratio is a PSS internal process that does not require any interface. The interface does not store any information and therefore is not capable of such a calculation. However, the information provided via the interface enable the PSS to implement this process.

The PSS shall flag all cases of overbooking, i.e., both on a service level and on a community level. Flagging has to happen based on the state of the system, i.e., internally, and without the need for any communication. Therefore, the interface is not involved.

The interface [PSID688 Event Management] can implement a notification event that informs the governance and/or the community, e.g., via its administrative board.

In case of overbooking and/or degraded services, the governance or, in case of a community, a community administrative board should be able to give some users priority. This can be handled in multiple ways, e.g., a community-internal pre-emption system, an ordered list of all members that automatically resolves those contention situations, groups with the same priority, etc. There are already a lot of protocols to handle priority, but none of them is generally accepted or can be applied to every situation. They are a PSS internal process, the generation of such priority systems and the corresponding assignment are subject to the PSS governance and might be implemented on demand from a user group. However, the data to communicate priority not only internally but with multiple systems involved can be added to multiple interfaces, e.g., if pre-emption **and** distributed matchmaking are implemented to function together. Therefore, at least the [PSID001 Customer Inquiry Management], the [PSID632 Party Management], and the [PSID639 Resource Inventory Management] can add corresponding features that allow for a reasonable process. Note that this requires a standardized priority system across multiple systems to work properly, which this project cannot provide. Again, the interface can only enable this process, but the details are up to the implementation.

Pre-emption is a PSS-internal process, as it handles the states of orders. However, the interface manages the "incoming higher priority service request" just like any other request. To implement the priority system, the interface needs to communicate the corresponding party profile for a request. The parties need to define a default profile, which can be generated by the governance during the accreditation process. Additionally, the resources that are booked, i.e., a subset of resources in the inventory, need to know who is allowed to request them in case of pre-emption. Therefore, this candidate requirement is partially implemented, while pre-emption itself is out of scope for this project, but it is enabled by the interface. Changes within an existing order are subject to the contract.Technically, such a change and creating a new order do not make a difference for the customer.

This candidate requirement concerns the governance and user interface When the system monitors the corresponding event, it should be able to generate another event via the [PSID688 Event Management] that requires an approval to implement a specific solution for a contention situation.

3.16 Use OpenAPI 3

3.16.1 Context and Problem Statement

We currently use TMF interface definitions written in OpenAPI 2. The newly created Inquiry API uses OpenAPI 3 and the Document API is converted to use new features. It may be sensible to convert the others, too.

3.16.2 Decision Drivers

3.16.3 Considered Options

3.16.4 Decision Outcome

Chosen option: "Upgrade to OpenAPI 3", because we need some of the new features (i.e. improved support for binary data and discriminators). It is foreseeable that other APIs may need those as well. Having all files written in the same format improves the overall readability. Since we already have the means to convert the definition files and all tools accept both formats, the additional workload to update the patch files is acceptable.

3.16.5 Compliance

All interface definition files are upgraded as part of the merge request containing this decision. New APIs will be automatically upgraded by the transformation script.

3.17 Use Bundled Groups from TMF620 V5

3.17.1 Context and Problem Statement

PSID currently utilizes the TM Forum *Product Catalog Management API* as described in TMF620 V4.1. Part of this is the *Product Offering* entity, which encompasses the option to define bundled offerings. Up to version 4.1 of TMF620 these bundles are built from a list of offerings, where each list can have a min, max and default cardinality.

A disadvantage of this simple, list-oriented definition is the lack of an option to build selection groups with a cardinality - e.g. “the customer has to select exactly one of the offered terminals”. To address this kind of problems, TMF620 V5, which is currently in a pre-production phase, will introduce bundled groups with optional cardinalities on each of them.

3.17.2 Decision Drivers

3.17.3 Considered Options

3.17.4 Decision Outcome

Chosen option: Patch TMF620 V4.1 *ProductOffering* based on TMF620 V5.

The review of the catalog data together with the providers revealed that selection groups / bundles are a common approach to structure the offerings. However, the possibilities offered by TMF620 V4.1 are not sufficient to meet these requirements.

TMF622 V5 tackles the gaps of the list-oriented implementation. Unfortunately, TMF620 V5 is currently in a pre-production phase with ongoing changes. For this reason, it is not advisable to switch to V5 now. But we could reduce updates to the important part only and pull up just the PSID *Product Offerings* from V4.1 to the latest state of V5.x. This should keep the version used as compatible as possible with future updates.

The latest relevant changes in *Product Offering*, see TM Forum Ticket AP-4271, introduced two new classes:

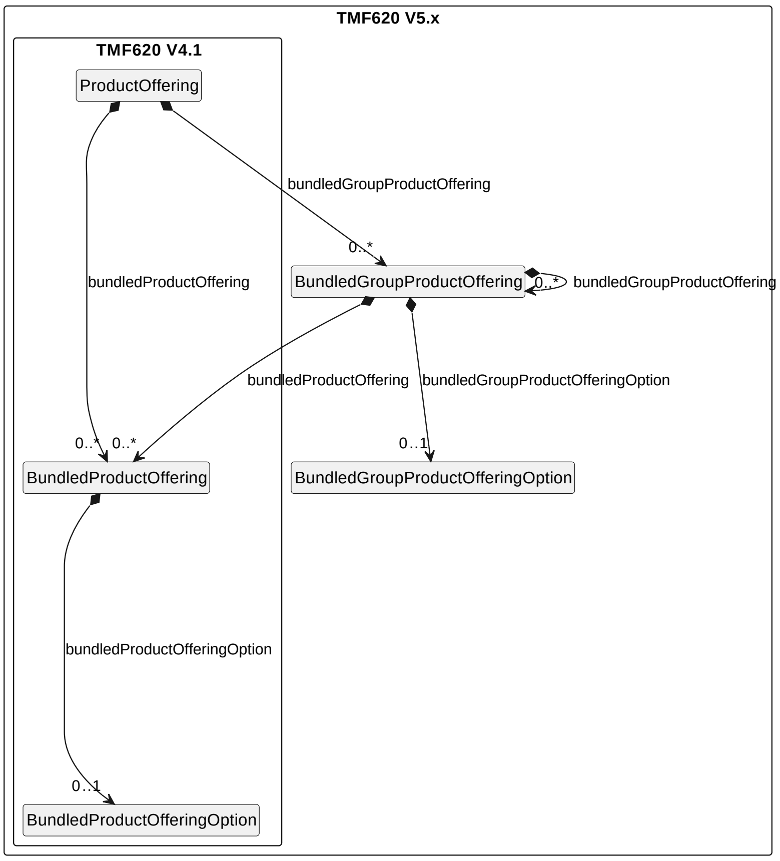

TMF620 V5.x
TMF620 V4.1
ProductOffering
bundledGroupProductOffering
0..*
bundledProductOffering
BundledGroupProductOffering
0..*
bundledGroupProductOffering
bundledProductOffering
bundledGroupProductOfferingOption
0..*
0..*
0..1
BundledProductOffering
BundledGroupProductOfferingOption
bundledProductOfferingOption
0..1
BundledProductOfferingOption

Figure 3.37: TMF620 V4.1 vs. V5.

The classes of V4.1 saw only very minor changes. Only the optional associations from V4.1 classes to the new V5 classes were added. This maximizes backwards compatibility.

Considering that PSID wants to stay close to the TM Forum standard and knowing that TM Forum will improve the schema in the upcoming version, PSID should follow this solution and avoid proprietary attempts.

3.17.5 Compliance

The interface definition of *ProductOffering* as part of the *Product Catalog Management API* will be updated following TMF620 V5, complemented by the mock-up implementation. The ICD documentation will reflect the improved schema.

3.17.6 Implications for the Scope

The new TMF620 V5 approach of *BundledGroupProductOfferings* will improve the flexibility on how to group and structure offerings.

3.18 Quality and Performance Management API

3.18.1 Context and Problem Statement

As part of the ongoing roadmap analysis for the second half of the project, *TMF657 Service Quality Management* was identified. This API should implement the requirements for link status monitoring and for monitoring the service KPIs into the solution. In the course of the analysis, however, it became apparent that these requirements break down into several problem domains, which only partially overlap. In particular, it became apparent that the interface is not suitable to implement reporting and alerting. Therefore, the requirements in this MADR were analysed again in order to find a more suitable solution.

3.18.1.1 Definition of Quality Terms

In order to come to a decision, firstly, the terms have to be defined.

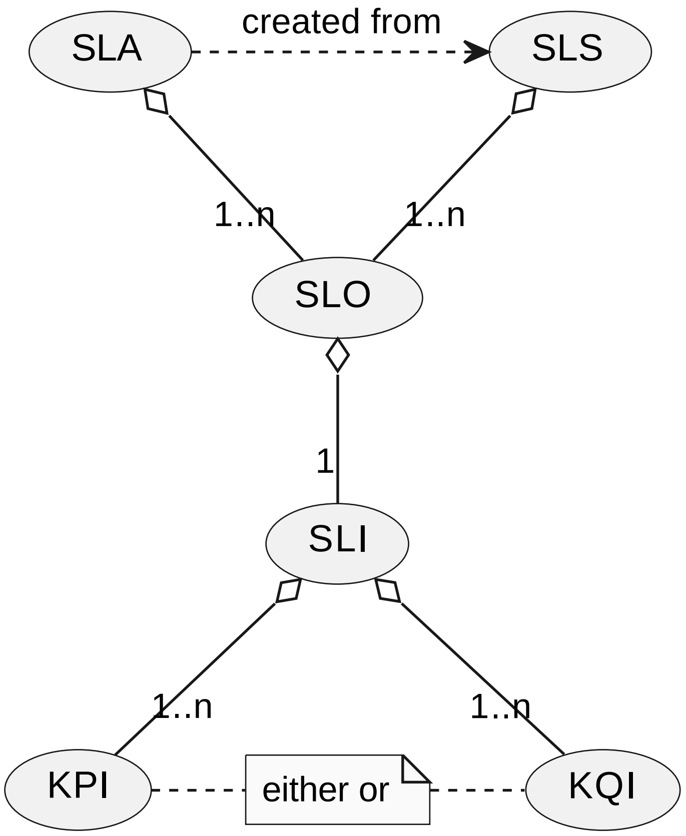


Figure 3.38: Quality Terms.

3.18.1.1.1 Key Performance Indicator - KPI

A KPI is a measurable and quantifiable metric. KPIs can be a directly measured value, e.g., latency, packet loss, and throughput. They can also be derived from multiple values, e.g., service availability over time. They exist in the system and can be recorded by a data acquisition system into time series. Their existence is not necessarily bound to a specific product or service.

3.18.1.1.2 Key Quality Indicator - KQI

A KQI is a measurable and quantifiable metric, similar to a KPI, but focused on the quality of service. Usually, it is harder to quantify the quality than it is for the performance. In contrast to the KPI, they are commonly bound to a specific product or service.

3.18.1.1.3 Service Level Indicator - SLI

An SLI is a metric definition of the performance of a service or a system. It is a foundation for both SLOs and SLAs as they provide the quantitative measures used to evaluate service performance and reliability. SLIs form the base of the hierarchy as they represent the raw data used to gauge service performance. SLIs are usually bound to a KPI.

3.18.1.1.4 Service Level Objective - SLO

An SLO is an agreement **within** an SLS or SLA about a specific metric with a given value range, granularity, and/or domain. Thus, SLOs set objectives for a service's performance. In other words: SLOs build upon SLIs by setting specific, measurable targets for those performance indicators.

3.18.1.1.5 Service Level Specification - SLS

An SLS is a collection of SLOs that *can* be applied to a service. They are part of the service specification and the basis for the SLA of a service instance.

3.18.1.1.6 Service Level Agreement - SLA

An SLA is a legal agreement between the business and the customer that includes a reliability target and the consequences of failing to meet it. It is at the top of the hierarchy, incorporating multiple SLOs into a formal, legally binding contract that defines the expectations and consequences for both the service provider and the customer.

3.18.1.2 SLI-SLO-SLA Interdependency

SLIs ensure that performance data is measurable and comparable through all providers, users, and other stakeholders of the system. By setting specific standards and expectations for service performance and reliability, SLAs and SLOs help service providers prioritize and allocate resources. They provide a clear understanding of what is expected of the service provider and what the consequences are if the agreed-upon standards are not met. This allows service providers to allocate resources effectively to ensure that they meet the agreed-upon standards and avoid penalties or other consequences. Additionally, SLOs help to prioritize and allocate resources to achieve effective capacity utilisation.

3.18.1.3 SLO Compliance Monitoring

Some SLO parameters can be easily monitored for compliance, e.g., the connection status itself. Many other SLO parameters are harder to monitor, e.g., if they are time-dependent. Time-based SLO parameters are commonly used to measure the performance and reliability of a service, e.g., It is obvious that not all time periods on which these parameters are based are deterministic. Thus, a consideration of whether an SLO has been adhered to can only take place retrogradely after the specific time period is known. For example, this can be the time frame around an incident or the mission duration.

For this reason, it makes sense for service providers to record the corresponding, required data “in stock”. The interface must allow access to this data, but not dictate the data recording itself. The interface must allow reports on the SLIs of a contract over a specific period of time to be retrieved from the provider.

A clear and objective definition of SLI parameters is key. Parameters that cannot be directly influenced by the provider should be avoided.

If, for example, a latency is defined as a “ping latency” on a certain server, it must be ensured that both provider and customer ping the same server. This ensures that the same measurement value is described from the customer’s perspective and the provider’s perspective. A latency to an external server, which may be measured by customers, can deviate considerably from the latency measured by the provider. However, this is within the sphere of influence of the provider.

Of course, given the nature of a satellite link, not all of these parameters can be ruled out.

3.18.1.3.1 Monitoring and Reporting

Monitoring and reporting are two related, yet distinct concepts. In summary, monitoring is the process of tracking and collecting data points to identify problems, while reporting is the process of organizing and presenting that data in a way that provides information over a greater time span.

