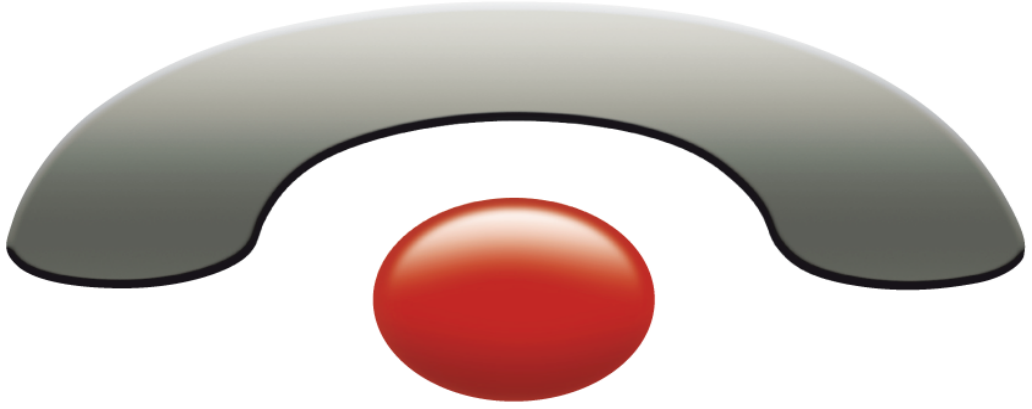
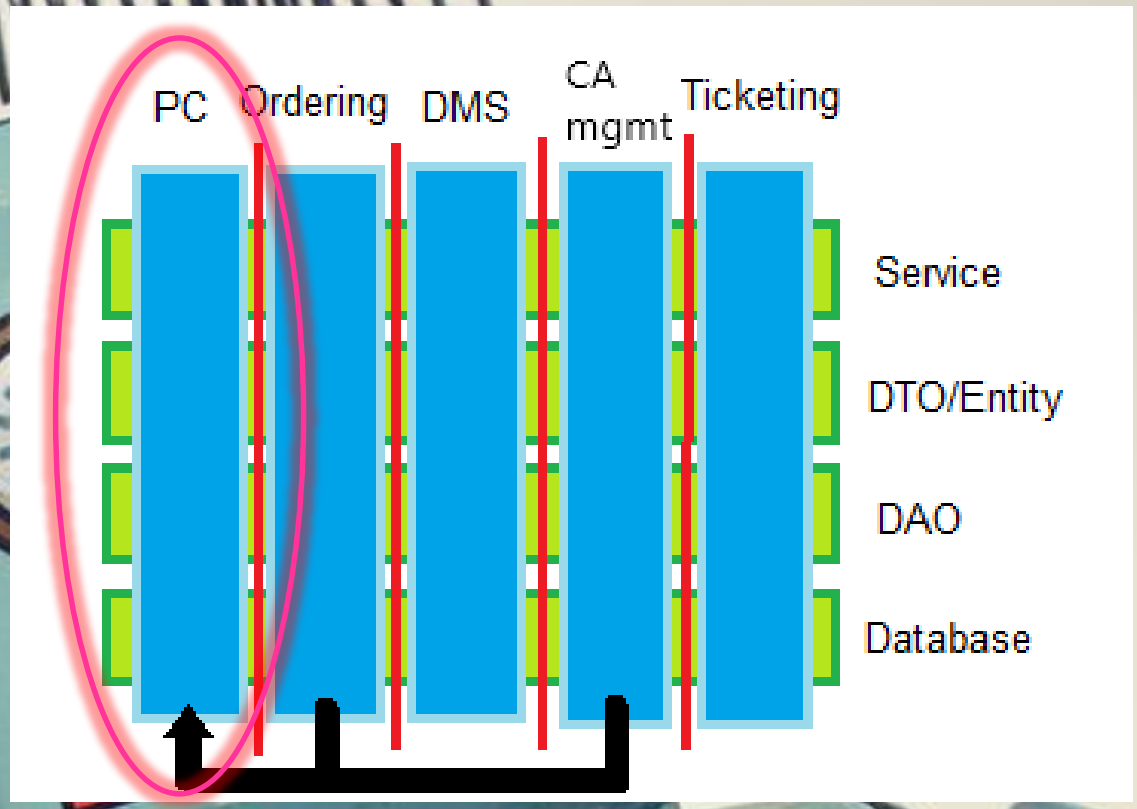


way forward

Emeldi Commerce® Technical overview

Emeldi Commerce® 6.x
Microservices architecture

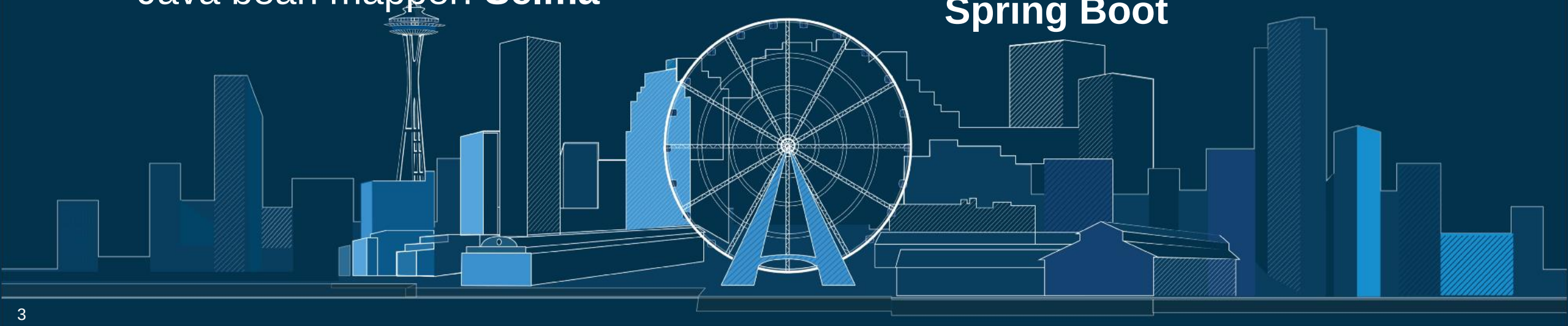
- Microservices architecture
  - Split modules and make boundaries
  - Variant of SOA, loosely coupled fine-grained services
  - Ability to **independently deploy functional blocks** (modules)
- Product Catalogue is one isolated microservice with no dependency on other microservices
  - ➔ may be deployed and used as a standalone component



Technology stack overview

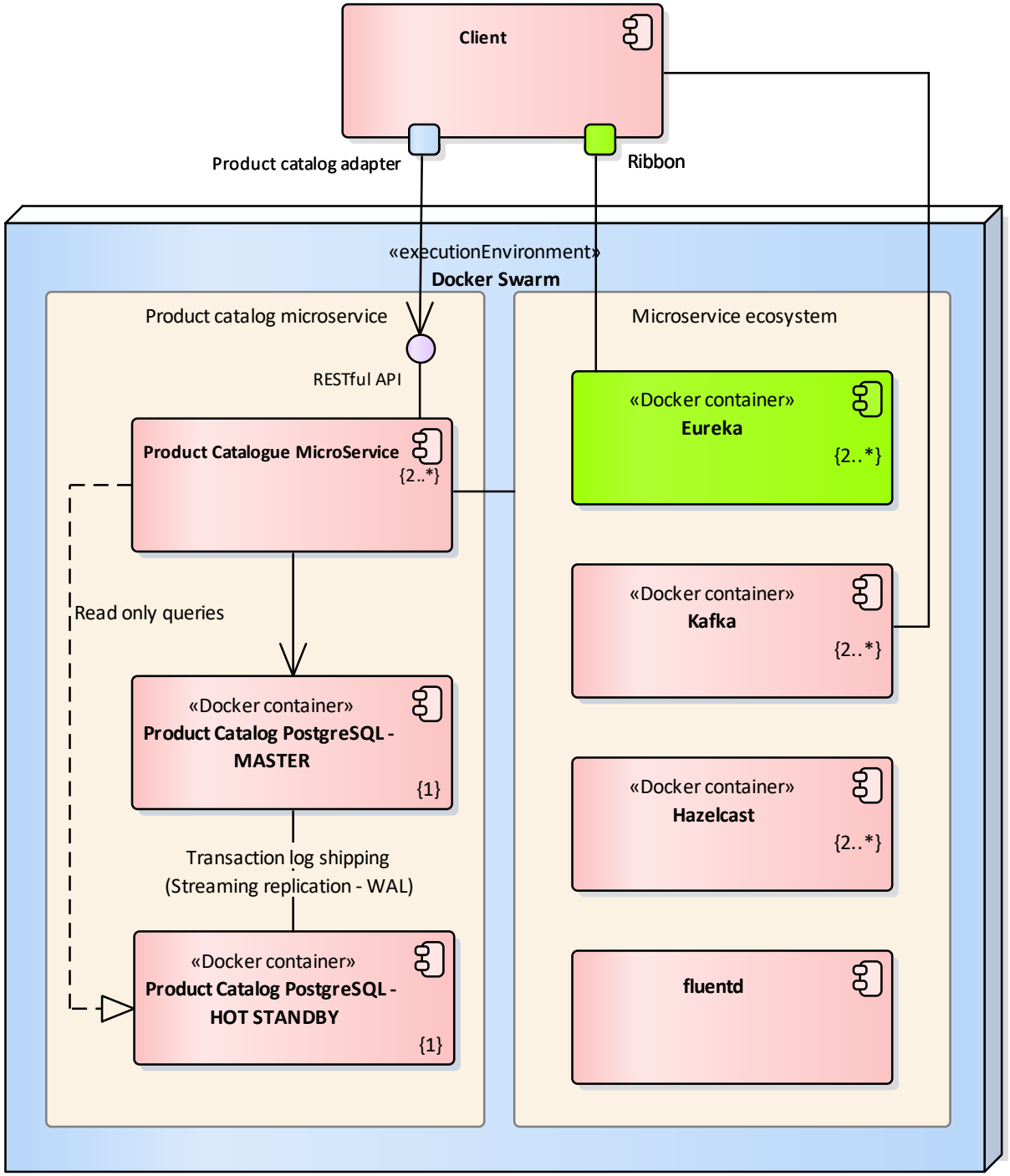
- Server-side programming language: **Java 8**
- RDBMS: **PostgreSQL** 9.6/10, **Oracle** 11g/12c Standard/Enterprise
- REST API documentation: **Swagger**
- Java bean mapper: **Selma**

