osCommerce

CONCEPTUAL SCHEMA

Albert Tort Pugibet
Universitat Politècnica de Catalunya (UPC)

[ÍNDEX]

SCHEMA
PRESENTATION

1

1 SCHEMA PRESENTATION

1.1 THE *osCommerce* SYSTEM

E-commerce allows people exchanging goods and services with no barriers of time or distance. Surfing the net, you can easily find online shops where you can buy almost anything you need, at any time and without leaving home.

Electronic commerce changes the way of business operate. Nowadays, it is possible to start a 24 hours opened online store with lower costs than traditional establishments. The idea of small local shops has no sense for e-commerce: online stores are international accessible since they start operating.

osCommerce is an e-commerce solution available as free software under the GNU General Public License. *osCommerce* project was started in March 2000 in Germany and since then, it has become the base of thousands of online stores around the world.

1.2 THE *osCommerce* CONCEPTUAL SCHEMA

The main purpose of this work is giving a conceptual schema of the *osCommerce* system as a result of a reverse engineering process. This conceptual schema is specified using standard UML and OCL with some extensions to improve expressivity.

The osCommerce conceptual schema is shown in chapters 2 and 3. It has been made up using the public documentation of the system (sometimes limited and imprecise), experimenting as a user with the current version and analysing the database schema.

We publish this work and we make it accessible for the community. It can be a complementary documentation of the system and a detailed specification of its knowledge and behaviour. It can be useful for engineers who are going to develop online stores based on *osCommerce* and for everyone interested on it.

STRUCTURAL
SCHEMA

2 STRUCTURAL SCHEMA

2.1 INTRODUCTION

In this chapter we develop the structural schema of the *osCommerce* information system.

The main purpose of the *osCommerce* structural schema is providing a description of the conceptualization of the *osCommerce* domain.

The structural schema is too large to be presented without partitioning it. For that reason, this chapter begins with an UML general view diagram with the most important conceptual entity types and their relationship types.

Afterwards, the whole schema is structured in several more detailed diagrams in order to make it more understandable. Each diagram corresponds to a part of the whole detailed schema and groups some related concepts which can be seen as a set of knowledge about the information system.

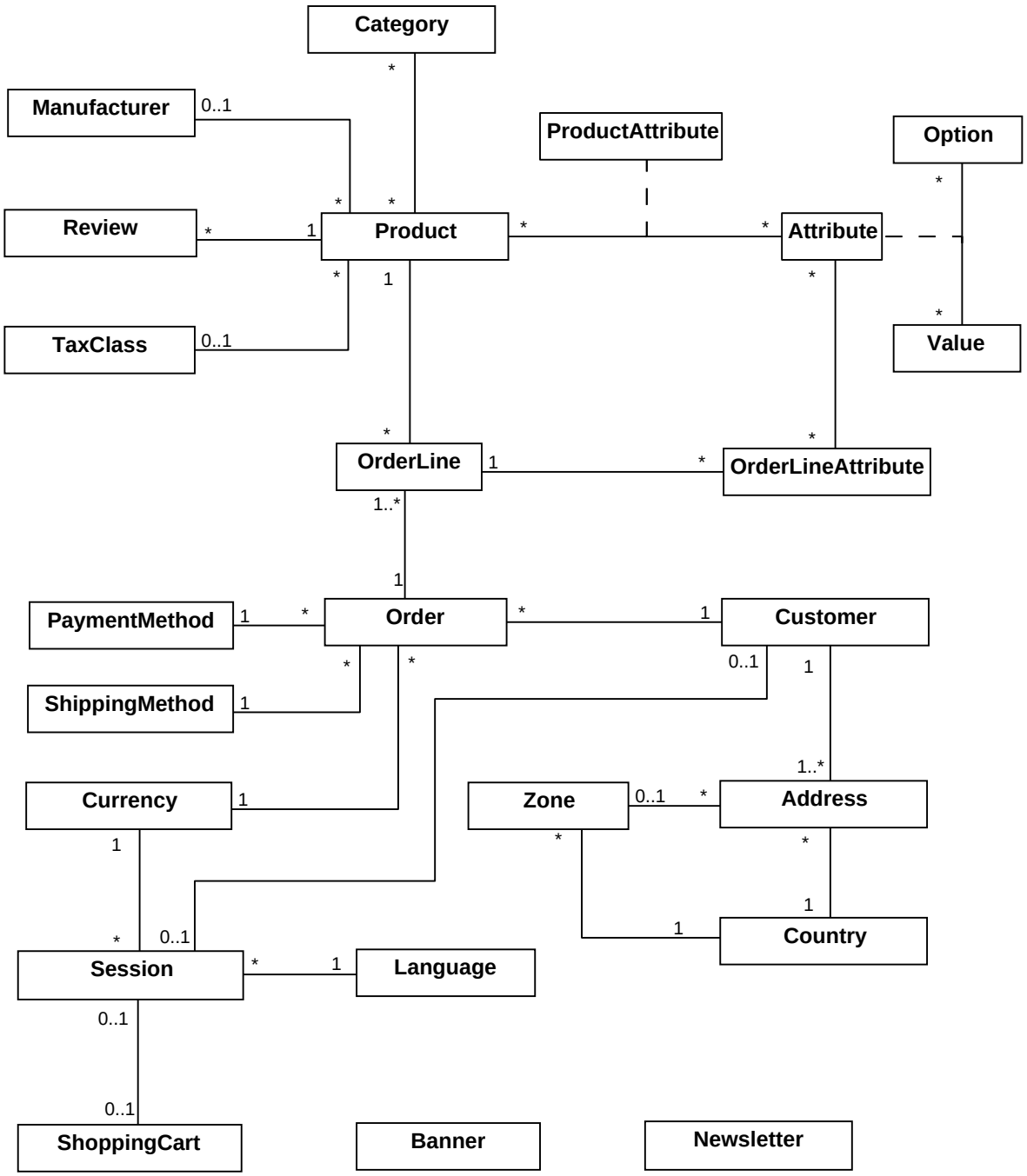
Entities which participate in relationships of a structural schema fragment but are full specified in other conceptual grouping diagrams are drawn without showing their attributes. Moreover, these “external” entities have a reference which links them to the page where its complete specification can be found.