3.18.1.3.2 Monitoring

3.18.1.3.3 Reporting

3.18.1.3.4 Alerting

3.18.1.4 OSS/BSS and KPI Reflection

The existing Operations Support Systems (OSS) and Business Support Systems (BSS) of (Gov)SatCom providers already give a range of KPIs to help satellite communication companies track their performance and identify areas for improvement. Such systems measure, e.g., raw signal characteristics, signal-to-noise ratio (SNR), and signal spectrum analysis of the carrier. Thus, service quality indications can be **derived** from this monitoring. This results in following types of KPIs:

And example is “latency”. As long as the signal monitoring does not indicate a problem with the link, the provider will assume that the latency of the link is within limits. But as soon as a customer complains about latency, the provider can measure the latency on the link. This is done on-demand and not permanently, as measuring the latency creates unnecessary traffic on the link. In this example, “latency” is a derived KPI in the general case, but can become a measured KPI. The SLI will reflect that. In the case that the customer complains about an SLO violation, the service provider will measure the true SLI actual value (the latency KPI) for this particular case. But the latency KPI is not monitored all the time. It’s just assumed that it is within limits as long as other KPIs indicate a good link.

3.18.1.5 Analysis and Break-Down

3.18.1.5.1 Subjects to Monitoring

The solution shall support service level agreements with key performance indicators associated for all ordered services between user organisation, the system and resource provider(s), i.e., which data has to be monitored. This basically demands the common definition of SLIs by the governance authority.

3.18.1.5.2 Compensations

The solution shall have a method for compensating the user organisation for breaches of mutually agreed SLAs. Applying the assumptions that

this demands SLOs and SLIs to be in place. This is enabled by an API to request reports on SLOs from the providers.

3.18.1.5.3 Coordination

The solution shall coordinate the monitoring of the key performance indicators between user organisations, the system and resource providers. That is, a method has to be implemented by the solutioin that allows monitoring the SLI actual value (KPI) vs the concrete thresholds defined by the SLOs. To enable this, the system has to have a governance interface for defining SLIs and an API to request reports over SLOs from the providers.

It is important to understand that not all SLIs can actually be measured directly by a provider (see *OSS/BSS and KPI reflection*). Also, not all of them are available in time series, because they depend on other factors. For example, a provider can only monitor the signal/noise level, transmit and receive signal strength as well as the general connection to external networks and derive parameters such as logical bandwidth, latency and jitter from these - using static parameters (see *Key Performance Indicator - KPI* on derived KPI and *OSS/BSS and KPI reflection*). PSI cannot enforce the existence of a KPI in an existing OSS/BSS systems but rather have to use what they expose. At this point, we are therefore talking about coordination, not actual data acquisition.

The system should only guarantee that, if necessary, corresponding reports on the KPIs can be delivered by the providers. Such reports must meet the aforementioned definition of SLI and SLO.

However, the system does not have to manage the collection, aggregation, and reporting of this data itself.

3.18.1.5.4 Logging of SLA Breaches

The solution shall log any breach of service level agreement and take appropriate actions. That is, the system is not only responsible for the coordination of the monitoring, but also for an actual logging and reaction *within* the system if an SLO is violated. Thus, at least some information needs to be made available to the PSS. A mere orchestration of the responsibility is therefore not sufficient.

3.18.1.5.5 Link Recovery

The solution shall be able to offer a link recovery function to restore any interrupted communication link and shall ensure that broken links shall be restored within a maximum pre-agreed delay time per service grade. This recovery can be conducted via the PSS; the information on corruption of communication links however needs to be made available to the PSS and relates to the SLOs between provider and customer.

3.18.1.5.6 Alerting

The solution shall request that resource providers inform the system on the status of the link for all active services, i.e., **alerting** of SLO compliance (and/or deviation). Additionally, this imposes that the actual data shall be requested from the owner of the data (the provider). That is, the system shall not request and aggregate all possible data by itself but rely on the provider's system to monitor and alert.

3.18.1.5.7 Link Status

A common SLI "link status" with the following value domain should be imposed:

This "link status" is a derived value that depends on various parameters and can also only be formed by considering the SLOs and the period under consideration. Link status is therefore a compound KPI that can be formed for a given service and a mandatory SLI for every service. It expresses the quality of the link. The link status has to be **pushed** to the system by the provider. It's therefore an alerting rather than a monitoring.

3.18.1.5.8 Jamming

Jamming attacks could be conducted on single channels as well as multiple channels. The attack might be automatically mitigated on the provider's side by failover. Although anti-jamming measurements are to be implemented on the provider's side, a PSS should be enabled to handle information related to jamming attacks or anti-jamming mitigations, if provided by the providers. Thus, the API should enable the forwarding of such information. As outlined above, the link status monitoring enables to spot faulty links and thus the forwarding of such information is enabled. The actual enforcement of anti-jamming measurements has to be done by the governance of a PSS.

3.18.2 Decision Drivers

3.18.3 Considered Options

Prerequisite for any solution is the definition of proper SLIs by the governance and sharing them with the providers. Unlike a single provider quality management application, all parties involved in a system must be aligned to a set of SLIs. Providers could match the SLIs to their internal KPIs or even evaluate the creation of new performance indicators based on the provided SLIs. An API for this definition and sharing of SLIs needs to be created from scratch because it is not supported from TM Forum APIs.

Additional comments:

3.18.4 Decision Outcome

Note that this decision was accepted during the MS5 review meeting. However, the final decision was further postponed and work on this topic was put on hold as decided during the sprint review meeting for sprint 35 on 2023-09-20. There was an ongoing TM Forum Catalyst project by SES *Open (Gov)SatCom management* that will most likely have impact on the design of the interfaces and the API.

Choose Option B because large parts of the requirements - given the definitions in this MADR - are covered from TM Forum APIs standard.

Based on the analysis, it would not be meaningful and in some cases not possible to monitor all SLI-relevant actual data. The solution delegates monitoring and reporting to the OSS/BSS of the providers' system.

Still, it ensures compliance via the governance definition of minimal SLIs and the obligation to push the system status and provide reports on demand. In summary, this option incorporates:

3.18.5 Compliance

TMF628 Performance Management, *TMF649 Performance Threshold*, and *TMF642 Alarm Management* are (as they are) only *partially* compliant with the requirement set. All these TM Forum APIs lack a proper tracking to resources and services. Therefore, a patched version has to be created that allows a link to contracts, resources, and parties.

The data exchange shared entities with providers' systems is done with the help of the PSI events (see Asynchronous Workflow Approach).

The SLI definition API has to be created from scratch. The definition of proper SLIs by the governance is key to requirement compliance, though this cannot be enforced by the solution.

Additionally, the inquiry API *PSI001 Customer Inquiry* shall be updated to allow the customers to inquire about SLOs that flow down into the final SLA.

3.18.6 Implications for the Scope

Direct injection of data into the stream by the PSS is an additional security risk. However, a process to request or enforce a measurement of KPIs/KQIs on demand can be investigated further.

KPIs/KQIs can be compared with SLOs, however, they need to be measured and provided by the CSPs.

The systems needs to access defined SLOs and SLIs and enable mitigations. The APIs enable an exchange of corresponding data, i.e., definitions of SLI/SLO parameters and the definition of the consequences of their deviation.

For the coordination, the system needs access to the KPIs/KQIs. The discussed APIs enable an exchange of the corresponding data.

The link status, being monitored, is driving the PSS-internal process of recovering a broken communication link. Thus, if the service includes continuous monitoring, the APIs enable a process to monitor it.

The link status is an aggregated KPI. If the service includes a continuously monitoring, the discussed APIs enable a process to monitor it.

If the PSS is able to do a continuously monitoring of the link status, the mapping to the four states is also possible.

The interface allows setting the link status (nominal, reduced, downgraded, broken) via API to spot faulty links. Thus, the interface is to be understood as an enabler for security aspects, not enforcing such for a PSS connected to it.

3.19 Improve Definition of Product Types

3.19.1 Context and Problem Statement

While investigating the “Change Order” workflow, it became apparent that the current focus of the model on “features” may not be sustainable in the used form. They are only available in the service and resource layers, but not for products and offerings, making it impossible to change them directly. On the contrary, the owner of TMF620 “Product Catalog” explicitly stated that features are not meant to be used in products, because add-on offerings are better to model those cases.

> “1. Feature entity on Product CatalogTMF620 -
> Hi All,
> Is there a plan to add FeatureSpecification and related entities to Product Catalog? Thanks in advance.”,
> Serkan Kaya, 25th April 2022 [9].
> “2. RE: Feature entity on Product CatalogTMF620 -
> There are currently no such plans. Do you think that there is a business jsutification for this at a product level? Seemingly a feature at product level would be modeled as an add-on product offering and spec, since it probably comes with pricing implications.”,
> Jonathan Goldberg, 26th April 2022.[10]

This also implies that our current interpretation of features is wrong, which is seconded by another community thread about their intended usage as “optional configuration of a service/resource”, whereas we assumed it is some kind of capability.

> “2. RE: Characteristics vs Features - SID & Open API Alignment -
> Hi Srinivas
> Features were introduced into the Open API **Resource** models (catalog and inventory), I think originally by @Vance Shipley, to give a more intent-focused interface as against individual characteristics.
> Later, they were added also the Open API **Service** models. It’s an implementation choice whether to use Characteristics or Features for any particular functionality.
> Hope it helps.”,
> Jonathan Goldberg, 31st July 2022.[11]

Overall, it came apparent that the PSS needs to have at least basic knowledge about the product, service and resource types it deals with. This will allow it to define a logic on how product characteristics are translated into services and resources. We have to examine how to allow a stricter definition of those types without sacrificing the flexibility of the interface to adapt to different implementations.

3.19.2 Decision Drivers

3.19.3 Decision Outcome

First, we step back from using features as “capabilities” as done before. They may still be used in the intended way of “optional configuration” if needed, but this is out of scope for this decision.

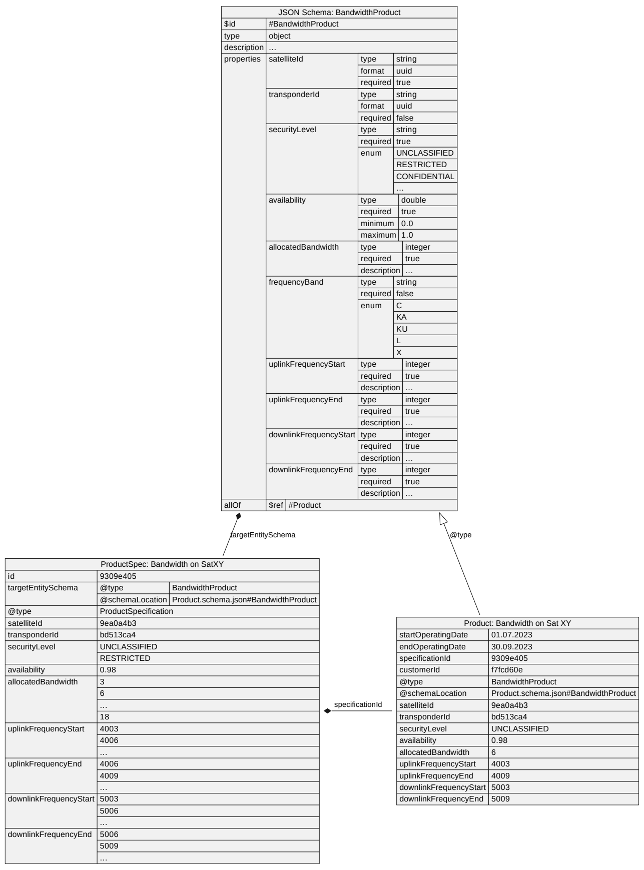
This of course requires a definition of an alternative concept. TM Forum provides such a concept, which is usually complementary to our current characteristic-based approach: JSON Schemas. Refer to the documentation of the catalog APIs for more information on the intended use of it, as the following description deliberately deviates from this in some aspects to combine both.

With this new approach, the PSS manufacturer (a software team or company responsible for the implementation, not necessarily operations or governance) is the first instance to define which types of products, services and resources are supported by the PSS. They do so by defining a JSON Schema for each supported entity type, which contains a fixed list of supported characteristics and their data types. The produced file shall be provided to providers via HTTP as part of the PSI implementation, but can also be sent via e-mail. The provider then uses this definition

[9] TM Forum - Community Thread: Feature entity on Product CatalogTMF620 (last viewed on 7th March 2024)
[10] TM Forum - Community Thread: Feature entity on Product CatalogTMF620 (last viewed on 7th March 2024)
[11] TM Forum - Community Thread: Characteristics vs Features - SID & OpenAPI Alignment (last viewed on 7th March 2024)

to build resource, service and product specifications. This process is simplified by the machine-readable format. Additionally, the governance may provide more concrete templates for services and resources via `TOD-04`.

An example of how the JSON schema can be applied to a bandwidth product is shown in 3.39.



JSON Schema: BandwidthProduct			
$id	#BandwidthProduct		
type	object		
description	…		
properties	satelliteId	type	string
		format	uuid
		required	true
	transponderId	type	string
		format	uuid
		required	false
	securityLevel	type	string
		required	true
		enum	UNCLASSIFIED
			RESTRICTED
			CONFIDENTIAL
			…
	availability	type	double
		required	true
		minimum	0.0
		maximum	1.0
	allocatedBandwidth	type	integer
		required	true
		description	…
	frequencyBand	type	string
		required	false
		enum	C
			KA
			KU
			L
			X
	uplinkFrequencyStart	type	integer
		required	true
		description	…
	uplinkFrequencyEnd	type	integer
		required	true
		description	…
	downlinkFrequencyStart	type	integer
		required	true
		description	…
	downlinkFrequencyEnd	type	integer
		required	true
		description	…
allOf	$ref	#Product	

ProductSpec: Bandwidth on SatXY		
id	9309e405	
targetEntitySchema	@type	BandwidthProduct
	@schemaLocation	Product.schema.json#BandwidthProduct
@type	ProductSpecification	
satelliteId	9ea0a4b3	
transponderId	bd513ca4	
securityLevel	UNCLASSIFIED	
	RESTRICTED	
availability	0.98	
allocatedBandwidth	3	
	6	
	…	
	18	
uplinkFrequencyStart	4003	
	4006	
	…	
uplinkFrequencyEnd	4006	
	4009	
	…	
downlinkFrequencyStart	5003	
	5006	
	…	
downlinkFrequencyEnd	5006	
	5009	
	…	

Product: Bandwidth on Sat XY	
startOperatingDate	01.07.2023
endOperatingDate	30.09.2023
specificationId	9309e405
customerId	f7fcd60e
@type	BandwidthProduct
@schemaLocation	Product.schema.json#BandwidthProduct
satelliteId	9ea0a4b3
transponderId	bd513ca4
securityLevel	UNCLASSIFIED
availability	0.98
allocatedBandwidth	6
uplinkFrequencyStart	4003
uplinkFrequencyEnd	4009
downlinkFrequencyStart	5003
downlinkFrequencyEnd	5009

Figure 3.39: JSON Schema applied to a product

In the specification, we still use the characteristic array as we did before, but use the defined attribute names as `id`s. This allows the provider to send additional information if desired, which may still be accepted by the PSS for later use (e.g., to display it in a simple "product detail" list). The schema name and location are transmitted via the `targetEntitySchema` attribute, which can also be used to query specifications of a specific type.