- Distributed in-memory cache: **Hazelcast**
- Messaging: **Apache Kafka**
- Service registry: **Eureka**
- Load balancer: **Ribbon**
- Application framework: **Spring Framework**, **Spring Cloud**, **Spring Boot**



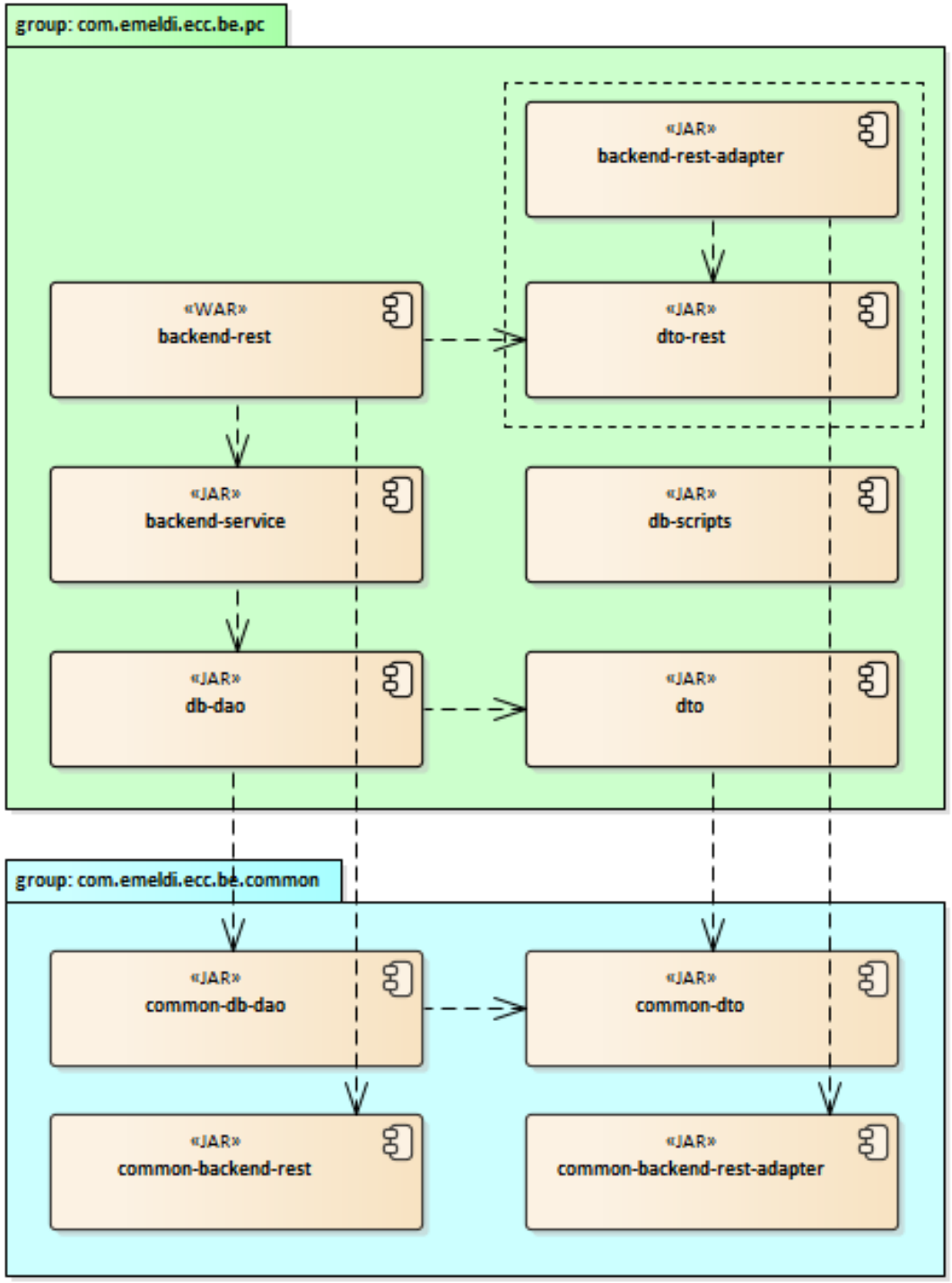
Deployment

- Focused on **robustness** and horizontal **scalability**
- Docker Swarm as cluster management and auto-scaling
- Product catalogue microservice
  - Several running instances to be fault-tolerant and support high-loads
- Product catalogue database
  - Master + hot standby



Internal modularization

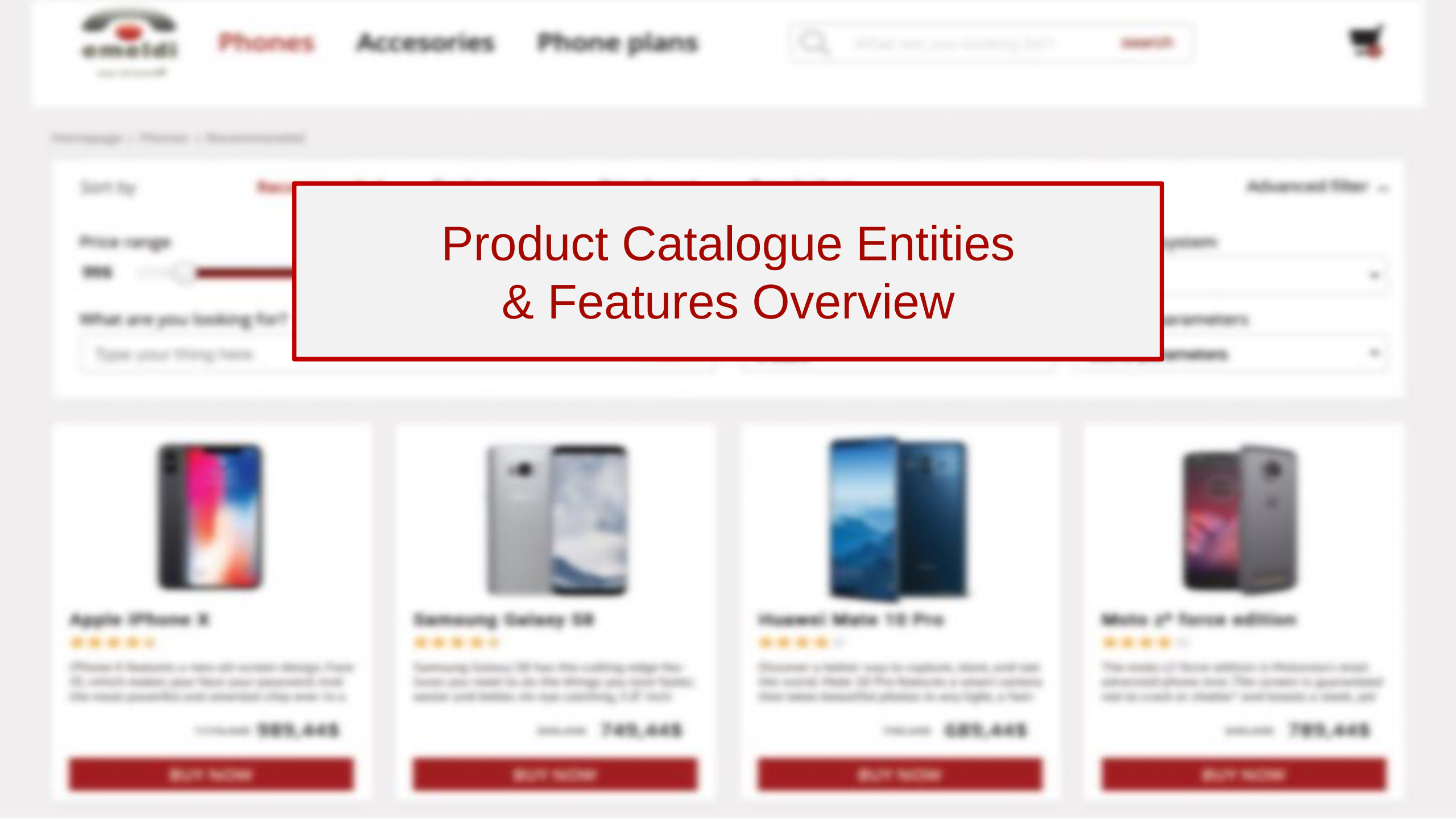
- Layers:
  - Common shared libraries
  - Backend components
    - Database layer
    - Service layer
  - Client adapters for easy integration