Each fragment of the whole structural schema diagram is represented in UML and it is introduced by a brief textual overview. Derived types and integrity constraints are specified in OCL. The structural schema specification uses standard UML with some extensions (specification of derived types and constraints by OCL operations, constant and permanent stereotypes…) explained in *Conceptual Modeling of Information Systems* [Oli07] written by Antoni Olivé.

In addition, a detailed description about the schema fragment and example instantiations of it, improve comprehension. Some of the example instantiations are inspired from real online shops based on *osCommerce* which can be found in the *osCommerce* website (www.oscommerce.com). Others are made up from real life experiences.

2.2 OVERVIEW DIAGRAM

The next diagram represents a simplified conceptual schema which gives an overview of the main concepts in the *osCommerce* domain. More details about each concept are given in the next section, where the whole schema is fully specified.



2.3 MAIN DOMAIN CONCEPTS

The products in the store are manufactured by **manufacturers**, are grouped into **categories** and belong to a **tax class**. Moreover, customers can write **reviews** of a product.

osCommerce is a multilingual system able to deal with any number of **languages**. Likewise, osCommerce allows working with different tax classes and **currencies**.

Products may have **attributes**. An attribute is an **option/value** pair which is used to offer multiple varieties of a product without needing to create many separate but very similar products. The price of a product is increased or decreased depending on the chosen attributes. The price variation produced by an attribute is indicated, for each product, by **product attribute** entity types.

Customers have one or more **addresses**. Each address is located in a **country**. If the country has **zones** (states or provinces) then the address must be located in one of its zones.

Every use of the online store is conceptually represented by a **session**. Sessions can be anonymous or belong to a customer. Moreover, every session has always a current currency and a current language.

In the context of sessions, users can surfing the online store. **Shopping carts** contain one or more selected items (not shown in the figure) each of which is a quantity of a product with a set of attributes.

When a customer confirms that he wants to buy the contents of his shopping cart the system generates an **order**. An order is made by a customer using a **payment method**. Furthermore, order prices are expressed in a specified **currency** and take into account the shipping costs, according to the chosen **shipping method.**

An order contains one or more **order lines**, each of which is a quantity of a product with a set of attributes.