The inventory instance will store the actual values in custom fields instead of characteristics, making access to them easier and compliant to the JSON Schema. In this entity, the schema name is defined in `@type` (while `@baseType` is still `Product`, `Service` or `Resource`), the location in `@schemaLocation`.

3.19.4 Compliance

Since this is a far-reaching change in concept, multiple things will be adjusted in separate backlog tasks:

3.19.5 Security Considerations

The schema definitions expose implementation details of the PSS and must be protected from unauthorized access. However, this is already covered by the generic "data visibility" chapter of the ICD.

3.19.6 Implications for the Scope

The schema contains all available types as well as all static and dynamic fields for a type. It can be given to users and providers for them to fill it. The PSS can compare the specification against the specified schema.

3.20 Bandwidth (De-)Fragmentation

3.20.1 Context and Problem Statement

When bandwidth is sold in chunks, over time it will probably end up fragmented. An example of a fragmented frequency band is shown in 3.40.

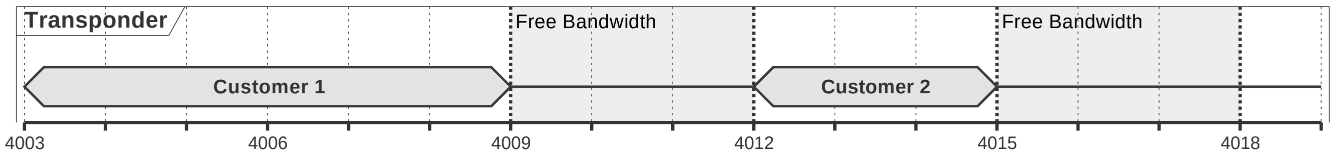


Figure 3.40: Fragmented Bandwidth

In theory, there are 6 MHz left for a third customer to use, but because of fragmentation it can not be fully utilized. We have to analyse if a defragmentation as shown in 3.41 is already supported by PSI or if further extensions have to be made.

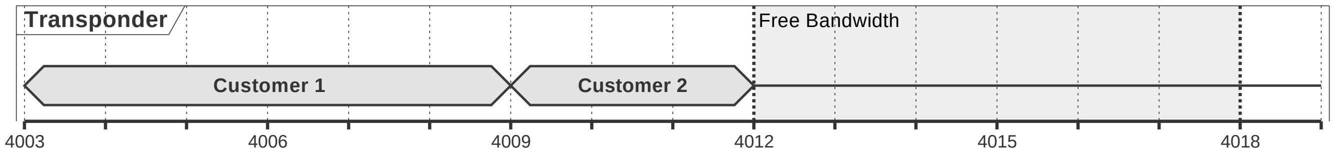


Figure 3.41: Defragmented Bandwidth

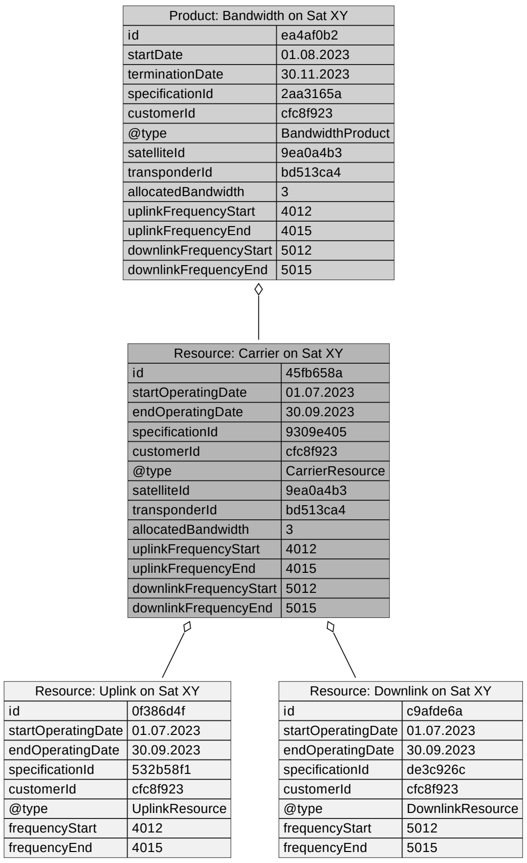
3.20.2 Decision Outcome

The defragmentation of bandwidth lies in the responsibility of the resource owner, which can be either the PSS (for committed resources) or the provider. It could be done automatically (for managed services), but may still require manual intervention to evaluate the impact on active connections and the resulting legal and financial risks. While the latter is out-of-scope for this project, PSI does already support the necessary communication between the systems to synchronize the configuration changes.

The following subsections describe the information flow and subsequent model changes depending on the initiator.

3.20.2.1 Initiation by the Provider

In case of uncommitted resources, the PSS inventory contains only the two booked chunks of bandwidth as shown in 3.42.



Product: Bandwidth on Sat XY	
id	ea4af0b2
startDate	01.08.2023
terminationDate	30.11.2023
specificationId	2aa3165a
customerId	cfc8f923
@type	BandwidthProduct
satelliteId	9ea0a4b3
transponderId	bd513ca4
allocatedBandwidth	3
uplinkFrequencyStart	4012
uplinkFrequencyEnd	4015
downlinkFrequencyStart	5012
downlinkFrequencyEnd	5015

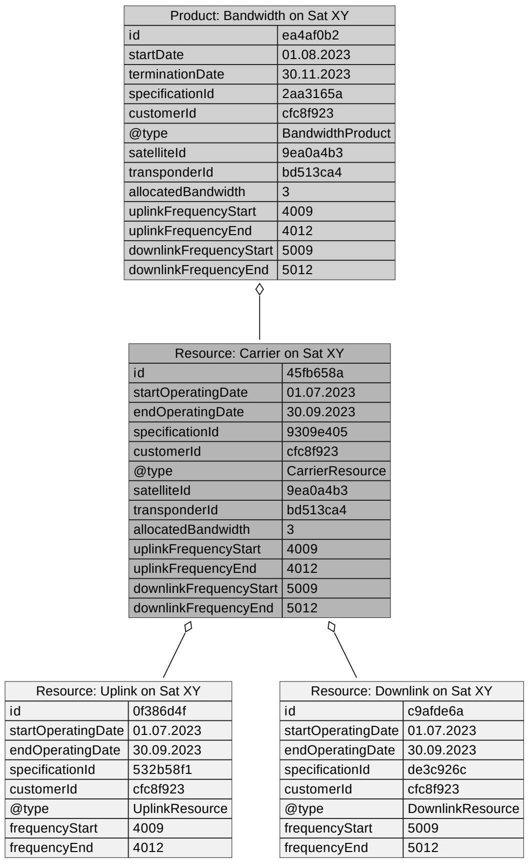
Resource: Carrier on Sat XY	
id	45fb658a
startOperatingDate	01.07.2023
endOperatingDate	30.09.2023
specificationId	9309e405
customerId	cfc8f923
@type	CarrierResource
satelliteId	9ea0a4b3
transponderId	bd513ca4
allocatedBandwidth	3
uplinkFrequencyStart	4012
uplinkFrequencyEnd	4015
downlinkFrequencyStart	5012
downlinkFrequencyEnd	5015

Resource: Uplink on Sat XY	
id	0f386d4f
startOperatingDate	01.07.2023
endOperatingDate	30.09.2023
specificationId	532b58f1
customerId	cfc8f923
@type	UplinkResource
frequencyStart	4012
frequencyEnd	4015

Resource: Downlink on Sat XY	
id	c9afde6a
startOperatingDate	01.07.2023
endOperatingDate	30.09.2023
specificationId	de3c926c
customerId	cfc8f923
@type	DownlinkResource
frequencyStart	5012
frequencyEnd	5015

Figure 3.42: PSS Inventory before defragmentation by the provider

After checking the technical and legal feasibility, the provider can decide to move the bandwidth “Customer B” to 4009-4012 MHz (downlink alike) as shown in 3.43. The BSS/OSS then invokes the operation TOD-05-01-02 (`PATCH /resourceInventory/v1/resource/{id}`) on the PSS, sending the new feature characteristics in the request body.



Product: Bandwidth on Sat XY	
id	ea4af0b2
startDate	01.08.2023
terminationDate	30.11.2023
specificationId	2aa3165a
customerId	cfc8f923
@type	BandwidthProduct
satelliteId	9ea0a4b3
transponderId	bd513ca4
allocatedBandwidth	3
uplinkFrequencyStart	4009
uplinkFrequencyEnd	4012
downlinkFrequencyStart	5009
downlinkFrequencyEnd	5012

Resource: Carrier on Sat XY	
id	45fb658a
startOperatingDate	01.07.2023
endOperatingDate	30.09.2023
specificationId	9309e405
customerId	cfc8f923
@type	CarrierResource
satelliteId	9ea0a4b3
transponderId	bd513ca4
allocatedBandwidth	3
uplinkFrequencyStart	4009
uplinkFrequencyEnd	4012
downlinkFrequencyStart	5009
downlinkFrequencyEnd	5012

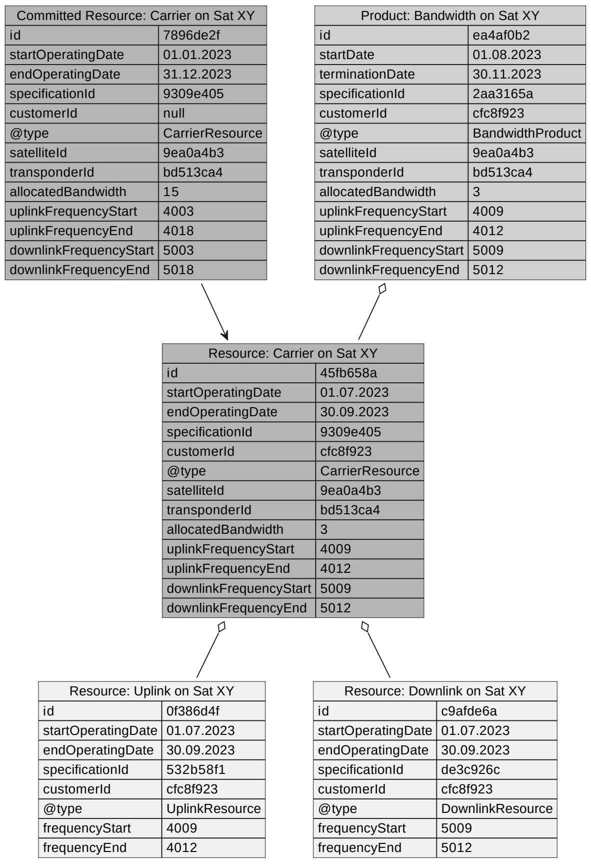
Resource: Uplink on Sat XY	
id	0f386d4f
startOperatingDate	01.07.2023
endOperatingDate	30.09.2023
specificationId	532b58f1
customerId	cfc8f923
@type	UplinkResource
frequencyStart	4009
frequencyEnd	4012

Resource: Downlink on Sat XY	
id	c9afde6a
startOperatingDate	01.07.2023
endOperatingDate	30.09.2023
specificationId	de3c926c
customerId	cfc8f923
@type	DownlinkResource
frequencyStart	5009
frequencyEnd	5012

Figure 3.43: PSS Inventory after defragmentation by the provider

3.20.2.2 Initiation by the PSS

If a resource is committed, it has a dedicated entry in the inventory resembling the full frequency range. This allows the PSS to perform defragmentation on its own in the given boundaries. An example for an initial state *before* defragmentation is shown in 3.44.



Committed Resource: Carrier on Sat XY	
id	7896de2f
startOperatingDate	01.01.2023
endOperatingDate	31.12.2023
specificationId	9309e405
customerId	null
@type	CarrierResource
satelliteId	9ea0a4b3
transponderId	bd513ca4
allocatedBandwidth	15
uplinkFrequencyStart	4003
uplinkFrequencyEnd	4018
downlinkFrequencyStart	5003
downlinkFrequencyEnd	5018

Product: Bandwidth on Sat XY	
id	ea4af0b2
startDate	01.08.2023
terminationDate	30.11.2023
specificationId	2aa3165a
customerId	cfc8f923
@type	BandwidthProduct
satelliteId	9ea0a4b3
transponderId	bd513ca4
allocatedBandwidth	3
uplinkFrequencyStart	4009
uplinkFrequencyEnd	4012
downlinkFrequencyStart	5009
downlinkFrequencyEnd	5012

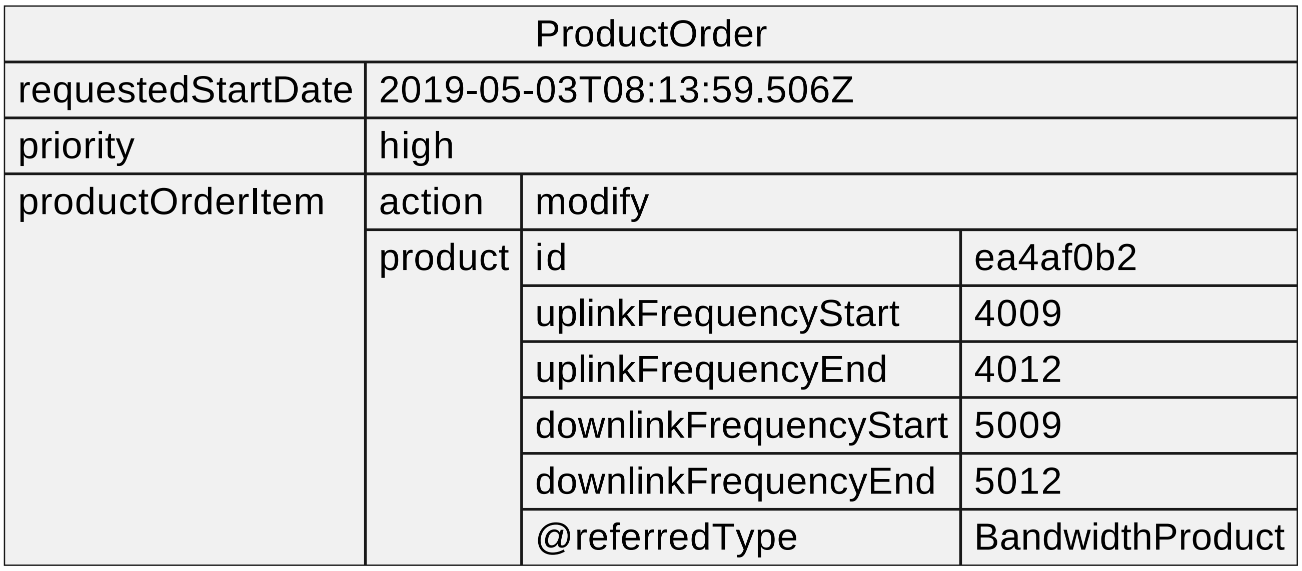
Resource: Carrier on Sat XY	
id	45fb658a
startOperatingDate	01.07.2023
endOperatingDate	30.09.2023
specificationId	9309e405
customerId	cfc8f923
@type	CarrierResource
satelliteId	9ea0a4b3
transponderId	bd513ca4
allocatedBandwidth	3
uplinkFrequencyStart	4009
uplinkFrequencyEnd	4012
downlinkFrequencyStart	5009
downlinkFrequencyEnd	5012