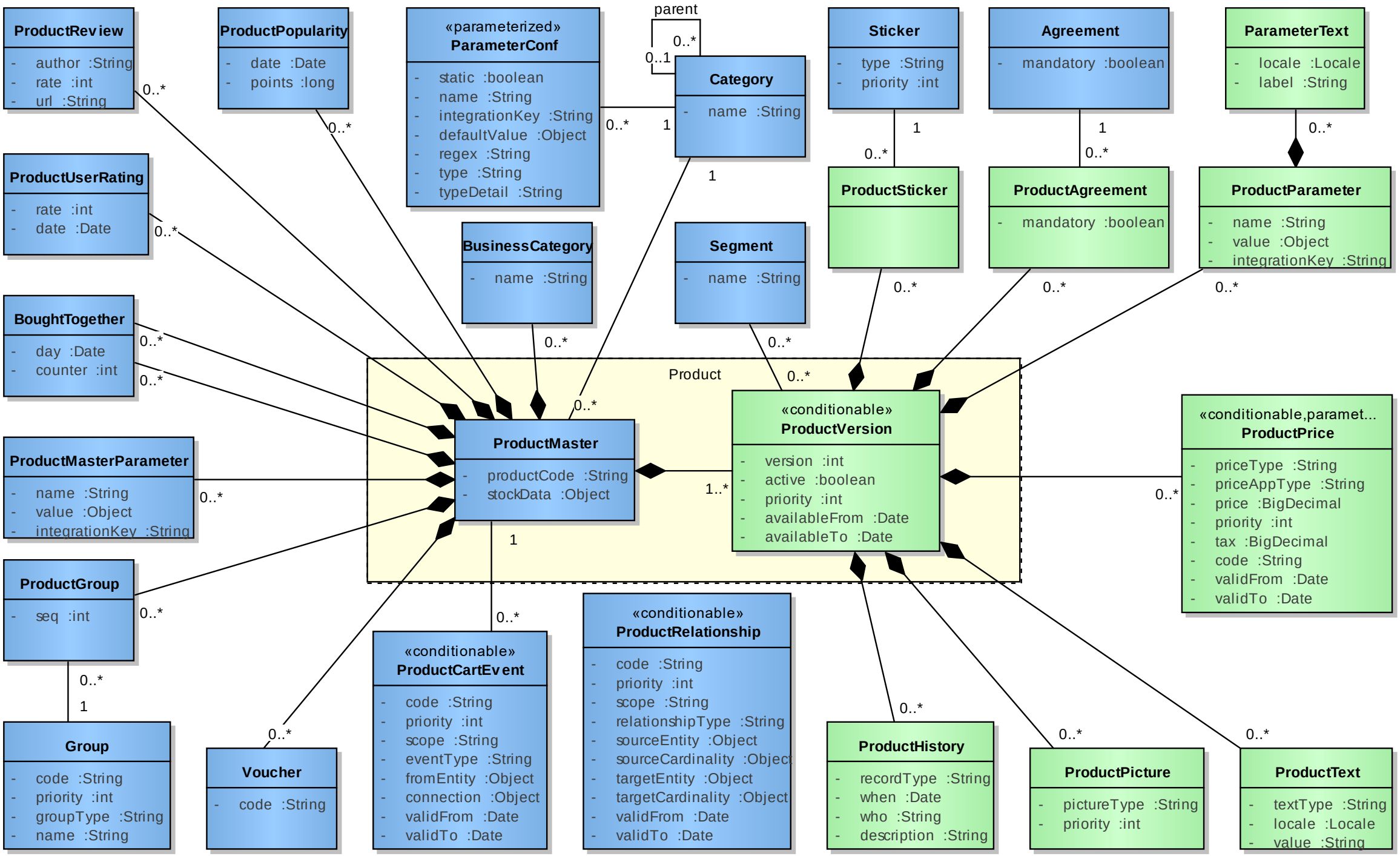


Integration

- Open system and easy to integrate with several options:
- Exposed **REST API**
  - Follows TM Forum best practices
    - TMF630+TMF631 (REST_API_Design_Guidelines_Part_1 & 2)
  - Emeldi is member of TM Forum
- **Adapters to REST API**
  - provided for easy integration from JAVA applications
- Using **Events** distributed by Kafka

Product Catalogue Entities & Features Overview

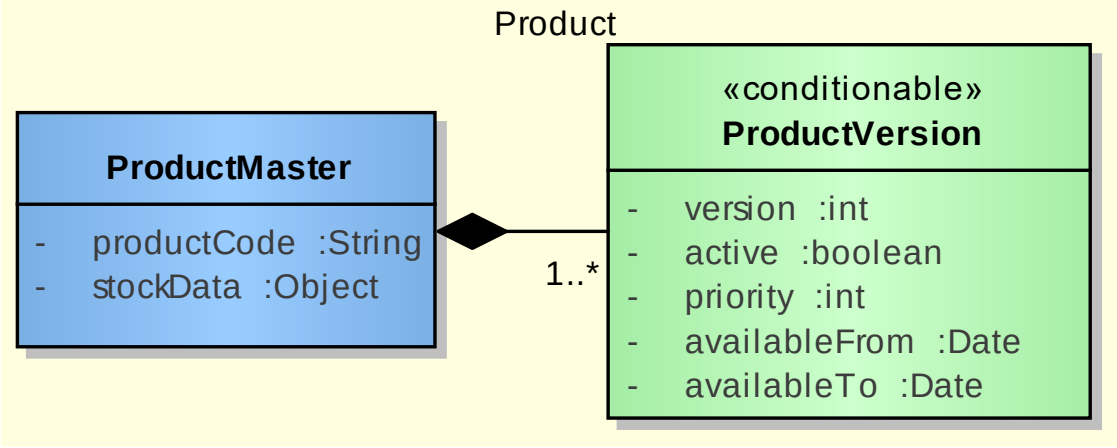



ProductReview
- author :String
- rate :int
- url :String
ProductPopularity
- date :Date
- points :long
«parameterized»
ParameterConf
- static :boolean
- name :String
- integrationKey :String
- defaultValue :Object
- regex :String
- type :String
- typeDetail :String
parent
0..*
0..1
Category
- name :String
Sticker
- type :String
- priority :int
Agreement
- mandatory :boolean
ParameterText
- locale :Locale
- label :String
ProductUserRating
- rate :int
- date :Date
BusinessCategory
- name :String
Segment
- name :String
ProductSticker
ProductAgreement
- mandatory :boolean
ProductParameter
- name :String
- value :Object
- integrationKey :String
BoughtTogether
- day :Date
- counter :int
Product
ProductMaster
- productCode :String
- stockData :Object
«conditionable»
ProductVersion
- version :int
- active :boolean
- priority :int
- availableFrom :Date
- availableTo :Date
«conditionable,paramet...
ProductPrice
- priceType :String
- priceAppType :String
- price :BigDecimal
- priority :int
- tax :BigDecimal
- code :String
- validFrom :Date
- validTo :Date
ProductMasterParameter
- name :String
- value :Object
- integrationKey :String
ProductGroup
- seq :int
Group
- code :String
- priority :int
- groupType :String
- name :String
Voucher
- code :String
«conditionable»
ProductCartEvent
- code :String
- priority :int
- scope :String
- eventType :String
- fromEntity :Object
- connection :Object
- validFrom :Date
- validTo :Date
«conditionable»
ProductRelationship
- code :String
- priority :int
- scope :String
- relationshipType :String
- sourceEntity :Object
- sourceCardinality :Object
- targetEntity :Object
- targetCardinality :Object
- validFrom :Date
- validTo :Date
ProductHistory
- recordType :String
- when :Date
- who :String
- description :String
ProductPicture
- pictureType :String
- priority :int
ProductText
- textType :String
- locale :Locale
- value :String
0..*
1
1..*

Versioning

- Product entity is made of one

  Product Master and 1-N its versions
- **Product Master** = unique product code
- **Product version** is either active or inactive
  - *Multi-active version mode* = more than one active version allowed
- Product Master and associated entities don’t support versioning
- Any change on versioned part will result to the new version
- **Approval process** takes place to switch active version

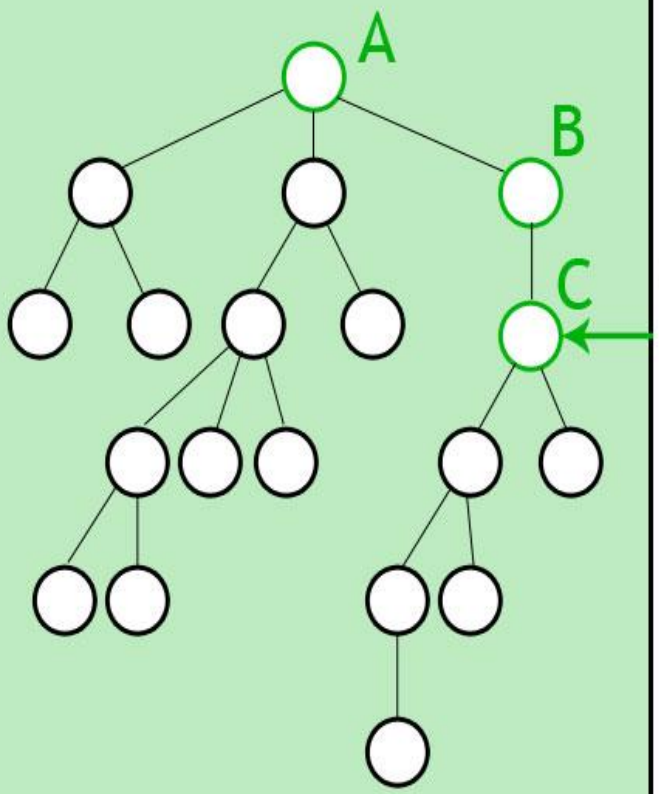


Product Eligibility

- **Eligible product has to:**
  - **Have active version**
  - **Be valid at given time (with respect to Product validity from-to)**
  - **Satisfy Context Rules (see later)**

Technical Categories

- Reflect technical needs when modeling a product, maintained by IT
- Could be applied in business/eligibility rules
- Single inheritance is supported
- Every product master belongs exactly to one technical category (and indirectly to all its parent categories)
- Determines product parameters



PRODUCT

Business Categories

- Reflect business needs when modeling a product, maintained by business
- Could be applied in business/eligibility rules
- No inheritance is supported, business categories constitutes a flat structure
- Every product master may belong to zero, one or more business categories

Product Parameters

- **Technical category** determines list of **product parameters**
  - Full **inheritance** supported
- Various **types** supported
  - Simple types (*string, integer, boolean, date*)
  - Complex types (*Address, Codebook*)
  - Custom types (with provided *serializer*)

categories	*parameters*
A	P_1, P_2=X
∟B	P_3, P_2=Y
∟C	P_4, P_5

C: P_1, P_2 (Y), P_3, P_4, P_5

B: P_1, P_2 (Y), P_3

A: P_1, P_2 (X)

Product Parameters cont.