Finally, osCommerce offers some administration tools like **banners**, used to customize the online advertisements in the store, and **newsletters**, used to send information by email to customers.

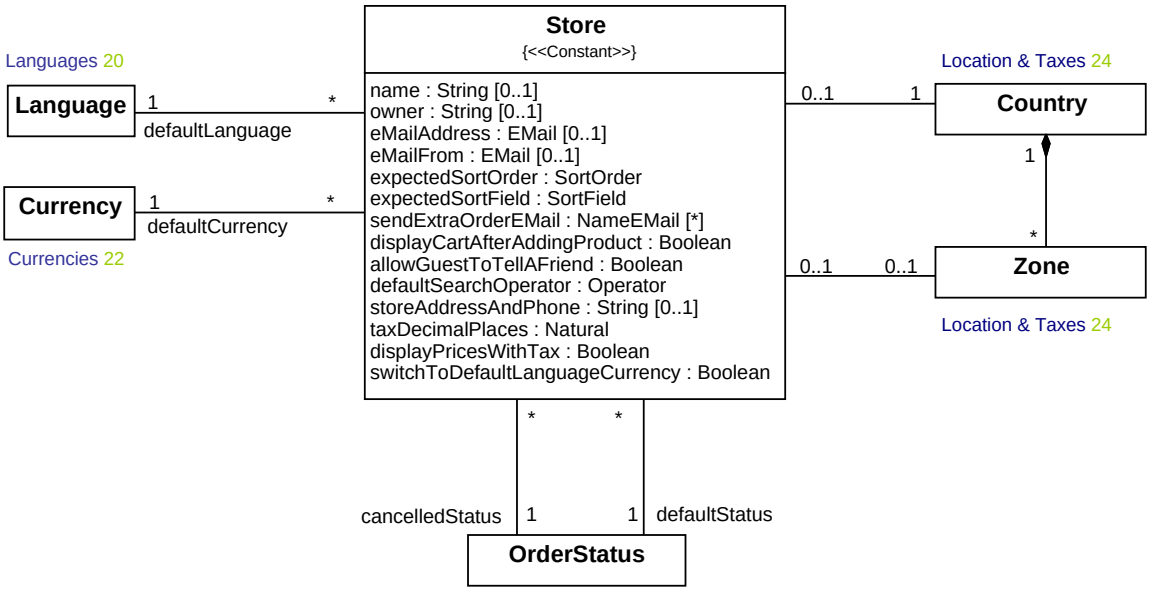
2.4 STORE CONFIGURATION

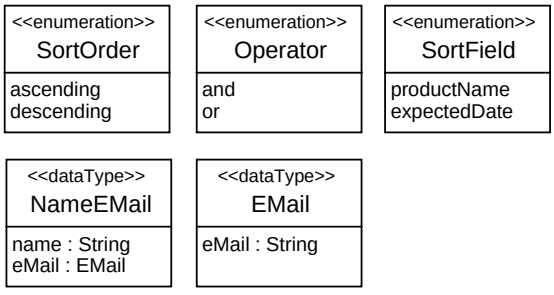
Store Data

Overview

osCommerce keeps general data about the store and some other information which is used to customize the behaviour of the system.

Conceptual Diagram





Constraints

[1] There is only one instance of *Store*

```
context Store::alwaysOneInstance: Boolean
  body : Store.allInstances() -> size() = 1
```

[2] The store's zone is part of the country where the store is located.

```
context Store::zoneIsPartOfCountry: Boolean
  body : self.zone -> notEmpty() implies self.country.zone -> includes (self.zone)
```

Description

There is only one instance of *Store* which is created and initialized on installation. It stores the general data of the store and some other customizable properties:

- **Name:** The store's name.
- **Owner:** The store owner's name.
- **Email address:** The store's email address.
- **Email from:** The email address used to send emails.
- **Country:** The country where the store is located.
- **Zone:** The state, zone or province where the store is located.
- **Expected sort order:** Specifies how products are listed, either in ascending or descending order.
- **Expected sort field:** Specifies which field is used to sort products.
- **Send extra order e-mail:** This is a set of *NameEmail* entities. It stores the email addresses where orders will be received. There can be several email addresses for backups.
- **Display cart after adding a product:** Specifies whether the shopping cart will be shown automatically by the system after adding a product.
- **Allow guest to tell a friend:** Specifies whether users can send an e-mail to a friend with information about the store.