Resource: Uplink on Sat XY	
id	0f386d4f
startOperatingDate	01.07.2023
endOperatingDate	30.09.2023
specificationId	532b58f1
customerId	cfc8f923
@type	UplinkResource
frequencyStart	4009
frequencyEnd	4012

Resource: Downlink on Sat XY	
id	c9afde6a
startOperatingDate	01.07.2023
endOperatingDate	30.09.2023
specificationId	de3c926c
customerId	cfc8f923
@type	DownlinkResource
frequencyStart	5009
frequencyEnd	5012

Figure 3.44: Inventory before defragmentation by the PSS

The PSS governance may perform the same manual checks as the provider above, but may also employ automatic defragmentation algorithms. Since it has no direct control over the service or resource, it requests the changes by sending product orders with action "modify" via operation TOD-03-02-01, see 3.45. Note that by using the `requestedStartDate` property may be used to plan the migration path for multiple chunks in a defined order.



ProductOrder			
requestedStartDate	2019-05-03T08:13:59.506Z		
priority	high		
productOrderItem	action	modify	
	product	id	ea4af0b2
		uplinkFrequencyStart	4009
		uplinkFrequencyEnd	4012
		downlinkFrequencyStart	5009
		downlinkFrequencyEnd	5012
		@referredType	BandwidthProduct

Figure 3.45: Product Order with modify action

The provider then handles it like any other change request (e.g., from the customers themselves) and either carries it out or rejects giving the reasoning. On acceptance, the updates of the inventory are then performed exactly like shown in the previous example.

3.20.3 Compliance

Configuration changes will be included in the case study, hinting at the possibility that defragmentation can be a reason to trigger them.

3.20.4 Implications for the Scope

The described workflow can be used by the customers themselves (through the PSS) to request changes to any other characteristic.

3.21 Event API: Authentication

3.21.1 Context and Problem Statement

With the implementation of the Event Management API in the prototype, the question of authentication gained new weight: While all other APIs require a customized implementation that can do the authentication in a bespoke manner, event callbacks are by nature used rather generic. We have to decide how to handle authentication here, potentially improving the API or documenting another chosen approach.

3.21.2 Decision Drivers

3.21.3 Considered Options

3.21.4 Decision Outcome

Chosen option: “Configuration as part of the implementation”, because it is the most flexible and secure approach. While it is not self-describing, we consider a separate configuration a security improvement.

3.21.5 Compliance

3.21.6 Pros and Cons of the Options

3.21.6.1 API-Tokens as URL-Parameters

API-Tokens are randomized strings that are generated by (and therefore bound to) individuals or organizations, i.e., providers. They may have a limited lifetime (usually days or months), but are often manually invalidated. Since they are simply appended to the callback URL, this approach would require no changes in the API.

3.21.6.2 Allow Definition of additional Headers on Callback Registration

HTTP defines an “Authorization” header, which could be used in different ways. In “Basic” mode it transports classic credentials, but it can also be used for API-Tokens. We could extend the Event Management API to allow the definition of such headers together with the callback URL.

3.21.6.3 Allow explicit Definition of Authentication on Callback Registration

To allow OAuth2 as an authentication method, we would have to add more attributes to the registration endpoint than just fixed headers. We could use the definition in the Open API standard as inspiration. In addition, we’d have to store the actual credentials with it.

3.21.6.4 Configuration as Part of the Implementation

As of writing this decision record, authentication is defined to be implementation-specific. This results from the different security requirements of every system that can not be easily unified.

Keeping this concept, we could suggest an implementation specific, host-based configuration of authentication. This would impose a list of allowed hosts for callbacks, which would improve the security in itself. Additionally, the list would be augmented with a specific authentication method for each one, including OAuth2 or any other method discussed in this decision. The implementation can offer this as a self-service to the providers or limit it to the system operators (e.g. for security reasons or complexity concerns). The Event API would be kept as-is, because the workflow is expected to be done beforehand.

3.21.7 Implications for the Scope

The interface is developed with state-of-the-art designs and is based on a common industry standard. The interface is to be understood as an enabler for security aspects, not enforcing such for a PSS connected to it.

3.22 InquiredProduct-InquiryResult Relation

3.22.1 Context and Problem Statement

The CustomerInquiry API allows requesting multiple products at once, e.g. Internet Access at different locations. In these cases, it may return a result with multiple ProductOfferings, e.g. one for each location. However, the customer cannot determine which offering belongs to which location.

Figure 3.46 describes this situation based on the ProductOffering array. Parallel to ProductOffering, the Product-Specification and Product arrays are also available in InquiryResult. For the sake of simplicity, these additional entities have been omitted from the diagrams.

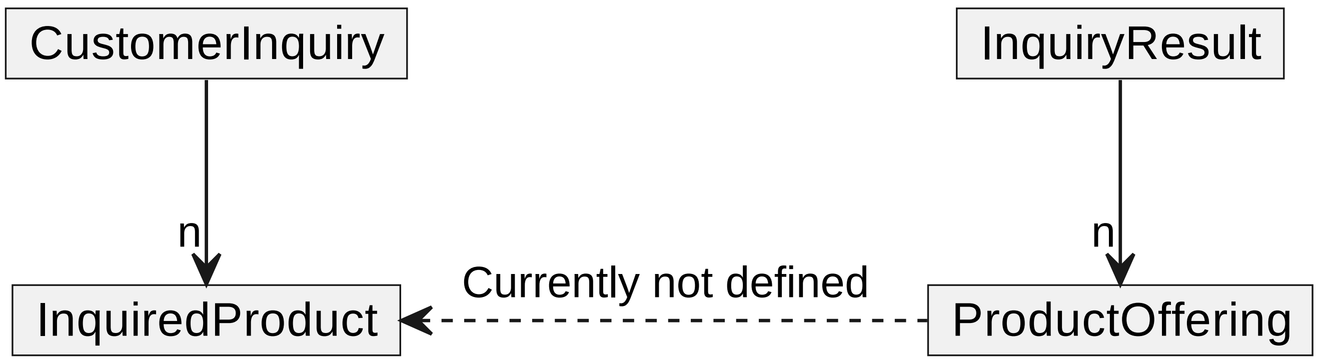


Figure 3.46: Current Situation: no Relationship between Inquiry and Result

Furthermore, it is necessary to exchange additional information about the relationship, such as a rating of the match, the offered service period, etc.

3.22.1.1 Scenarios

Depending on the number of InquiredProducts requested and the number of ProductOfferings received, different situations arise. Some of them do not need any further explanation, but if the cardinality of one side differs from “1”, explicitly defined relationships are required.

3.22.1.1.1 Scenarios with Implicitly Known Relations

Following scenarios have either a single InquiredProduct or a single ProductOffering, which directly defines the relationships.

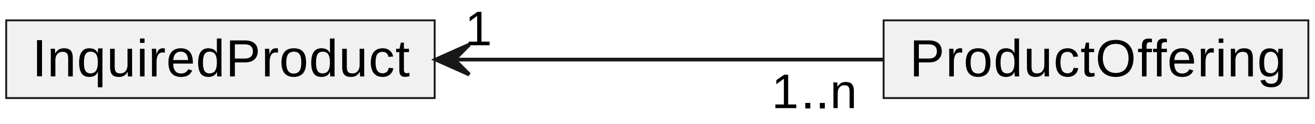


Figure 3.47: One ProductOffering to 1..n InquiredProducts

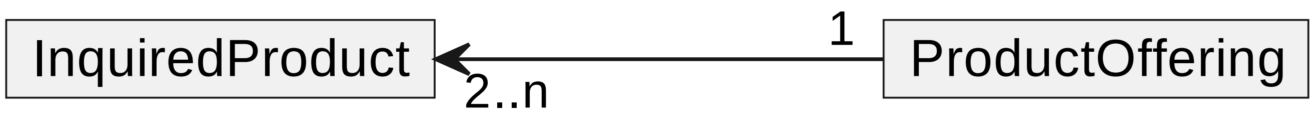


Figure 3.48: One ProductOffering to n InquiredProducts.

3.22.1.1.2 Scenarios requiring Explicit Definition of Relations

The first scenario that requires a relationship is about *2..n* InquiredProducts that results in only one ProductOffering, but with a quantity > 1. The difference to the example above is that this single offering is used not just once, but several times for independent order items - to take the differences of the InquiredProducts into account.

For example, a customer could request the same Internet access at different locations. An appropriate outcome of this request could be a single offering with a service area that covers the requested locations. However, multiple instances of the offered product are needed (e.g. due to the different location, date rate, etc.). This results in the same offering being ordered twice or more.

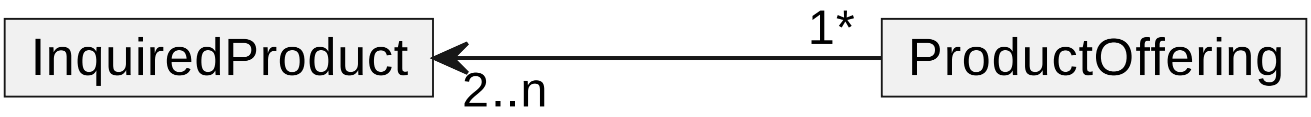


Figure 3.49: One ProductOffering but x Instances needed.

And finally several InquiredProducts that need to be mapped to several offerings.

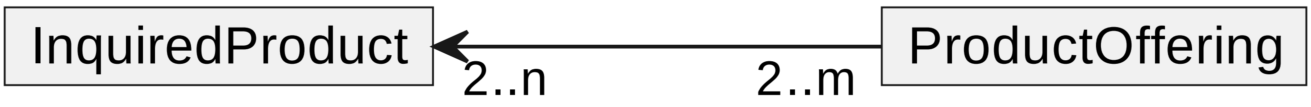


Figure 3.50: m ProductOfferings assigned to n InquiredProducts.

3.22.1.2 Relationship Enrichment

It can also be useful to use these new structures to facilitate the exchange of additional information about the relationship. This could include:

It should also be noted that the calculation of the total price at the InquiryResult level could get complex. For example, prices can refer to one or more time periods, possibly with different discount. To avoid any problems, the total prices should be calculated and provided by the provider, too.

3.22.2 Decision Drivers

3.22.3 Considered Options

3.22.3.1 Extending PSID620_ProductCatalog

The ProductOffering, which is part of PSID620 ProductCatalog, is already used as an array in the InquiryResult. An array of entities could be added in ProductOffering that includes:

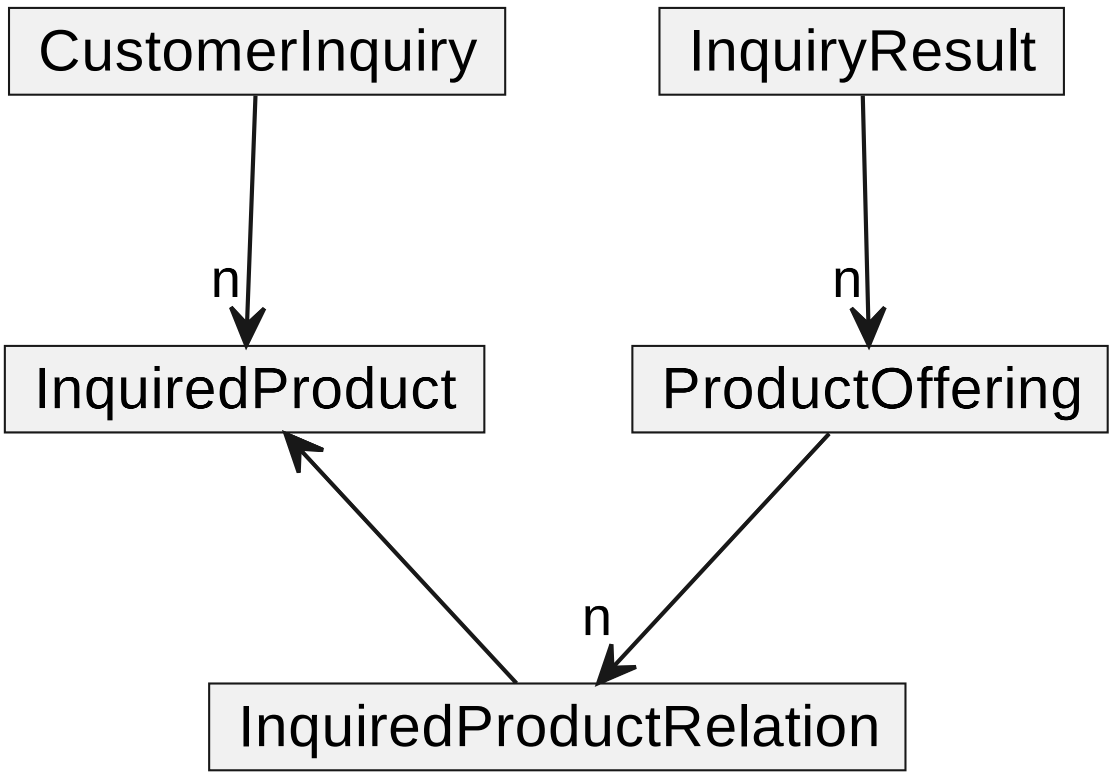


Figure 3.51: ProductOffering with Reference.