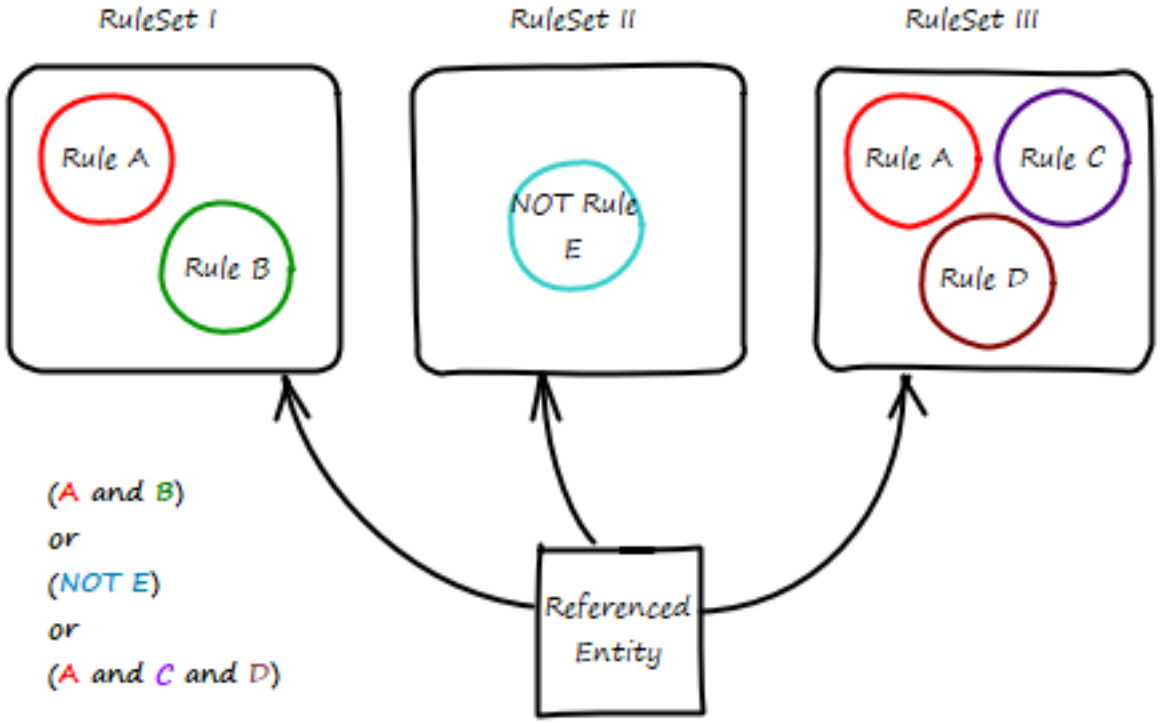
- **Static** / **dynamic** parameters
  - Static with preset value and **unmodifiable**
  - Dynamic **modifiable** with possible default value
- Product parameter **Metadata**
  - E.g. visibility on GUI, GUI input type, mandatory
  - Fully customizable
- Validations
  - Regular expressions out-of-the-box
  - Custom validators
- Localization
  - Localized parameter names, Localized static parameter values

Context Rules

- Context Rules = rules evaluated on dynamic context
- Written in CREL or Groovy
- CREL = Context Rule Expression Language
  - Entity based query language, examples:
    - `account.name = "Karel"`
    - `account.name regexp "^Kar.*$"`
    - `order.personalIDCardValidity > SYSDATE+365`
    - `customer.segment = CustomerSegmentType.CORPORATE`
- Groovy scripts for complex rules
  - May use provided Product catalogue rich java API

Complex rules

- Elementary rules can be combined to complex rules (*Rulesets*) using logical operations AND, OR, NOT
- Rulesets can be associated to any entity, for example to:
  - Evaluate **Product** eligibility
  - Provide eligible **Product price**
  - Apply allowed **Product Relationships**



Prices

- Product can have 0-N Prices of each **Price Type**
- Example of Price Types (but not limited to):
  - OC = One-time Charge, RC = Recurrent Charge, UC = Usage Charge, …
- Every price has a **priority** and **validity** (from-to)
  - Only valid and eligible Price with highest priority is applied (per Price Type)
- Additional attributes:
  - Price value and Tax
  - Code
  - Customizable key-value parameters

Prices – configuration example

- Example Prices for product GENERAL_FEE:
  - One product with **two price types** (OC + RC)
  - Different OC price value based on **time and rule**

Priority	Charging code	Price	Type	Validity	Rule
0	FEE_OC_RESIDENT	1€	OC	till 31.8.2018	`segment=Resident`
0	FEE_OC_RESIDENT	3€	OC	from 1.9.2018	`segment=Resident`
1	FEE_OC_OTHER	5€	OC	*unlimited*	-
0	FEE_RC	1€	RC	*unlimited*	-

- Result:
  - Resident in August and earlier: 1€ OC, 1€ RC
  - Resident after August: 3€ OC, 1€ RC
  - Non-Resident: 5€ OC, 1€ RC

Volume based pricing

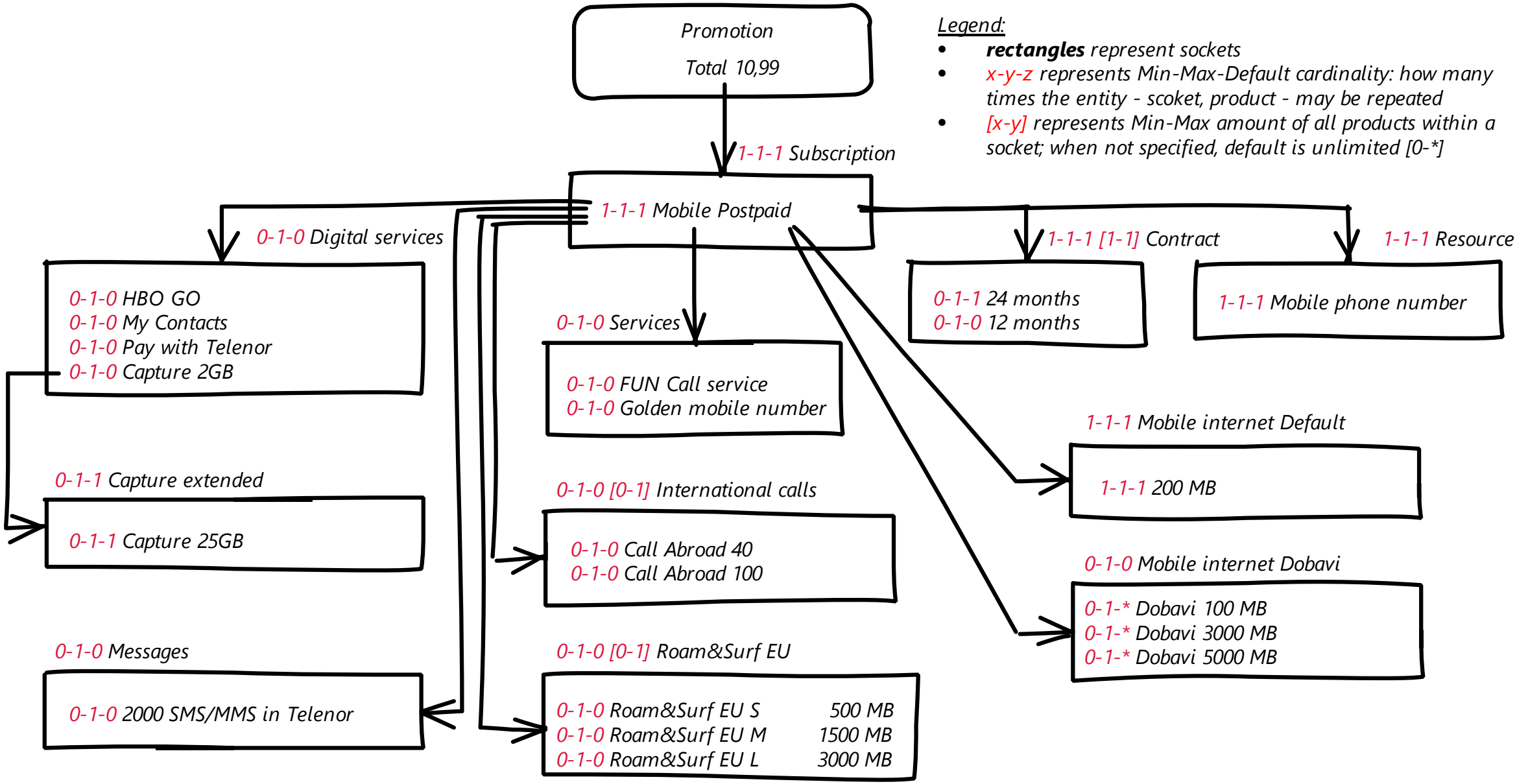
- “All-units” model supported using Rules
  - the price of each unit is equal to the unit price for the cheapest volume tier reached

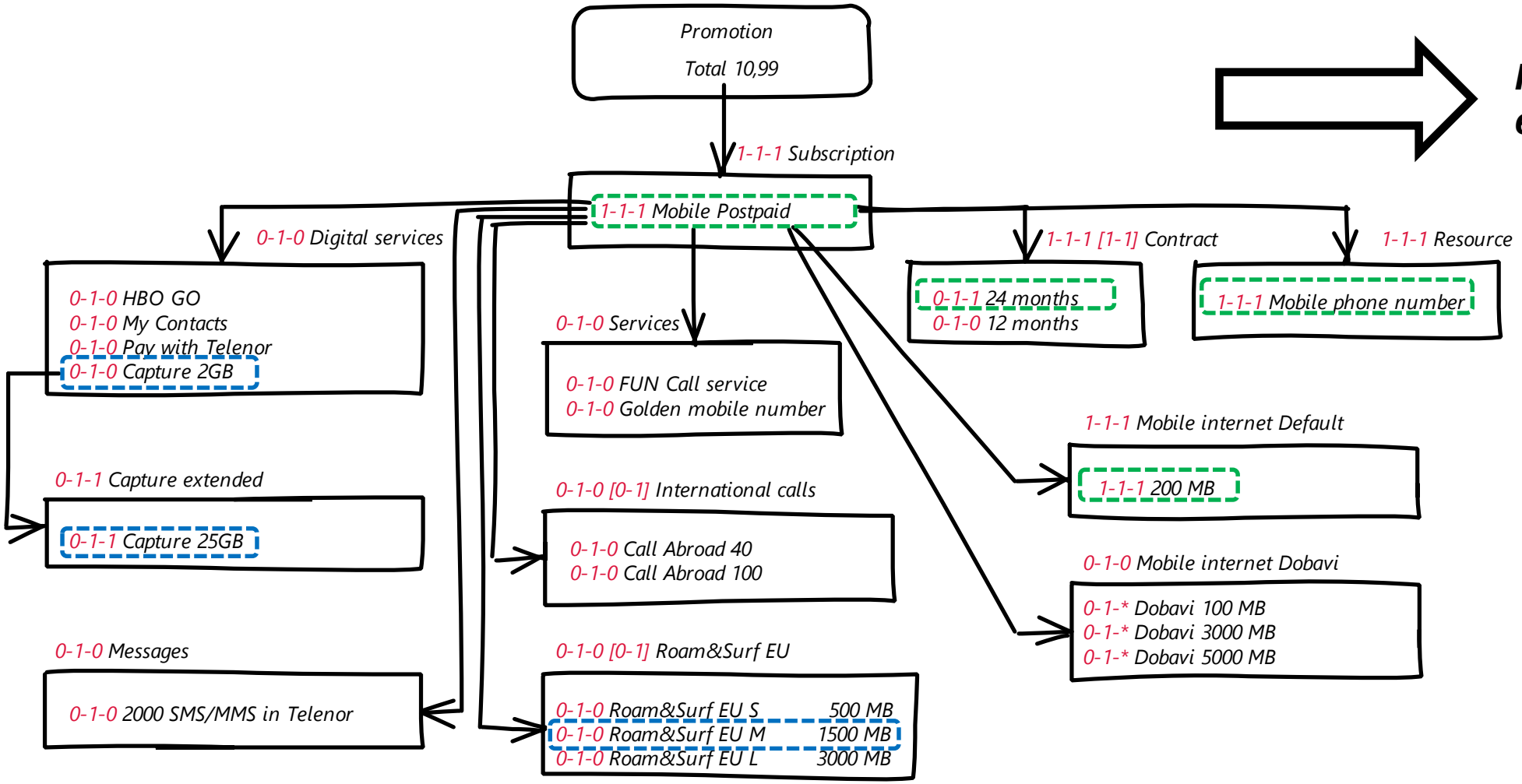
Quantity	Unit Price	Total Price
1	100.0	100.0
2	99.0	198.0
5	95.0	475.0
>5	90.0	540.0 (for amount of 6)

Product Sockets

- Way to configure **Product tree structure**
- Powerful way to solve questions:
  - What are **allowed child products** of given product?
    - E.g. What VAS is allowed under the tariff? What commitments can be configured? What discount may be given?
  - **How many instances** of given product are allowed under another product?
    - E.g. Is it possible to add additional data package?
  - What are the **parent products** allowing to use given product?
    - E.g. What tariffs provide requested TV program?

Sockets configuration example for TnBG





Resulting configuration example

Mobile Postpaid 10,99

- Contract 24 months
- Mobile phone number
- 200 MB
- Roam&Surf EU M 1500 MB
- Capture 2GB
  - Capture 25 GB

Legend:

- Green = automatically added with the chosen promotion
- Blue = manually added based on sockets configuration offer

Product Groups

- Group is a logical combination of Products
  - Group may contain any amount of (optionally ordered) Products
  - Group has type, unique code, priority, name, description
- Example use-cases:
  - Product **variants** (all products within the same group are variants of each other)
  - Product **aggregation** for reporting purposes
  - **Presentation** purposes (distribute all offered VAS to different tiles on GUI and sort them within these tiles)

Product Groups – example usage

Groups configuration

Group **HW** (priority 1): P_7, P_{12}, P_9, P_{10}, ...

Group **Roaming** (priority 2): P_8, P_1, ...

Group **Data** (priority 3): P_3, P_6, P_4, P_5, ...

Product Groups – example usage

Groups configuration	Products offer
Group **HW** (priority 1): P_7, P_{12}, P_9, P_{10}, ...	P_1, P_2, P_3, P_4, P_5,
Group **Roaming** (priority 2): P_8, P_1, ...	P_6, P_7, P_8, P_9, P_{10}
Group **Data** (priority 3): P_3, P_6, P_4, P_5, ...	

Product Groups – example usage

Groups configuration

Group **HW** (priority 1): P_7, P_{12}, P_9, P_{10}, ...

Group **Roaming** (priority 2): P_8, P_1, ...

Group **Data** (priority 3): P_3, P_6, P_4, P_5, ...

Products offer

P_1, P_2, P_3, P_4, P_5,

P_6, P_7, P_8, P_9, P_{10}

HW

P_7

P_9

P_{10}

Products offer presented to the customer

Product Groups – example usage

Groups configuration

Group **HW** (priority 1): P_7, P_{12}, P_9, P_{10}, ...

Group **Roaming** (priority 2): P_8, P_1, ...

Group **Data** (priority 3): P_3, P_6, P_4, P_5, ...

Products offer

P_1, P_2, P_3, P_4, P_5,

P_6, P_7, P_8, P_9, P_{10}

HW

P_7

P_9

P_{10}

Roaming

P_8

P_1

Products offer presented to customer

Product Groups – example usage

Groups configuration

Group **HW** (priority 1): P_7, P_{12}, P_9, P_{10}, ...

Group **Roaming** (priority 2): P_8, P_1, ...

Group **Data** (priority 3): P_3, P_6, P_4, P_5, ...

Products offer

P_1, P_2, P_3, P_4, P_5,

P_6, P_7, P_8, P_9, P_{10}

HW
P_7
P_9
P_{10}

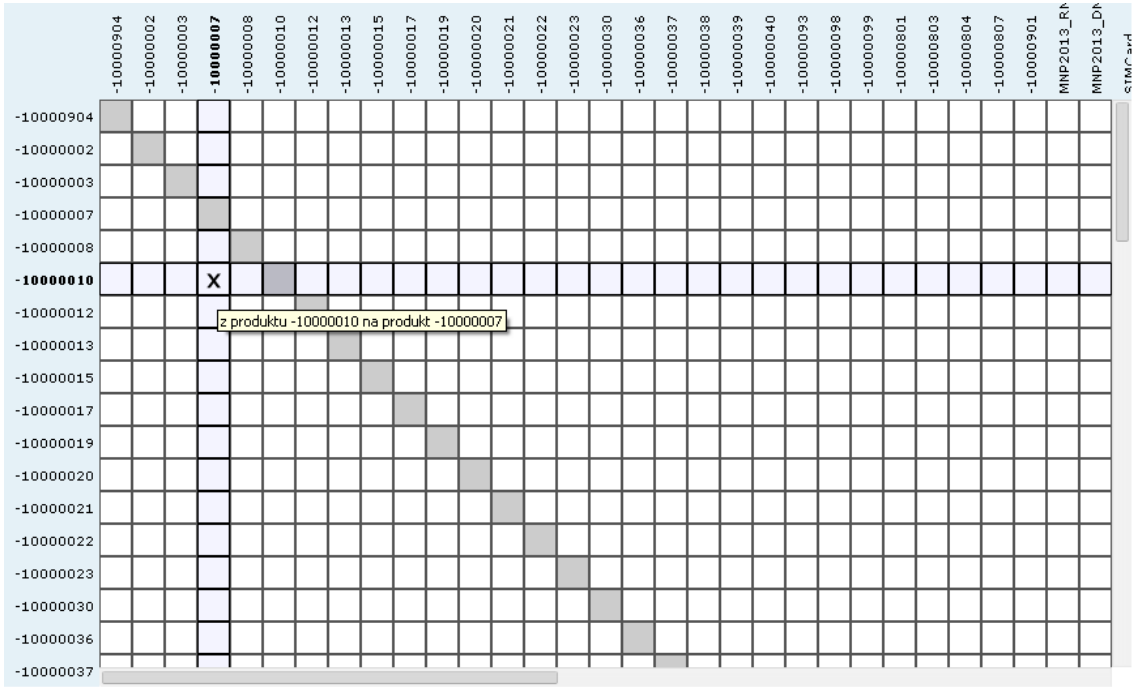
Roaming
P_8
P_1

Data
P_3
P_6
P_4
P_5

Products offer presented to customer

Relationships

- Supports "matrix-based" product-to-product relationships
- Relationship has
  - **type** (e.g. compatibility, allowed migrations, bundle components, or recommended products for cross-sale/up-sale)
  - **source** and **target** (e.g. Product, Product group, Product technical category or business category)
  - **Cardinality** (min-max)
  - **Meaning** (POSITIVE, NEGATIVE)
  - **Scope** (e.g. ORDER, PARENT, PARTY)
  - Time based **validity**