An example of the enhanced ProductOffering with enrichment fields could look like this:

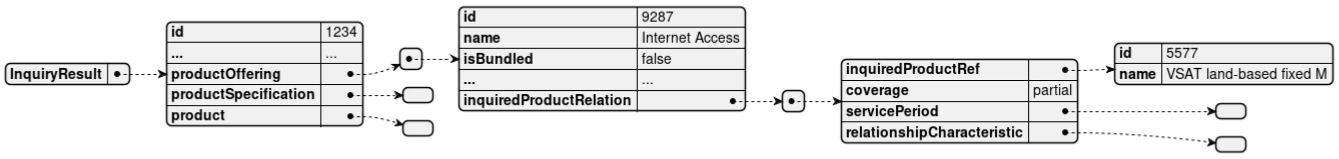


Figure 3.52: InquiryResult with ProductOffering and direct Reference.

The same enhancement would be necessary for ProductSpecification (PSID620-ProductCatalog) and Product (PSID-637-ProductInventory).

3.22.3.2 Extending PSID001_CustomerInquiry by *Wrapper* Objects

A single InquiryResult object consists in general of an ID, a ProductOffering array and/or a ProductSpecification array or a Product array. A wrapper could be added for each of the ProductOffering, ProductSpecification and Product. Each of the wrapper is then referencing the InquiredProduct. Please note that ProductSpecification and Product entities have been omitted from the diagram for the sake of simplicity.

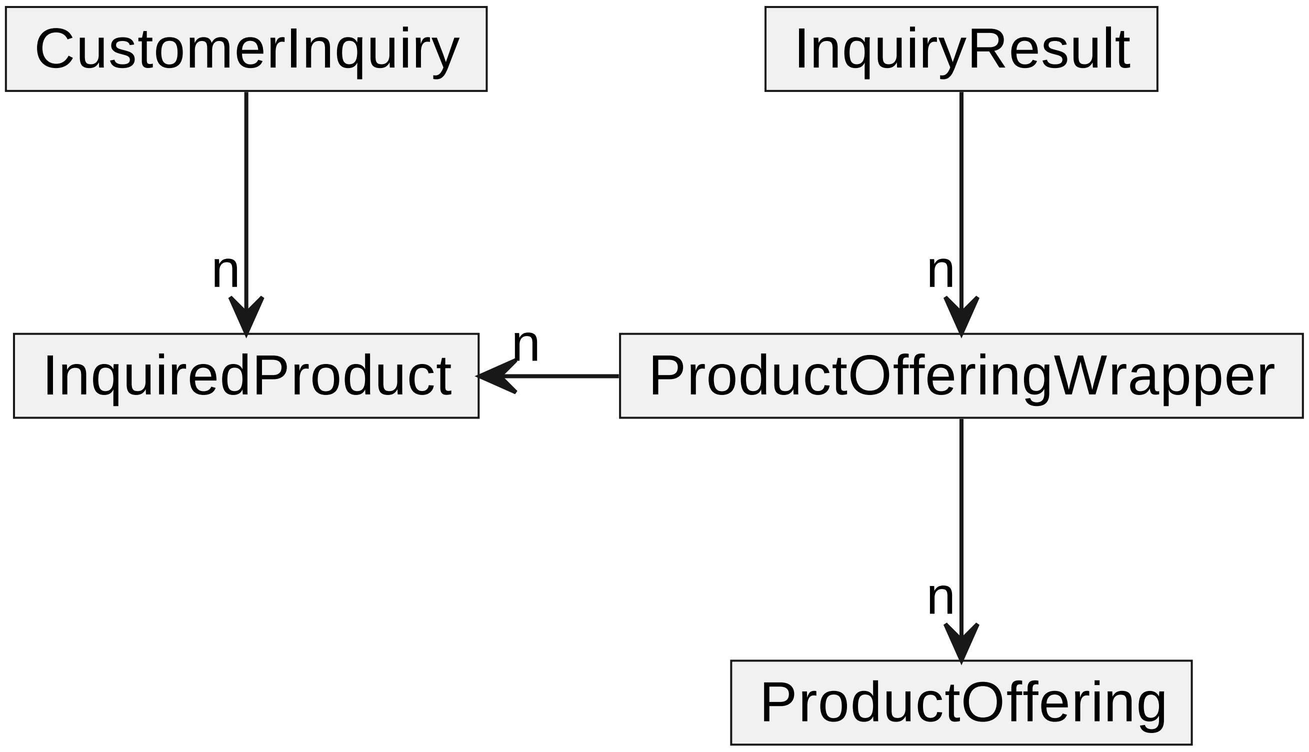


Figure 3.53: ProductOfferingWrapper.

An example of the ProductOffering wrapper with enrichment fields could look like this:

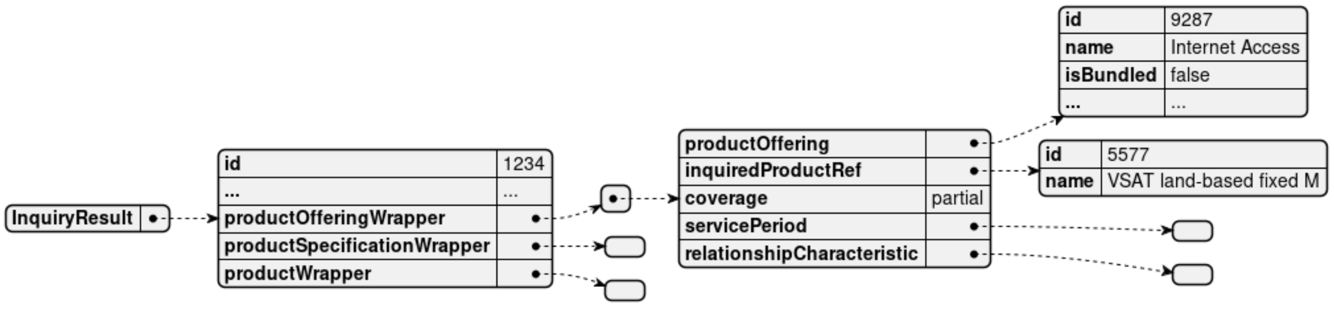


Figure 3.54: InquiryResult with Wrapper Objects.

Or a new InquiredProductResult entity could collect the ProductOffering, ProductSpecification and Product together with the additional details:

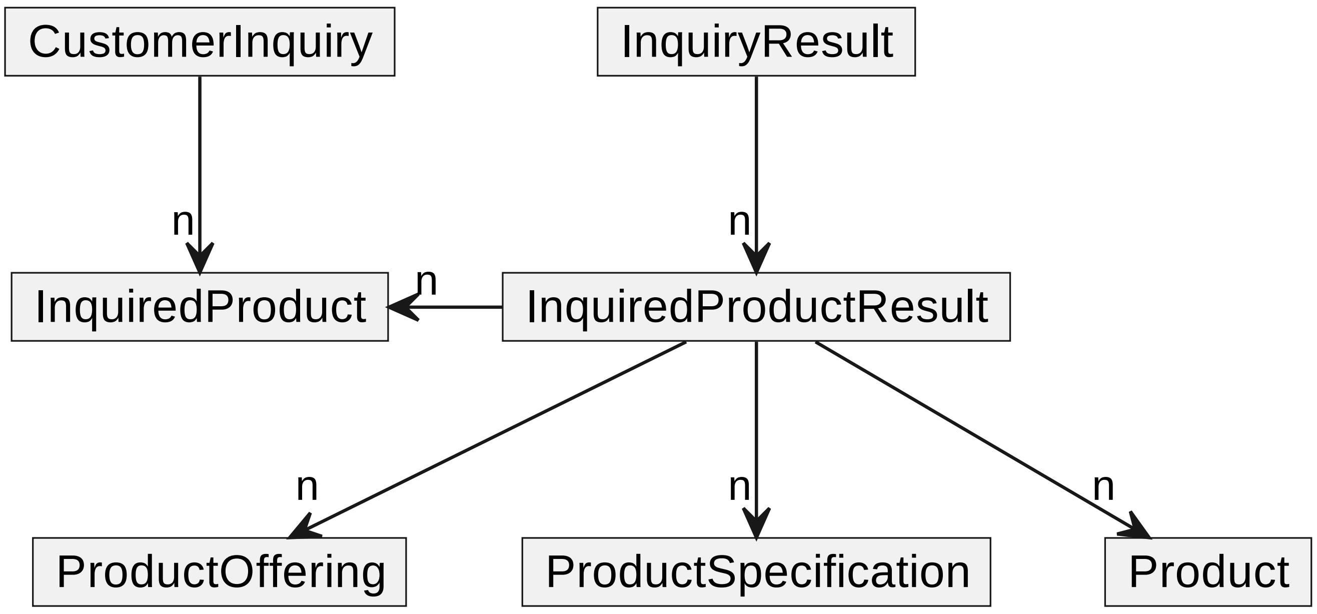


Figure 3.55: InquiredProductResult.

The structure of the InquiredProductResult could look like this:

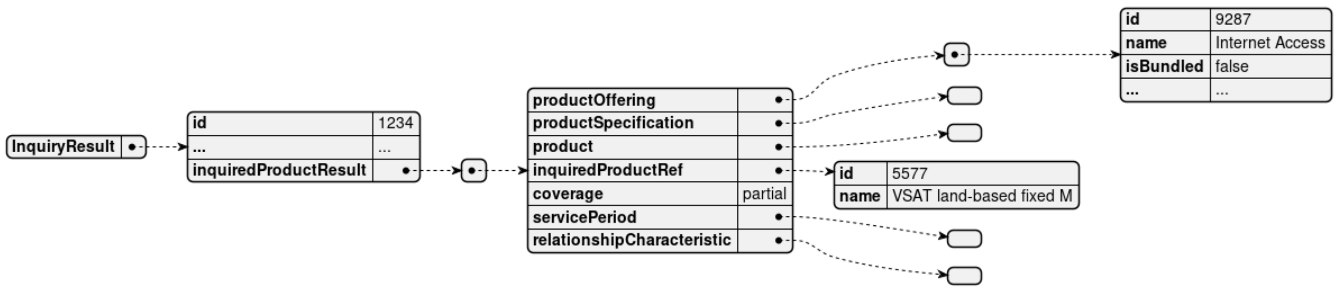


Figure 3.56: InquiryResult with a single Wrapper Object.

3.22.3.3 Extending PSID001_CustomerInquiry by an `InquiredProductRelationship`

Another option would be to keep the existing structures and add a new entity `InquiredProductRelationship` in the InquiryResult. This new entity refers to local ProductOfferings and the related InquiredProduct entries.

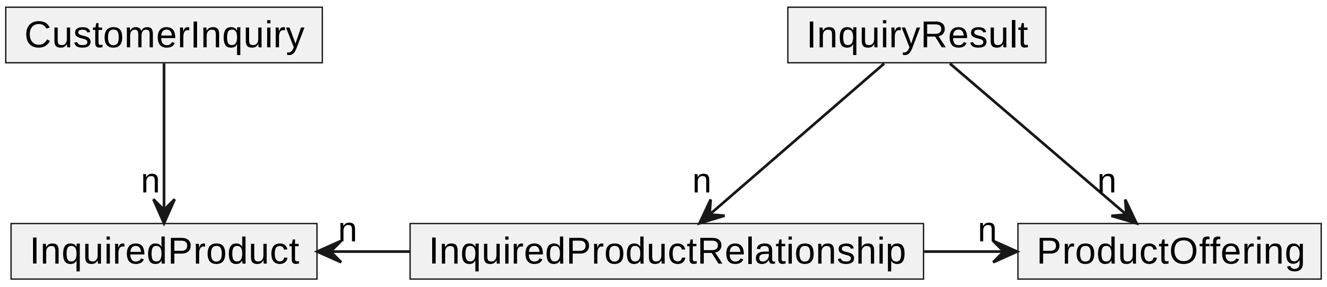


Figure 3.57: InquiredProductRelationship.

With the above-mentioned fields, the Structure of the `InquiredProductRelationship` could look like this:

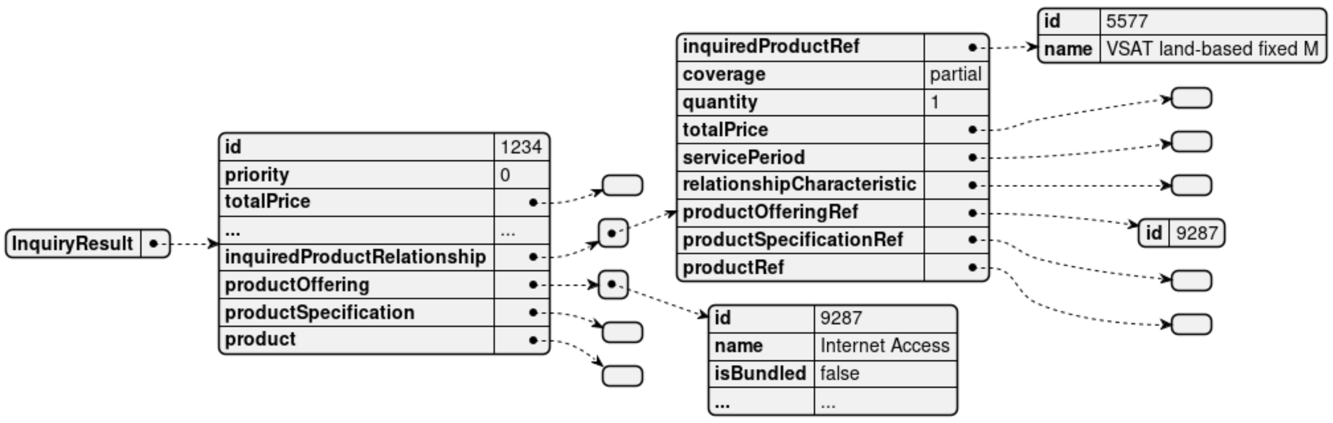


Figure 3.58: InquiredProductRelationship JSON Example.

3.22.4 Decision Outcome

It was decided to go for the *Extending PSID001 CustomerInquiry by an* `InquiredProductRelationship` proposal, as it reduces data duplicates and avoids changes to data structures originating from TM Forum.

3.22.5 Compliance

The interface definition of the CustomerInquiry as part of the Product Catalog Management API will be improved. The ICD documentation will reflect the improved schema.