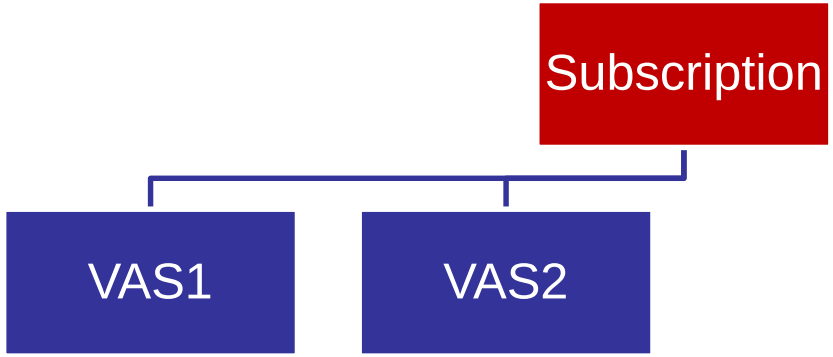
Relationships – Compatibility Example

- COMPATIBILITY relationships examples:
  - product with code X is compatible with 0-2 products of technical category Y
    - in other words, when there is product X, it is allowed to have no more than 2 products of tech. category Y
  - product of business category X is incompatible (*meaning=NEGATIVE*) with any product within product group Y
  - product with code X is compatible with 1-1 product with code Y under the same Party (*scope=PARTY*)
    - in other words, when there is product X, it requires exactly one product Y under the same Party (Party = Customer, Customer Account, Tariff Space, …)

Triggers & Stage Handlers

- Configured as **Product events** with associated handlers
- Examples of events: before/after add/remove Product from shopping cart
- The event can be handled in several ways:
  - **Add** another product
    - In addition may establish a relation between products (e.g. sibling relationship or product link or parent-child relationship in any direction)
  - **Remove** another product
  - Check there is another **required** product
  - **Forbid** the operation

Product events example – initial configuration



Client is going to:

1. ADD **VAS3**
2. REMOVE **VAS1**

Example Product events configuration:

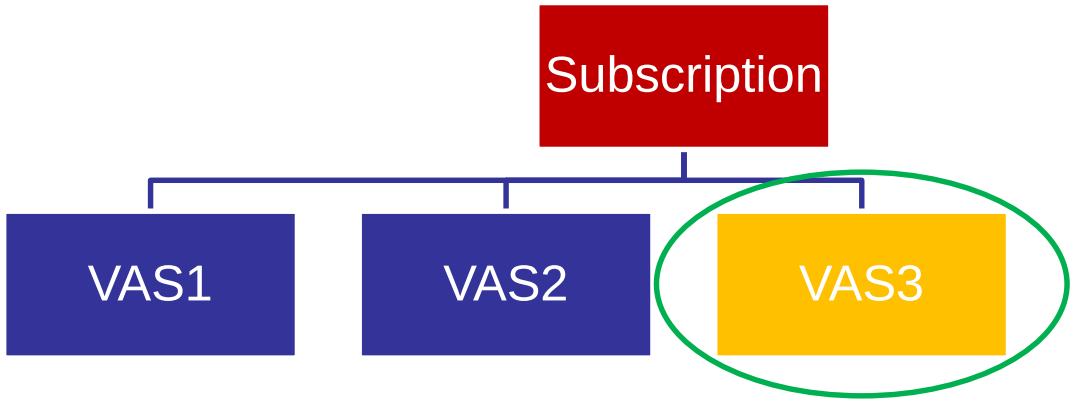
RULE I:

- event type: **ADD VAS3**
- handler: **ADD sibling VAS4**

RULE II:

- event type: **REMOVE VAS1**
- handler: **REMOVE sibling VAS2**

Product events example



Client add VAS3

⇨

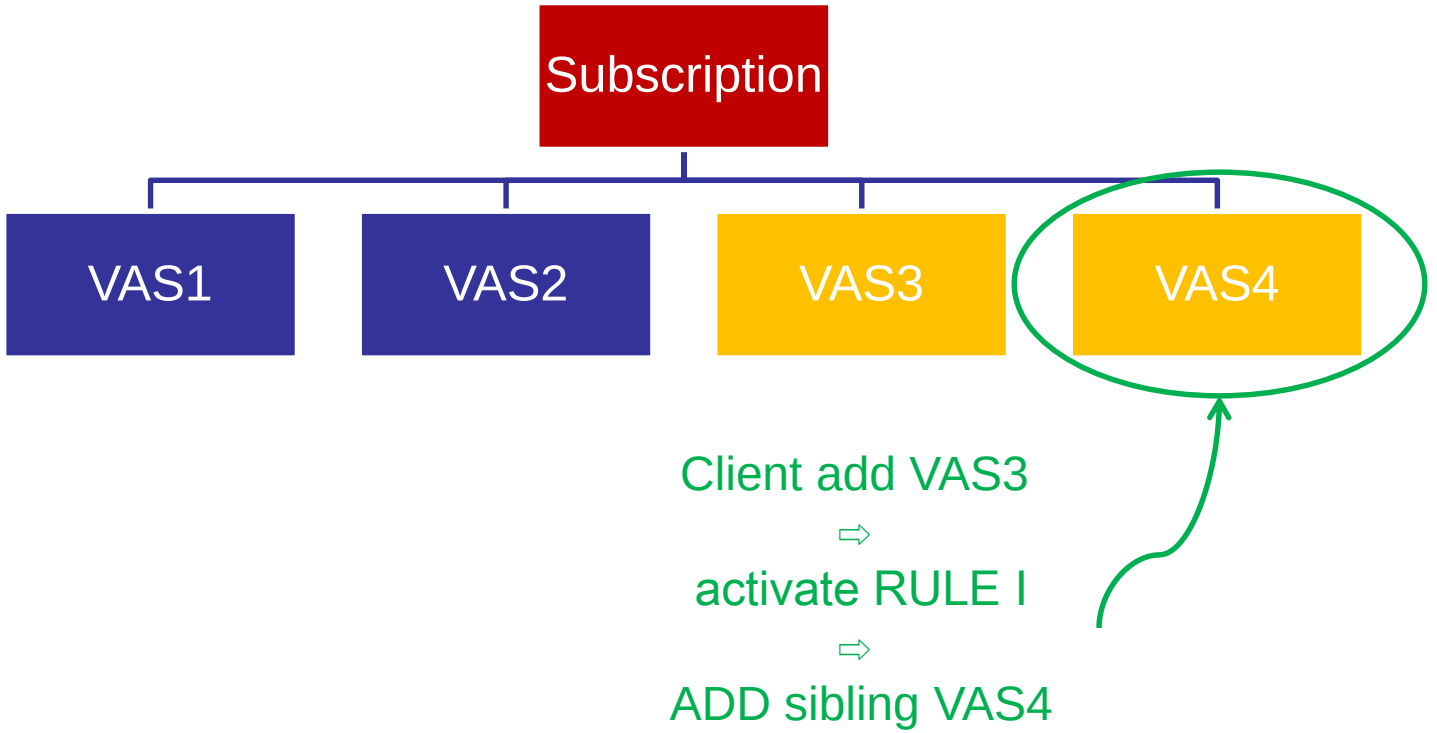
activate RULE I

Example Product events configuration:

RULE I:
- event type: **ADD VAS3**
- handler: **ADD sibling VAS4**

RULE II:
- event type: **REMOVE VAS1**
- handler: **REMOVE sibling VAS2**

Product events example

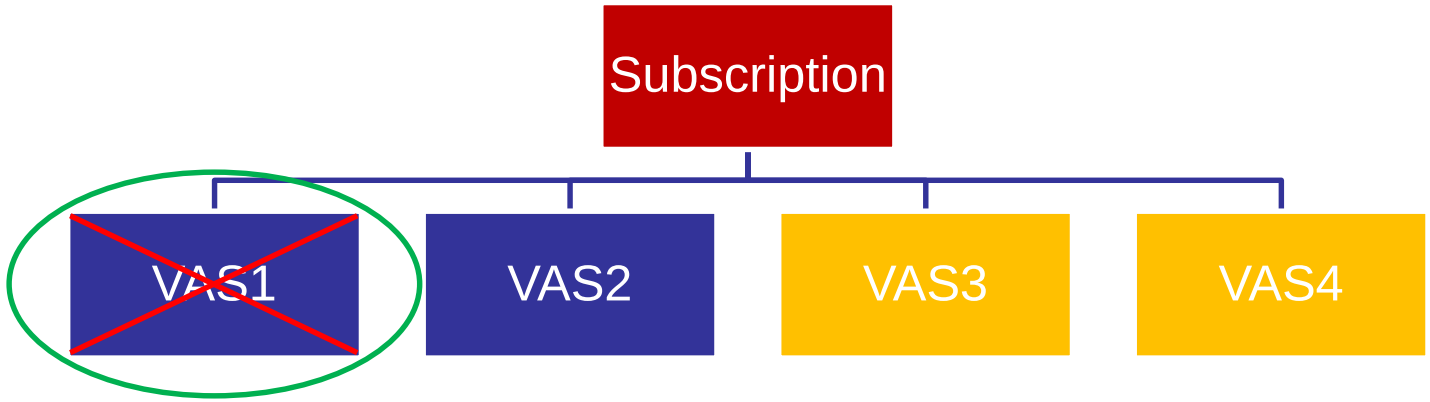


Example Product events configuration:

RULE I:
- event type: **ADD VAS3**
- handler: **ADD sibling VAS4**

RULE II:
- event type: **REMOVE VAS1**
- handler: **REMOVE sibling VAS2**

Product events example



Client remove VAS1

⇨

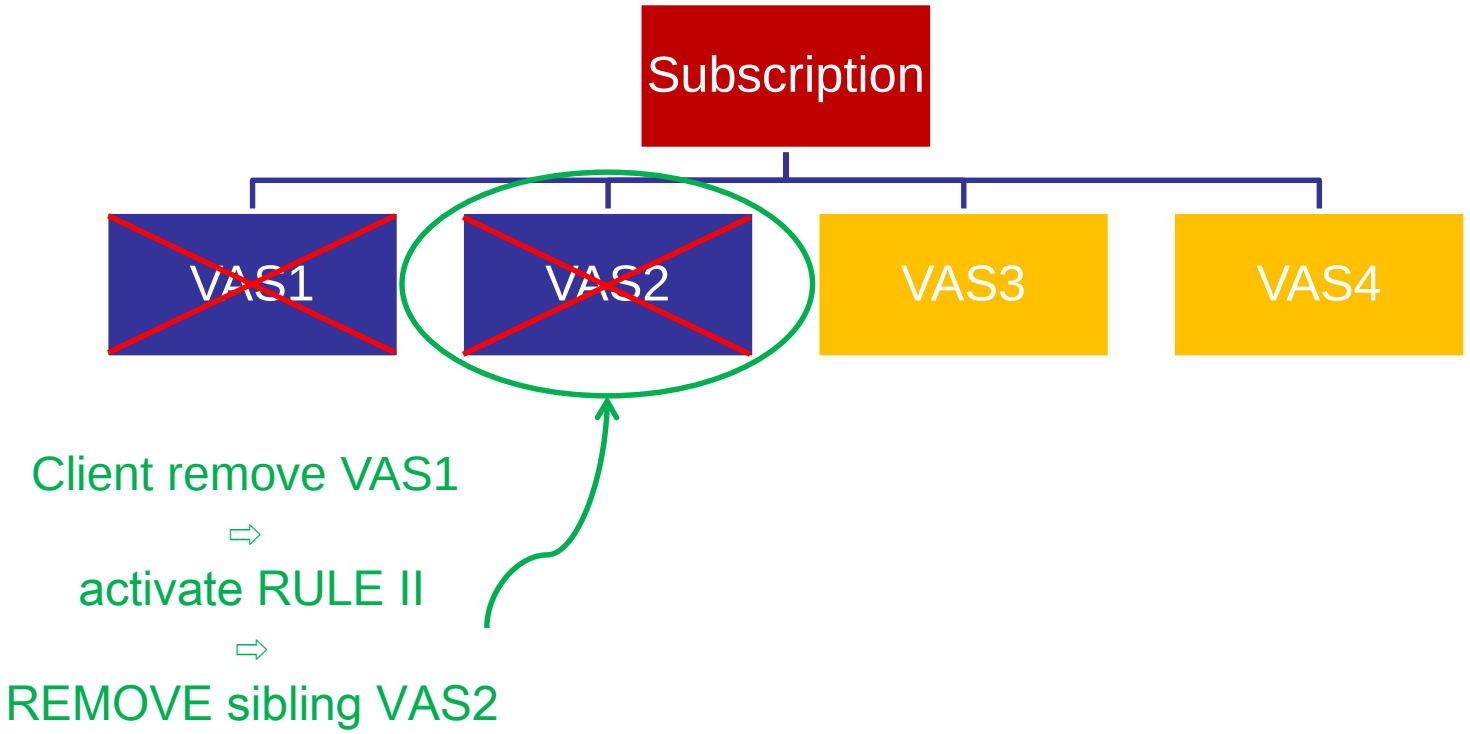
activate RULE II

Example Product events configuration:

RULE I:
- event type: **ADD VAS3**
- handler: **ADD sibling VAS4**

RULE II:
- event type: **REMOVE VAS1**
- handler: **REMOVE sibling VAS2**

Product events example

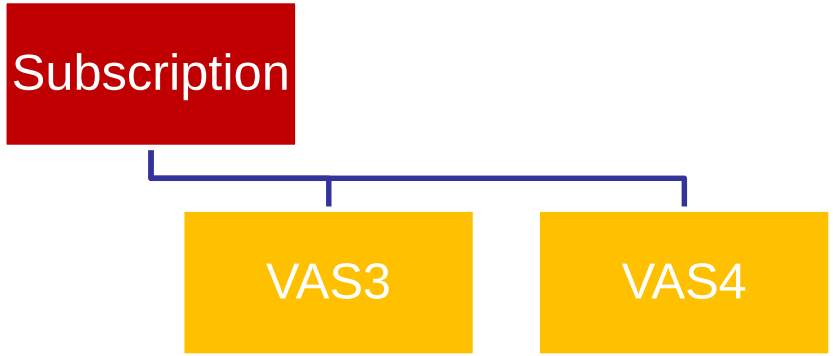


Example Product events configuration:

RULE I:
- event type: **ADD VAS3**
- handler: **ADD sibling VAS4**

RULE II:
- event type: **REMOVE VAS1**
- handler: **REMOVE sibling VAS2**

Product events example – resulting configuration

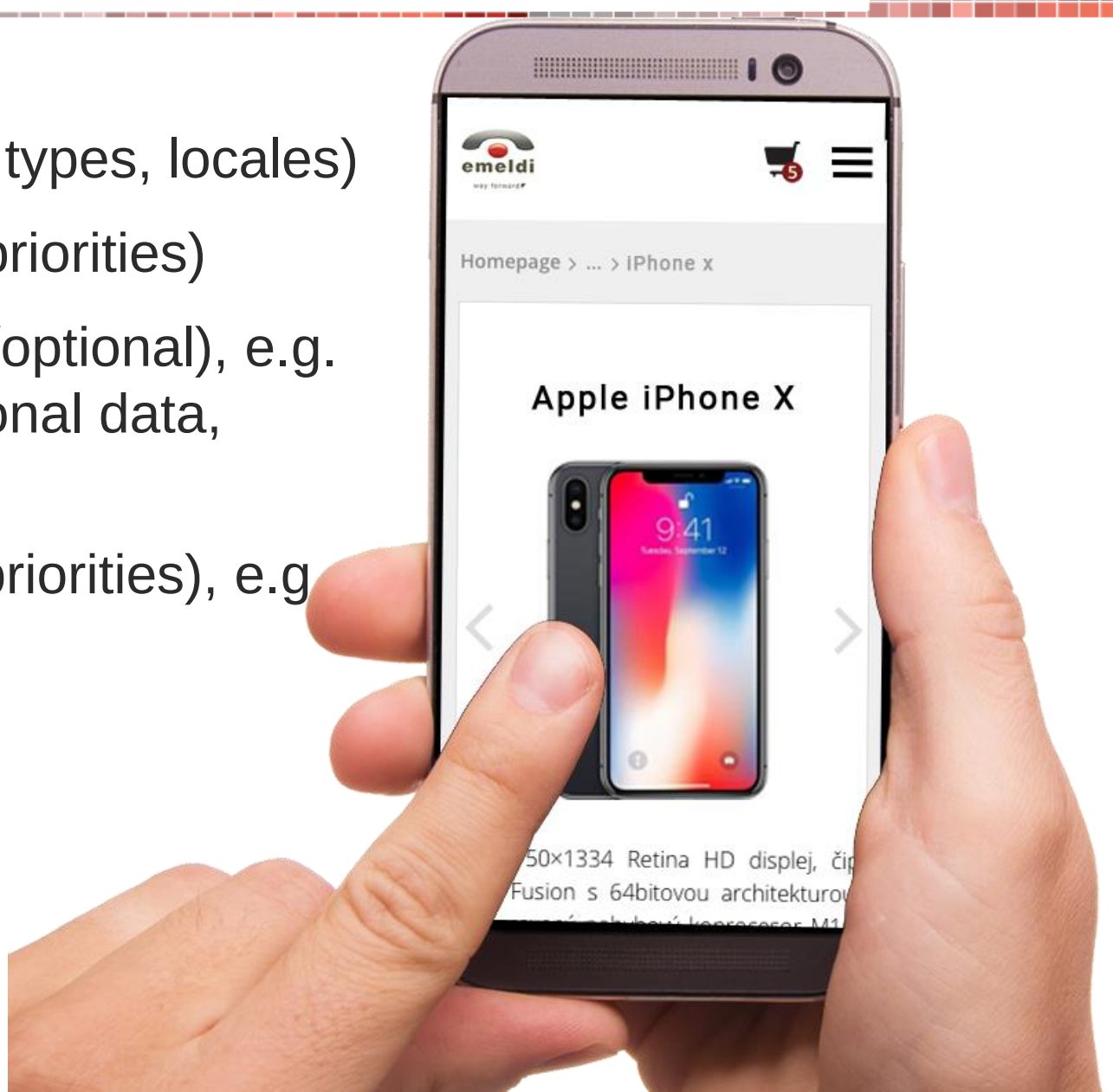


Scenario recapitulation:

- Client add **VAS3**
  - **VAS4** added by configuration
- Client remove **VAS1**
  - **VAS2** removed by configuration

Other Features

- Product **Localized Texts** (various types, locales)
- Product **Pictures** (various types, priorities)
- Product **Agreements** (mandatory/optional), e.g. consent to the processing of personal data, consent to business terms
- Product **Stickers** (various types, priorities), e.g TOP, NEW, ...
- Product **Review**



Technical Documentation

- Integral part of source code
  - Always accurate and up-to-date with released version (part of review and merge process)
  - Transformed to various formats (PDF, HTML, DOC, …)

```
### Specific operations provided in GET requests

1. Basic object retrieval

    Example : GET http://www.site.com/product/45564

This is the simplest form requesting data with id=45564, so requesting specific set of data.
Typical result of such an operation is single JSON object, containg specified element.

| Result        | HTTP Response code | Returned data                          | Comment                      |
|:--------------|:-------------------|:---------------------------------------|:-----------------------------|
| Found         | 200                | JSON with data                         |                              |
| Not found     | 404                | None or JSON with further details (TBD) |                              |
| Not modified |                304 | Empty                                  | ETag provided in request header |


The data retrieved can be further restricted by 'fields' parameter, containing list of fields to be retrieved,
as also specified by more general options below.

1. List of objects retrieval options

    Example : GET http://www.site.com/product/filter='{...}'&sort='...'&fields='...'
```

Specific operations provided in GET requests

1. Basic object retrieval

   Example : GET http://www.site.com/product/45564

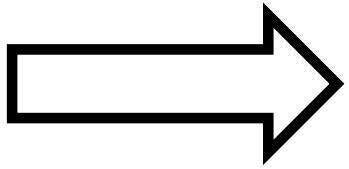
This is the simplest form requesting data with id=45564, so requesting specific set of data. Typical result of such an operation is single JSON object, containg specified element.