3.22.6 Pros and Cons of the Options

3.22.6.1 Extending PSID620_ProductCatalog

To be more precise, this approach requires extensions for three different entities:

All three entities will be extended by a reference to the InquiredProduct plus all additional details like `servicePeriod`, `totalPrice` and more.

3.22.6.2 Extending PSID001_CustomerInquiry by *Wrapper* Objects

A wrapper for each of ProductOffering, ProductSpecification and Product could be added. Each of the wrappers is then referencing the InquiredProduct.

A variation of the wrapper approach is the InquiredProductResult solution, which joins the three different wrappers into one. This raises additional issues:

3.22.6.3 Extending PSID001_CustomerInquiry by an `InquiredProductRelationship`

This solution keeps the existing structures by adding one additional entity in InquiryResult that manages the enrichment data based on references to InquiredProduct and to ProductOffering/Product/ProductSpecification.

3.23 Mission Component Scope

3.23.1 Context and Problem Statement

We want to create an ODA component for the newly developed Mission API to enable faster acceptance. Additionally, a prototypic Web-UI should demonstrate its capabilities and the possible integration into a PSS. This decision record defines the functional scope to be achieved and outlines the envisioned technical architecture.

3.23.2 Decision Drivers

3.23.3 Decision Outcome

The scope of the backend is mostly defined by the definition of an ODA component by TM Forum, which implies the use of Kubernetes and therefore allows for high availability setups. The “core function” of it must be a microservice implementing the Mission API, i.e. providing the given REST endpoints to create, list, modify and delete missions. Except for input validation, there will be no business logic implemented. Additionally, the component will contain the necessary storage backend and outgoing telemetry.

As for the security functions, it was decided to implement JWT-based access control directly into the microservice. The identity provider should be configurable, because it is out of scope from a specific component. For multi-tenancy, the JWT-claim (attribute) to be used to identify the owner and other parties with access permissions of a mission should be configurable as well. While it is possible to implement the corresponding checks in an additional microservice, it makes pagination much harder and the added complexity leads to an operational overhead that seemed unreasonable and is a potential risk to data confidentiality. Optionally, it should be possible to deactivate internal JWT validation to have a similar behaviour as the reference implementations. This could be useful, if the operator uses other means to protect access to the backend.

While the microservices above form “the ODA component”, the work package also includes an exemplary/generic UI for it. This could be integrated as an optional microservice as part of the component. However, the reference implementations are API-first and do not contain UIs. It is understood that this stems from the much higher customization of such a UI, which makes a production use of it unlikely. That is why it will be delivered in a separate package that can be combined with the backend or used as an inspiration for a custom implementation. Note that the UI will probably not be able to showcase the full potential as described in the GID.

Figure 3.59 shows the logical overview of the envisioned architecture. As required by the ODA Canvas, it is based on Kubernetes with Helm as a package manager. The K8s Ingress accepts the incoming traffic from Users and maps one sub-path to the Mission Management frontend, which is implemented by an OpenResty web server. OpenResty is derived from NGINX and serves the statically compiled files for the UI itself, but also acts as a reverse-proxy for the backend API. This allows it to store the JWT in a local session, which increases security by not sending it to the potentially unsafe browser. When the frontend needs to call any backend, including the Mission Management backend or for example the Inquiry API, it will do so through the reverse-proxy to ensure authentication. The operator can choose to expose the backend directly via a separate Ingress endpoint to allow machine-to-machine communication.

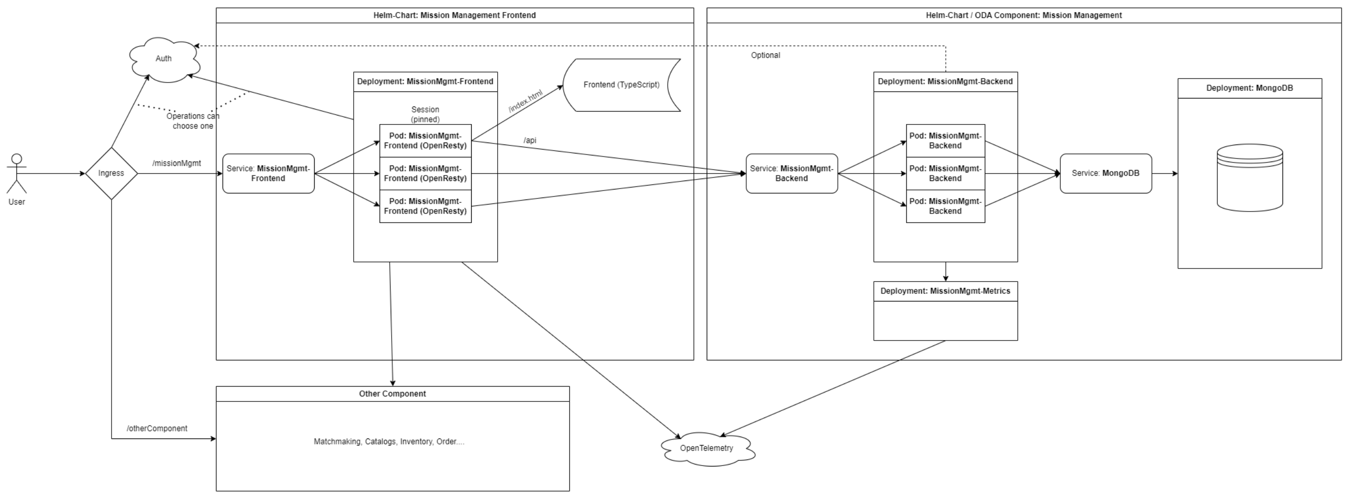

Helm-Chart: Mission Management Frontend
Helm-Chart / ODA Component: Mission Management
Auth
Optional
Operations can choose one
User
Ingress
/missionMgmt
Service: MissionMgmt-Frontend
Deployment: MissionMgmt-Frontend
Session (pinned)
Pod: MissionMgmt-Frontend (OpenResty)
Pod: MissionMgmt-Frontend (OpenResty)
Pod: MissionMgmt-Frontend (OpenResty)
/index.html
Frontend (TypeScript)
/api
Service: MissionMgmt-Backend
Deployment: MissionMgmt-Backend
Pod: MissionMgmt-Backend
Pod: MissionMgmt-Backend
Pod: MissionMgmt-Backend
Service: MongoDB
Deployment: MongoDB
Deployment: MissionMgmt-Metrics
Other Component
Matchmaking, Catalogs, Inventory, Order....
/otherComponent
OpenTelemetry

Figure 3.59: Mission Component Architecture View

3.23.4 Compliance

Tasks to create the described items will be added to the backlog and planned for the next sprints. The TO will govern the implementation of the given architecture.

3.24 ODA Implementation - Language and Framework for the Backend

3.24.1 Context and Problem Statement

The implementation of a TM Forum ODA component is planned. The programming language with the appropriate frameworks must be selected for the backend.

3.24.2 Decision Drivers

3.24.3 Considered Options

3.24.4 Decision Outcome

Chosen option: "Go". All options would do a good job in terms of REST API, JWT and data persistence. But solutions with Python, Node.js and Java would result in ODA containers that are too large, hindering fast downloading and testing for interested parties. That leaves Rust and Go. The argument of the simpler language remains in favour of Go.

3.24.5 Compliance

All necessary settings for PSI ODA creation are part of the repository. The selected programming language is reflected in the settings and preset plugins for Visual Studio Code

3.24.6 Pros and Cons of the Options

3.24.6.1 Java

Java was started by Sun Microsystems in 1995 as a multiplatform, object-oriented programming language. Despite its age, it is still one of the most popular and powerful programming languages in the world.

Pros:

Cons:

3.24.6.2 Node.js

Node.js is a server-side runtime environment for JavaScript, known for its non-blocking, event-driven architecture, making it ideal for scalable network applications. It's built on Chrome's V8 engine and uses npm for managing libraries and modules.

Pros:

Cons:

3.24.6.3 Python

Python is a popular interpreter language for web development due to its simplicity and readability. Its extensive libraries and tools support various web technologies, making it a versatile choice for both backend and full-stack development.

Pros:

Cons:

3.24.6.4 Rust

Rust was launched in 2006 as a private project and is now being further developed as open source by the Rust Foundation. It emphasizes performance and type safety, without a runtime or a garbage collector. In December 2022, it became the first language other than C and assembly to be supported in the development of the Linux kernel.

Pros:

Cons:

3.24.6.5 Go

Go (also called Golang or Go language) was developed by Google engineers to create dependable and efficient software. Most similarly modelled after C, Go is statically typed and explicit. It was designed by taking inspiration from the productivity and relative simplicity of Python, with the ability of C.

Pros:

Cons:

In comparison with Rust, the executables are slightly slower and slightly bigger, whereas the compilation is slightly faster.

3.25 Implementation of Mission Management ODA Component

3.25.1 Context and Problem Statement

How can a mission management ODA component be implemented regarding GUI, UX and structure of mission planning? The way we thought about mission planning in the past was very linear. The user defines a name, duration, location, service needs and gets service options as a result. When thinking of missions related to crisis response, this approach is not sufficient anymore. The orchestration of a crisis response is probably centralized, so a mission will be planned on a very high level. Organizations of a certain area or expertise would plan a sub-mission within the main mission. They will send teams like e.g. firefighters, medical teams, THW etc. into the field and each team might have different service needs.

3.25.2 Decision Drivers

3.25.3 Decision Outcome

The new concept will be decoupled from UCSM to rethink the concept of missions. The following sections describe the topics of the discussion and their outcome.

3.25.3.1 Creating a mission from templates

The users are no experts regarding products from earth observation or SatCom domains, which creates a need for supporting tools. Therefore, it should be possible for the user to use a template to create a mission. This will provide an entry-point on a low level and draws a frame, in which the user can easily define service needs.

3.25.3.2 Creating sub-missions on the basis of a need-to-know principle

Being able to create sub-missions gives actors the freedom to adapt certain aspects of the main mission by default, while being able to manage local situations more specifically. This will follow a hierarchical structure, which also mirrors the organisational structure of crisis responders. The head of the organization will define a main mission as a frame. This might only include data, which are true for all participants, like event type, date, name of the mission. Organizations of certain expertise or of a specific location are system users with a dedicated role, which allows them to plan a sub-mission inside the main mission. The sub-mission will inherit certain data from the main mission, some of which are editable and regarded a default value, and some of them are fixed. The sub-mission planner could then define specific needs for their sub-mission, this can also be supported via templates. The stakeholders share information on the need-to-know principle. E.g. the sub-mission planner might send a team of firefighters to a location and shares that information with the headquarters, but specific information about the team-members won't be shared.

3.25.3.3 Dealing with teams as resources instead of managing them separately

The problem of dealing with teams as an own entity is, that it produces a big overhead. Would it be managed in a dedicated subsystem? Do we need to provide a team-editor? Do we need to deal with a team on member-level? The solution was, to treat and manage teams as resource. The subsystem and API are already there and need only be updated, teams can easily be integrated in existing data models. It also enables to pool and share human resources by describing their skills. By linking any resource or product to another one, one could also easily model requirements for something like a firetruck, that may need internet connection and firefighters alike.

3.25.3.4 Seeing the mission from three different perspectives

Mission as a concept is highly abstract and therefore, very flexible. This enables to provide different views on the mission. These views are:

This helps the user to gather a full overview of the mission.

3.25.4 Compliance

The new mission management approach will be visualized via wireframes. Additionally, we will create a proof-of-concept implementation, that will showcase the main storyline. Depending on time and human resources, some parts might not be implemented in code but will be in future extension of the project.

3.26 MEF Convergence

3.26.1 Context and Problem Statement

MEF is a global industry association which started with the standardization of terrestrial Ethernet and expanded its scope to cover other forms of connectivity services, too. Like PSID, their APIs (called Lifecycle Service Orchestration - LSO) are based on TM Forum and introduce several enhancements. The goal of this decision record is to evaluate possible ways to leverage synergies and improve PSI. In particular, it concerns two aspects:

3.26.2 Decision Drivers

3.26.3 Considered Options

3.26.4 Decision Outcome

Chosen option: Gradually converge schemas and catalogs towards MEF over the next releases, but keep POA/PDM API and Inquiry API separated. Please refer to the following sections for details.

3.26.4.1 Source Schemas

Source schemas are provided by MEF for a wide range of products. The concept is similar to the schemas that are contained in the annex to the PSI-ICD: Each one defines possible (and sometimes required) attributes of a certain product type. This allows the PSS to define what can be shared on the platform and leads to comparability between the providers.

From the currently available schemas, the “Internet Access” type is the only common one. The result of the comparison between the MEF and PSI-ICD “Internet Access” schemas is:

As a first step, we decide to derive a new “Internet Access” schema from the MEF definition. We will remove non-applicable fields and add the SatCom-specific ones. Doing so, we enable the customer to request terrestrial and satellite connection in a very similar way or even request such a service without specifying the technology. Providers adopting PSI will be able to easily communicate with providers already using MEF and vice versa, e.g., for multi-modal services. If necessary, TM Forum adapters can easily be implemented as well.

MEF does not support any product types similar to raw bandwidth services and terminal resources. Therefore, we will re-create our existing schemas to resemble the same structure. Additional input that we received since the initial definition may be used to improve them at the same time.

3.26.4.2 Intermediate and Contextual Schemas

When it comes to application of a schema, the approaches of MEF and PSI differ more:

Figure 3.60 shows an exemplary product catalog with the source schema at the top and two products (of different providers) using intermediate schemas. It is reduced to a minimal set of attributes, from which two are highlighted:

Additionally, each provider is able to define additional attributes. For fields that are not required by the source schema, they may skip them completely (`ipAddress`) or set them to required for their own product (`forwardCIR`), which enforces the customer or PSS to set a value. Just like with the current characteristic approach in PSI, the PSS may decide if and how those are displayed to the customer.

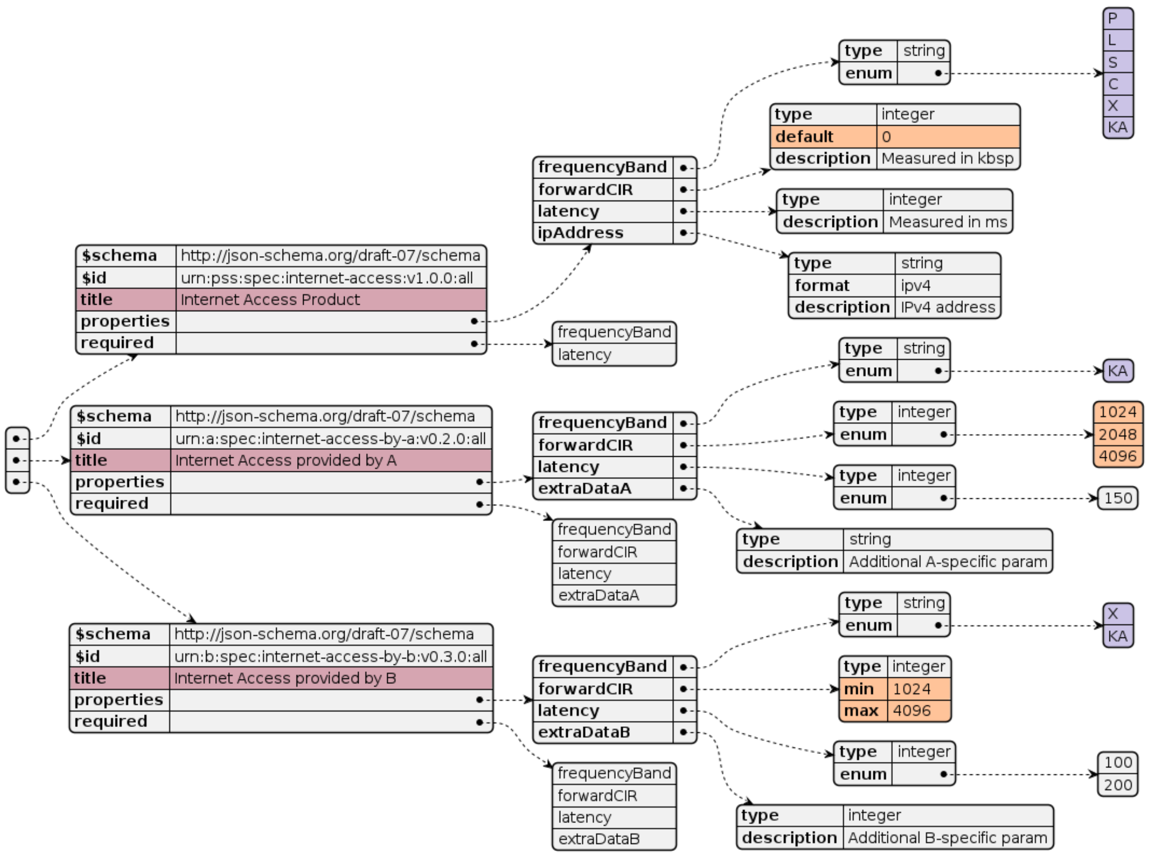


Figure 3.60: Derived Schemas

This approach has the advantage of being much more descriptive and easier to maintain in a pure digital ecosystem than the characteristic-based modelling we are currently using. For providers that are not fully digitalized, the PSS could offer a very similar UI to register products and generate those schemas under the hood. The same goes for users and inquiries: The UI would abstract away the definition of a schema, but the backend would generate it and use it for matchmaking.

Since this is a major change, the implementation of this is not feasible for the upcoming release. Therefore, this release will still be a hybrid approach based on TMF characteristics, but with MEF compatible schemas. A task will be put into the backlog to make full use of intermediate and context schemas in future releases. It is still open for decision whether this will be a full adaptation of MEF APIs, or if we can use schema-based modelling that was introduced in newer releases by TM Forum as well.

3.26.4.3 Product Offering Availability And Pricing Discovery Management API

The MEF “Product Offering Availability And Pricing Discovery Management API” is a combination of multiple steps: The customer deliberately defines a product configuration (without checking the catalog) and the API checks if a product is available at the specified location and looks up a matching offering. While this sounds similar to the PSI “Customer Inquiry API”, the differences are significant:

While some points could be mitigated, adaption of this API does not promise any additional value for PSI. If the schemas converge as decided above, a PSS or provider can use both APIs given the respective use-case.

3.26.5 Compliance

Backlog items will be created to

3.27 Advanced OpenAPI Specification Patching

3.27.1 Context and Problem Statement

While updating the baseline for our APIs to TMF OpenAPIs v5, we encountered some issues with PATTY:

These problems multiplied with the complexity introduced in TMF v5 and even more when retrofitting the APIs that did not receive an update by TMF yet.

3.27.2 Decision Drivers

3.27.3 Considered Options

3.27.4 Decision Outcome

Chosen option: “Write a new transformation tool for OpenAPI specifications”, because the common transformation file only solved parts of the problem and no other tool was found to implement a similar workflow.

The new transformation tool was completely written for Gradle. It supports a set of transformations that are tailored to OpenAPI specifications, where PATTY was for all types of JSON data. This also means that we can now directly transform OpenAPI specifications in YAML format instead of having to convert them to JSON first. In addition to the previously available `add`, `remove` and `replace` rules, the awareness of OAS made the following additions possible:

All these transformation rules are written in a generic way in `buildSrc`. They are then tailored and applied to each API in the root `build.gradle` with one task (instead of one transformation file) per API. It is possible to move these tasks into their own build-script as well, but not done yet to preserve the general workflow.

The tool has already proven useful in other areas. It helped to improve the generation of the mock-up code (API and model) and reduced the need for manual intervention. In the future, it may be further developed and maintained decoupled from PSI itself to allow others to similarly tailor the PSI APIs to their system.

3.27.5 Compliance

The new tool completely replaces the PATTY solution with immediate effect. All invocations of `patty` in the project are replaced by calls to the new tool.

3.27.6 Analysed Candidate Requirements

3.27.6.1 Not Considered

The following candidate requirements were analysed but not considered for the PSI:

3.27.6.2 Implemented

The following candidate requirements were analysed and implemented (for reference see [PSI-RTM]):

4 List of Proposed Decisions

The following sections compile the list of management decisions that are proposed, but not yet decided by the PSI team.

There are currently no decisions in this category.

5 List of Rejected Decisions

The following sections compile the list of management decisions where all options were rejected by the PSI team. Though they did not introduce any change, they are important to understand why the scope has been limited.

5.1 UI Prototyping Tool

5.1.1 Context and Problem Statement

The API is defined using an OpenAPI definition and tested using ruby scripts. This makes it hard to understand it, especially because of the high nesting of the JSON schema. We want to create a basic user interface to visualize the concepts and let interested parties discover it autonomously. When possible, it should also allow for import/export of XLSX files.

5.1.2 Decision Drivers

5.1.3 Considered Options

5.1.4 Decision Outcome

It was decided not to provide a dedicated prototype UI for the interface, because the maintenance does not justify the little value added by it. The interface will be showcased in actual software at a later stage. Appsmith and Angular Material may be considered for other objectives in the future.

5.1.5 Compliance

N/A

5.1.6 Pros and Cons of the Options

5.1.6.1 Appsmith

> Appsmith is the open-source framework that lets your team build custom internal applications like dashboards, admin panels, CRUD apps.

Appsmith is not meant to be deployed with a predefined set of applications. It is possible to do so by including the data directory and our backend in a custom docker image, which has a total size of about 2 GB.

5.1.6.2 Budibase

> Budibase is an all-in-one low-code platform for building, designing, and automating business apps, such as; admin panels, forms, internal tools, client portals, and more.

5.1.6.3 Angular Material

Angular is a development platform, built on TypeScript. As a platform, Angular includes:

Deploying the application is possible via usual tools, e.g. docker. We can decide to do this in separate containers or a single one (about 700 MB).

5.2 Resource and Service Order Management

5.2.1 Context and Problem Statement

PSID currently utilizes the TM Forum *Product Ordering Management API* as described in TMF622 V4.0 to process orderings. Herewith customer orders, that were created based on available offerings (later also RFQs, etc.) are shared with corresponding providers. In the case that the provider signed up as a recipient for the event support of orders, PSS even distributes these orderings to the provider's system.

A closer look at the offerings reveals that a product offering consists of one or, in the case of bundled offerings, several product specifications. These product specifications are in turn build of either one or more service specifications or one or more resource specifications. Additionally, service specifications could be built from a hierarchy of further service specifications and/or resource specifications. To reflect this complexity and accompany the lifecycle of each of the resulting service and resource, TM Forum offers a matching subdivision of product orders into service and/or resource orders. We have to evaluate their benefit for the PSS-to-Provider interface.

5.2.2 Decision Drivers

5.2.3 Considered Options

5.2.4 Decision Outcome

It has been decided to **not** implement TMF641 and TMF652 and proceed with TMF622 to share and route (via event management) product orderings.

Since the currently supported interfaces at the product order level already transport all the necessary information to the provider side and an enhanced lifecycle tracking of decomposed orders is currently not required (not part of the candidate requirements), we should keep the number of interfaces that have to be supported by the providers low. As the result, any necessary refinement of the product order is performed only on the provider side.

5.2.5 Compliance

N/A

5.2.6 Pros and Cons of the Options

5.2.6.1 TMF822

TMF822 *Product Ordering Management API* is already part of PSID and allows sharing orders with providers within a PSS or even with PSS clients via the event management.

Processing of the product order on the provider side results in the creation of suitable products, services and resources. These inventory entities are linked to each other and to the original product order via the product. In addition, resources are used by the provider to supply the configuration parameters needed to set up the hardware on the customer side.

Should it appear necessary that attachments are also needed on service and/or product level - maybe even for product orders -, then we would expand the existing schemata by an additional attachment reference.

5.2.6.2 Introducing TMF641 and TMF652

Introducing TMF641 *Service Ordering Management API* and TMF652 *Resource Order Management API* would add two new APIs to PSID to deal with orderings. This would mean tripling the number of APIs to be supported by the providers.

With the help of TMF641 and TMF652 the decomposition of the order could be available in the PSS and therefore for the customers, too. So, the PSS would publish more details about the fulfilment state of the sub entities of an order. In addition, TMF641 *Service Ordering Management API* introduces new associations for e.g.

They are all used internally by the provider to fulfil the service order. It would be questionable whether the provider always wants to share these details with their customers.

Taking into account that all necessary information is already shared with the provision of the product order, the introduction of TMF641 and TMF652 seems to be another level of complexity that is not needed or even desired for PSS-to-Provider interfaces.

5.3 Adaptation of TMF645 and TMF679 (V0.1)

5.3.1 Context and Problem Statement

Should the TM Forum APIs “TMF645 Service Qualification Management” (SQM) and “TMF679 Product Offering Qualification Management” (POQM) be adapted for (Gov)SatCom use cases as part of PSI?

The APIs in question help in checking the technical and commercial feasibility of product offerings, i.e., of services and bundled product offerings. The decision revolves around whether these APIs should be integrated as part of the PSID ICDs and if they would need adaptions.

5.3.1.1 Configuration, Standardization, and Matchmaking

A PSS should support both experts with precise requirements and inexperienced customers with general requirements, suggesting the need for a robust product configurator within the system. Since product, service, and resource specifications will be and need to be standardized through JSON schemas, the space of combinations is already reduced and the comparability between results is enhanced. However, the diversity in configuring bundles can still complicate comparisons with similar offerings, indicating the utility of a CPQ (Configure, Price, Quote) engine.

A configurator for a PSS can consist of three steps:

The more details are specified for the inquiry, the more restrictions can already be applied by the matchmaking. The less detailed the inquiry is, the more generic are the results by the matchmaking that require additional configurations. Additionally, for complex products or missions consisting of multiple products, the variety of possible configurations can grow exponentially. However, only a fraction of these might be technically or commercially eligible, restricting the space of potential combinations.

TMF645 and TMF679 provide checks for eligibility, service compatibility, and availability. In the standard TM Forum processes, this check happens only after the configuration, however, it could be applied at any time in the process described above. Thus, the two qualification APIs can complement the Inquiry API and the matchmaking process of a PSS.

Additionally, this is potentially useful for order modification processes, i.e., to verify that a change request is eligible before the order is touched at all.

The core data models of the qualification APIs may not need significant adjustments as they mainly consist of a qualification object, referring to the offer, that provides either a positive verification result of the eligibility or an alternative option.

5.3.1.2 Existing Solutions

UCSM has a straightforward relationship between product specifications and offerings, without extensive configuration options. The adaptation already requires the addition of configuration functionalities to accurately represent data from providers like SES and Inmarsat. Implementing the eligibility check as part of the ordering process would be required additionally and might result in a lot of unforeseeable effort as this is work-in-progress.

In UCSM, there is a one-to-one (1:1) relationship model, which means:

This flexibility can accommodate complex product configurations and varied customer needs, which is essential in the dynamic (Gov)SatCom market.

Configuration options like optional add-ons or alternatives (e.g. different terminals for the same internet access service) add complexity but also flexibility, allowing for more tailored solutions to meet specific customer requirements. However, they also depart from the existing simple mapping structures, necessitating a system capable of handling these intricate relationships and requiring the corresponding logic for SQM and POQM.

REACH on the other hand has a different business model implemented. If the system was able to answer a request for qualification needs to be clarified.

5.3.1.3 Information Gaps

Certain critical information, like satellite capacity or peak transmission quota required for the effective functioning of the TMF645 and TMF679 APIs may not be readily available or shared by CSPs. This is because some data, such as the exact capacity of a satellite or the reserves allocated for peak transmission times, may be considered proprietary or confidential business information. Without access to this detailed information, the PSS's ability to accurately determine the technical feasibility and commercial suitability of product offerings could be compromised.

In satellite communications, where service eligibility and product offering feasibility are contingent on a myriad of technical parameters like bandwidth availability, satellite coverage, and ground station compatibility, the absence of complete information can significantly impact the qualification process. The APIs rely on accurate data to assess if a service can be added to an existing bundled product offering or if an upgrade, like increasing bandwidth, is viable both technically and commercially.

For the APIs to function as intended, there needs to be a clear and reliable source of information. This source must provide up-to-date and comprehensive data regarding service eligibility and product offering configurations. If implemented by the PSS, it might require CSPs to disclose more information than they typically would, or it could involve developing a secure and mutually agreeable method of sharing sensitive data to ensure that the qualification APIs can operate effectively. On the other hand, if the APIs are implemented by the provider, they don't have to disclose any information apart from the above mentioned yes/no/alternative response.

Adding this into the considerations highlights the dependency of the APIs' functionality on external information sources. It underscores the necessity for collaboration between the providers of satellite services and the entities managing the qualification APIs to ensure that decision-making regarding service qualification and product offering adaptation is well-informed and accurate. This collaboration might involve contractual agreements, the establishment of standardized data-sharing protocols, or the implementation of secure, real-time data exchange mechanisms to minimize the information gaps and facilitate the smooth operation of the APIs within the (Gov)SatCom domain. Thus, enabling these APIs might result in a need to involve the CSPs' systems for *every* request, which would result in a lot of over-head traffic.