Result	HTTP Response code	Returned data	Comment
Found	200	JSON with data	
Not found	404	None or JSON with further details (TBD)	
Not modified	304	Empty	ETag provided in request header

The data retrieved can be further restricted by 'fields' parameter, containing list of fields to be retrieved, as also specified by more general options below.

Documentation – Javadoc

- JAVA API documentation (Javadoc)
  - Integral part of source code
  - Especially usefull for adapters
  - HTML view generated



```
/**
 * Returns collection of Products filtered by given categories. This also includes
 * products with descendant categories.
 * Any product belonging to any given category and its descendants will be returned.
 * @param categories list of categories to be filtered by
 */
@Nonnull
Collection<ProductDto> getProductsByCategories(@Nonnull List<CategoryType> categories);
```

long	getProductsCountByFlexiGrid(com.emeldi.ecc.be.common.dto.dbfiltering.FlexiGridCriteria flexiGridCriteria, ProductVersionType productVersionType) Get products in the same group by productCode
List<Long>	getProductsInTheSameGroup(String productCode, ProductVersionType productVersionType) Get products in the same group by productCode
List<PcProductSocket>	getProductSocketsByPromotionCode(String promotionCode)
List<PcProductSticker>	getProductStickers(String productCode) Get product Stickers by product code
PcSocket	getSocketByCode(String code) Retrieves socket with given code
List<PcSocket>	getSocketsByPromotionCode(String promotionCode) Returns list of PcSockets which are valid for given promotion's code.
List<PcStickerType>	getStickerTypeByCriteria(com.emeldi.ecc.be.common.dto.criteria.BaseCriteria criteria) Get StickerTypes by criteria
List<PcStickerType>	getStickerTypeByFlexiGrid(StickersFlexiGridCriteria flexiGridCriteria) Finds stickers by flexiGridCriteria.
PcStickerType	getStickerTypeById(long stickerId) Get StickerType by id
long	getStickerTypeCountByCriteria(com.emeldi.ecc.be.common.dto.criteria.BaseCriteria criteria) Get StickerTypes count by criteria
long	getStickerTypeCountByFlexiGrid(StickersFlexiGridCriteria flexiGridCriteria)
void	increasePcPopularity(long productMasterId, int points) Creates a stores the PcPopularity object based on values taken from entry ProductPopularity
void	insertOrUpdatePcBoughtTogether(PcProductMaster pcWithProductMaster, PcProductMaster pcAlsoProductMaster) Method for insert or update (merge) PcBoughtTogether
boolean	isAllowedMigration(String sourceProductCode, String targetProductCode) Check if product migration is possible
boolean	isMAVM() Check versioning mode based on global property pc.versioning.mode
void	recalculatePopularity(com.emeldi.ecc.be.pc.enums.CategoryType cat, String procedureName) Recalculates popularity attribute for all product-masters within given category (no sub-categories!!).
void	recalculateTopStickers(com.emeldi.ecc.be.pc.enums.CategoryType cat, int maxOrderAge) Remove all TOP stickers from all products version within given category (no sub-categories!!).
void	recalculateUserRating() Recalculates pc_product_master.rating_user for all pc_product_master records.
void	refreshViews() It refreshes all materialized views used by product catalog.
void	rejectProduct(String productCode) Reject product, if product has future version delete it
void	removeObsoleteNewStickers(int days) Refresh StickerType.NEW product stickers.
void	setMultiActive(boolean multiActive)
List<String>	updateInventoryCounts(Map<String,Integer> products, com.emeldi.ecc.be.pc.enums.ServicePartner servicePartner) Update inventory counts for products belonging to servicePartner

Method Detail

getProductByProductId

```
PcProductVersion getProductByProductId(Long productId,
                                       ProductVersionType productVersionType)
                                throws com.emeldi.ecc.be.common.dao.DaoException
```

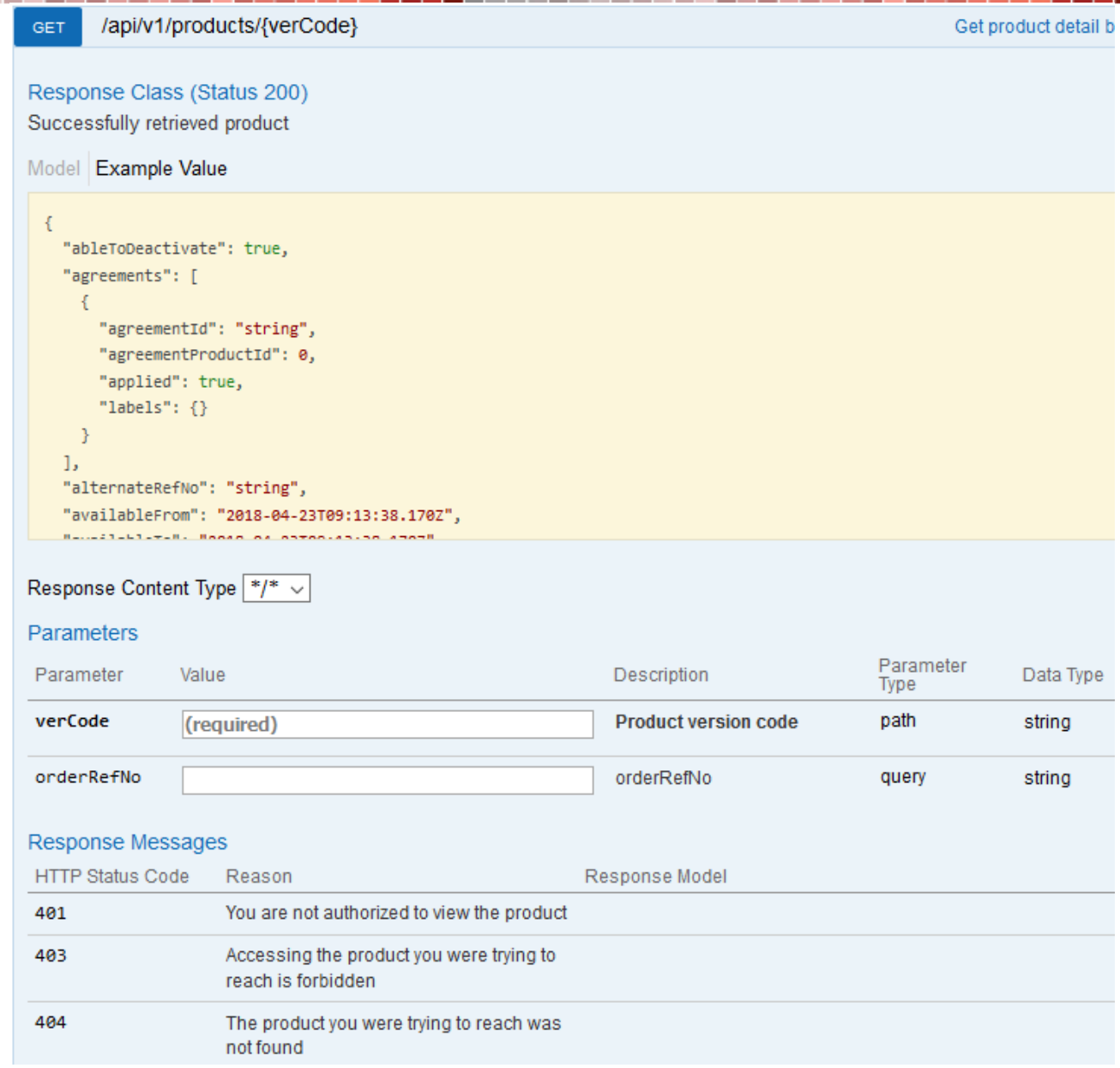
Find product version by productId and version type

Throws:

com.emeldi.ecc.be.common.dao.DaoException

Documentation – REST API

- REST API Swagger generated doc
- Available Online



GET /api/v1/products/{verCode} Get product detail b

Response Class (Status 200)

Successfully retrieved product

Model | Example Value

```
{
  "ableToDeactivate": true,
  "agreements": [
    {
      "agreementId": "string",
      "agreementProductId": 0,
      "applied": true,
      "labels": {}
    }
  ],
  "alternateRefNo": "string",
  "availableFrom": "2018-04-23T09:13:38.170Z",
```

Response Content Type */*

Parameters

Parameter	Value	Description	Parameter Type	Data Type
verCode	(required)	Product version code	path	string
orderRefNo		orderRefNo	query	string

Response Messages

HTTP Status Code	Reason	Response Model
401	You are not authorized to view the product	
403	Accessing the product you were trying to reach is forbidden	
404	The product you were trying to reach was not found	