5.3.1.4 Potential for Zero-Touch Automation

The matchmaking process, that based on customer inquiries, can potentially create tailored bundles, reducing manual work for CSPs. This advanced level of automation within the (Gov)SatCom service provisioning and management aims to minimize or entirely eliminate manual intervention from CSPs. The concept is particularly relevant in the context of the TMF645 and TMF679 APIs, which could be used to automate the process of service qualification and product offering qualification, i.e., testing the eligibility of the configured bundles automatically.

In the (Gov)SatCom industry, the provisioning of services can be complex due to the intricate technical parameters involved and the need to align with the varying capabilities and availability of satellites and satellite networks. Zero-Touch Automation seeks to simplify this process by allowing systems to automatically handle SQM and POQM based on predefined rules and dynamic data inputs. The concept of Zero-Touch Automation in the context of the TMF APIs would ensure that the service qualification and product offering processes are not only more efficient, but also scalable. As the number of satellite services and potential configurations grows, the ability of the PSS to handle these without manual input becomes increasingly valuable. It allows for rapid scaling of services, quicker response to market demands, and a more seamless customer experience. Subsequently, the PSS enables Zero-Touch Partnering, i.e., to create product bundles with products of other companies/organizations seamlessly.

In essence, Zero-Touch Automation represents a significant leap forward in service management automation, promising efficiency and scalability but requiring careful planning, sophisticated technology, and close cooperation between all stakeholders involved.

5.3.2 Decision Drivers

5.3.3 Considered Options

5.3.4 Decision Outcome

Chosen option: “Not adapting TMF645 and TMF679”, because the major implementation effort required does not justify the marginal benefits at the current stage.

5.3.5 Compliance

N/A

5.3.5.1 Positive Consequences

5.3.5.2 Negative Consequences

5.3.6 Security Considerations

5.3.6.1 Adapting TMF645 and TMF679

5.3.6.2 Not Adapting TMF645 and TMF679

5.3.6.3 CIA Comparison for All Given Options

Option #	Confidentiality	Integrity	Availability
1	Low	Medium	Medium
2	Low	Low	Low

Table 5.1: CIA table for different options

5.3.7 Links

5.4 Implementation of (Gov)SatCom Hardware Logistics Process (V0.1)

5.4.1 Context and Problem Statement

We need to decide if and how to implement a logistics process for SatCom-related hardware, considering that TM Forum lacks a proper process description for this. The current system design does not have a process for logistics of hardware. In addition, the ability to book 'off the shelf' services that automate hardware selection and shipment qualification is required to improve system acceptance and reduce time to market.

The logistics process for SatCom-related hardware requires a systematic approach to address the unique challenges of shipping physical products, which may range from small components to large equipment. The process must be capable of integrating various factors, such as customer preferences, legal restrictions, shipment options, and cost calculations. Considering the TM Forum APIs and the need for a standardized approach, the process to be implemented should include the following steps and considerations:

A PSS should allow for booking off-the-shelf services/products, i.e., hardware offerings should allow for automatic qualification and selection in the system, promoting zero-touch automation. For instance, if a customer selects a service that includes a satellite terminal, the system should automatically provide shipping options and costs.

5.4.2 Decision Drivers

5.4.2.1 TM Forum API Integration

The state of the available TM Forum Open APIs varies widely, as some APIs are still under development and some are already obsolete. Additionally, there is no data model available in the SID to account for logistics properly, although the available TM Forum documents and Open API definitions hint that this is needed and will be implemented in the future.

TM Forum Open APIs investigated are:

5.4.2.2 Data Requirements

Assuming the PSS shall be able to at least make an educated guess for the shipping costs, it needs an API to get the corresponding information for a product offering containing physical goods from the CSP. For example, the following data would allow for a minimal setup that can even be used for booking of-the-shelf products. Note that the following tables are only an example that is not meant to be implemented as is, but a basis for further discussions and investigations.

Cargo Classes can be freely defined by the provider. Each terminal is then assigned to a class, the PSS doesn't really have to know why.

Cargo Class	Description
A	$<$ 20kg, EU-UNCLA
B	$<$ 50kg, EU-UNCLA
C	$>$ 50kg or EU-CLA

Table 5.2: Example for different cargo class options.

The CSP could generate a list similar to the one below for the PSS to account for shipment. This might require the CSP to validate the qualification of the shipment.

Destination Country	Cargo Class	Shipment Possible	Shipping Restrictions	Shipping Cost	Shipping Time (min)	Shipping Time (max)
DE	A	YES		10,00 €	3 Days	5 Days
DE	B	YES		100,00 €	1 Week	2 Weeks
DE	C	YES		1.000,00 €	1 Week	2 Weeks
DZ	A	LIMITED	ECCN necessary	500,00 €	2 Weeks	4 Weeks
DZ	B	LIMITED	ECCN necessary	2.000,00 €	3 Weeks	2 Months
DZ	C	NO	Shipment to EU only			
RU	A	NO	Sanctions			
RU	B	NO	Sanctions			
RU	C	NO	Sanctions			

Table 5.3: Example for a logistics table including estimated prices and shipping duration.

This list will be much longer in reality. If a country/class combination is not assigned, the PSS shall inform the user that shipment has to be checked manually.

Note that this is connected to candidate requirements stating the availability of services and products in a given region, especially P&S_016.

Additional data required for the process might include:

Note that, if the CSP defines a list similar to the one in 5.3, they would also be responsible to keep it up-to-date.

5.4.3 Considered Options

5.4.4 Decision Outcome

Chosen option: "Postponing the logistics process implementation," because there is no immediate candidate requirement. The remaining time of the project needs to focus on the implemented APIs and the demonstration thereof.

Delivery of hardware can be omitted when assuming that the customer has their own, compatible hardware in place already.

5.4.5 Compliance

N/A

5.4.5.1 Positive Consequences

5.4.5.2 Negative Consequences

5.4.6 Pros and Cons of the Options

5.4.6.1 Postponing the Logistics Process

5.4.6.2 Developing a New Data Model and APIs

5.4.6.3 Using the Service Catalog for Delivery

5.4.7 Security Considerations

The security implications of postponing the physical logistics process are minimal.

5.4.8 Implications for the Scope

Shipment restrictions should already be considered when making a product or product offering available.

5.4.9 Links

5.5 Advanced Billing

5.5.1 Context and Problem Statement

At the current project phase, payment is considered to be done directly between customer and provider. Only the outstanding debt and an indication of successful settlement are communicated with the PSS via the Customer Bill API. This is sufficient for a pure broker scenario, where the PSS is just an intermediary.

But a PSS may want to handle payments for products, that are acquired through the platform, for different reasons:

While the first two are applicable to any scenario, the last one can be considered to be specific to the hub scenario. In any case, the customer has a contract with the PSS and the PSS has contracts with the providers[12]. This leads to two different steps in the billing workflow as depicted in figure 5.1. Note that PSI does **not** cover the actual money flow.

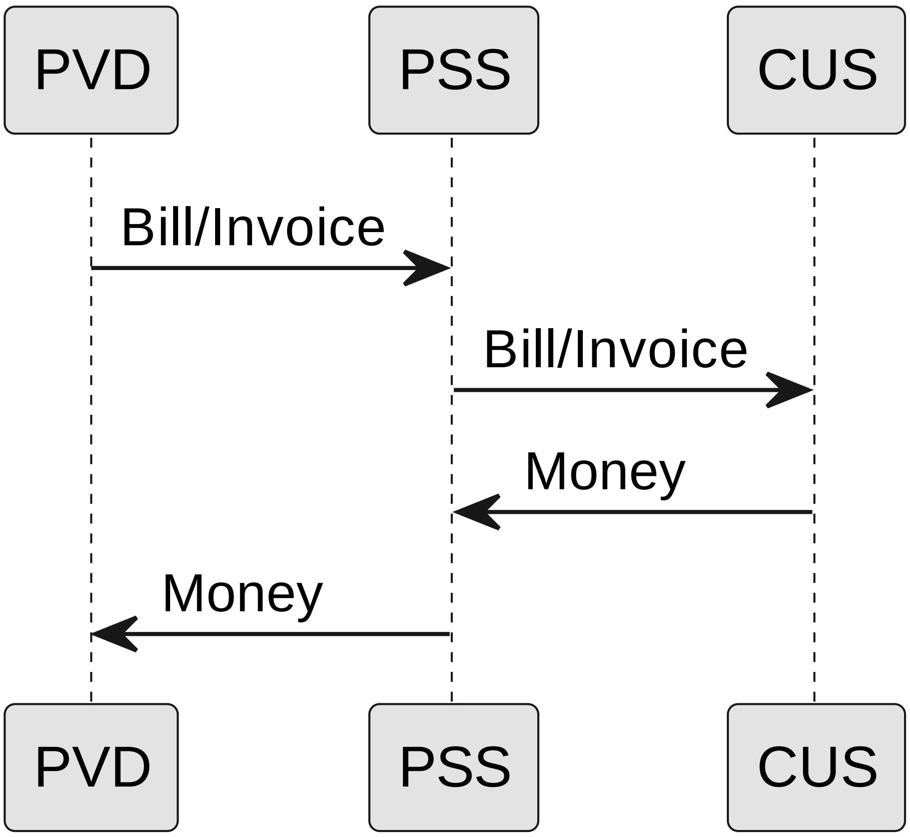


Figure 5.1: Billing workflow with PSS involvement.

5.5.2 Decision Drivers

5.5.3 Considered Options

5.5.4 Decision Outcome

The current set of APIs already enables the implementation of arbitrary billing workflows. Therefore, the `TMF666 Account Management API` will not be adapted for PSI. Additionally, there are no candidate requirements that

[12] Additional contracts between customer and provider are possible, but excluded from the description to preserve readability.

imply a (Gov)SatCom specific adaptation. At some point in the future, a decision might be made to implement it. An anticipated reason might be to allow access to summed up balances instead of single bills, especially when they are using virtual currencies.

With the existing `PSI678 Customer Bill Management API`, the provider can send the due amount for booked services to the PSS. Depending on the concrete implementation and use case, the bill is either linked to a single customer, multiple customers, or the PSS itself via the `relatedParty` property. The PSS can then either pass the bill as-is to the named customers (pure broker scenario) or create derived bills which are then passed on to the individual customers. It also allows references to billing accounts in case `TMF666` is implemented.

5.5.5 Security Considerations

Both discussed APIs carry sensitive data and therefore have to be protected accordingly. Otherwise, attackers could extract payment information or even initiate unwarranted payments. As stated in [PSI-ICD], the security measures depend on the needs of the system and are therefore not discussed in detail here, as they have to be applied to a different communication layer.

5.5.6 Implications for the Scope

While the interface enables joint missions, the concept of a mission is kept internal to the PSS.

The interface allows defining arbitrary debtors for a bill and multiple debtors for a bill.

5.6 Introduction of a common model

Technical Story: Define a common PSID-Model

5.6.1 Context and Problem Statement

How can *model conflicts* be resolved when generating Java code based on 1+n PSID swagger definitions?

5.6.1.1 Detail

5.6.1.1.1 Current State

Problems with TMF definitions and/or Patty adjustments:

The current state of the PSI “transform-and-build-workflow” for the PSID and mock-up artefacts contains the following main steps:

Above relates to Swagger (JSON) File Patching for ICD Generation

5.6.1.1.2 Problem Description

Multiple swagger definitions contain the same common components. If you only want to use one definition, you won’t notice any difference. If you use at least two with common components, this can lead to problems.

In most cases, the swagger definitions for common components are ‘nearly’ the same. ‘Nearly’ means: the structure and naming of the properties maps fully, however additional fields may be different in detail. There is a pipeline validation step established (`generateRestApiSchemas`) to validate this on a structural level based on the component name. However, this validation succeeds if structural equality is given. This is error-prone because common components need to be duplicated in multiple swaggers via copy & paste on any change by developer. Even with a combination of at least two APIs, the generated code can differ for each generation, depending on which swagger definition was used first.

The following example focuses this behaviour via a difference on the `description` field:

First example of a swagger definition for a sample object (omitting the enclosing elements).

```
"FooComponent": {
    "type": "object",
    "properties": {
        "bar": {
            "type": "string",
            "description": "The foo-bar description"
        }
    }
}
```

In opposite here a snippet of a second swagger file with a redundant definition, however with a difference in detail:

```
"FooComponent": {
    "type": "object",
    "properties": {
        "bar": {
            "type": "string",
            "description": "Bar explanation in detail.",
        }
    }
}
```

Above definition is used for generation of a `FooComponent.java`. If the code generation runs one after the other without deterministic sequence, it results in code which differ and/or lead to a conflicting code-base (*Step 4*). Also, this generation only produces one `FooComponent.java` file at the end.

In extended cases, it is also possible to change a common component like above on structural level in one swagger file together with a missed change of the same component in another swagger file. Here, classes like `FooComponent.java` and `FooComponent_1.java` will be generated by the OpenApi generator. Also, the generated classes may differ by class properties. Or, if the combined generation runs over multiple definitions, the classes will not be valid for *all used* OpenAPI definitions.

A decision needs to be made on how to ensure that the API definitions provided for release, have the same common components. This means that instead of cascading changes to one component in a swagger file to others via manual copy and paste, an automatic method should be found to standardise these components to a common version across different definitions/files.

5.6.2 Decision Drivers

5.6.3 Considered Options

5.6.4 Decision Outcome

TBD: Chosen option: “{option 1}”, because {justification. e.g., only option, which meets k.o. criterion decision driver	which resolves force {force}	…	comes out best (see below)}.

Table 5.4: Unnamed Table

5.6.5 Compliance

5.6.5.1 Positive Consequences

5.6.5.2 Negative Consequences

5.6.6 Pros and Cons of the Options

5.6.6.1 OpenAPI - Reference Object

Usage of references (https://learn.openapis.org/referencing/) to a separate common schema file.

5.6.6.2 PSID swagger streamlining tool

A Gradle task / groovy script could run after patty transformation to do the following approach:

5.6.6.3 Harmonize all patty rules and improve build validation

5.6.6.4 Locally Maintained Data Model

Maintain required data models in single schema file per component:

5.6.7 Links

Last Page of Document