- **Default search operator:** Specifies which operator is used in searches.
- **Store address and phone:** The store owner's name, phone, and other public information that will be shown to customers.
- **Tax decimal places:** Sets how many decimal places are used in taxes.
- **Display prices with tax:** Indicates whether prices are shown with taxes or not.
- **Switch to default language currency:** Specifies whether the system automatically changes the currency when the language is changed.
- **Default language:** Specifies the language used by default.
- **Default currency:** Specifies the currency used by default.
- **Cancelled status:** The *OrderStatus* used to indicate that an order is cancelled.
- **Default status:** The *OrderStatus* assigned when an order is created.

Example

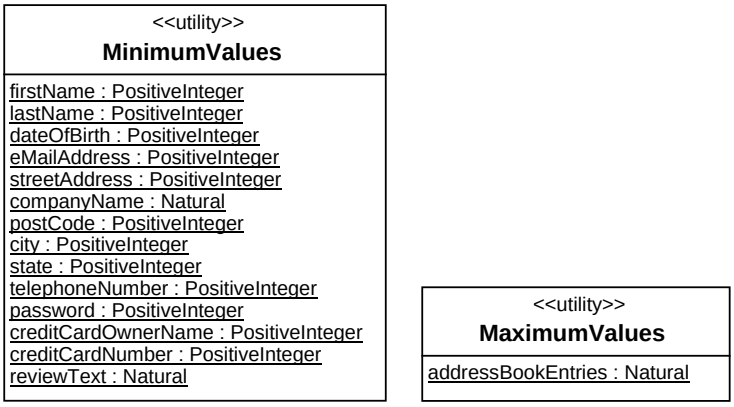
Gaudisc is a classical music *online* shop based on *osCommerce.* This is a possible instantiation of *Store* for this shop:

Minimum and maximum values

Overview

osCommerce allows defining the minimum and maximum length for some *String* attributes.

Structural Schema



Description

MinimumValues sets the minimum length of the following customer attributes:

- **First name**
- **Last Name**
- **Date of birth**
- **Email From**
- **Street address**
- **Company**
- **City and postal code**

- **State**
- **Telephone number**
- **Password**
- **Owner's credit card name**
- **Credit card number**

Moreover, *MinimumValues* specifies the minimum length of:

- **Product reviews text**

Finally, *MaximumValues* specifies the maximum number of:

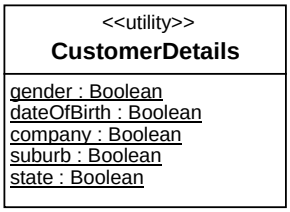
- **Address book entries** permitted for each customer**.**

Customer details configuration

Overview

The system allows specifying whether some customer attributes are shown and required when creating, editing or showing an account.

Structural Schema



Description

CustomersDetails allows configuring whether the following customer attributes are shown or not:

- **Gender**
- **Date of birth**
- **Company name**
- **Suburb**
- **State**

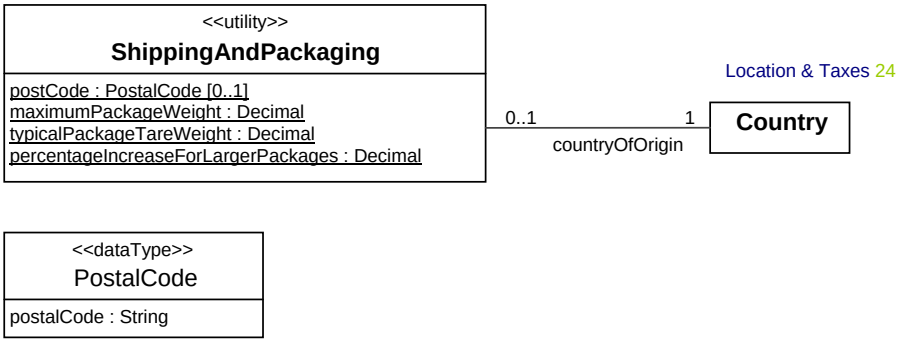
Customer attributes which are not shown, are not required when creating or editing an account, even if they are mandatory customer attributes.

Shipping and Packaging configuration

Overview

The system allows setting up some configuration values used in shipping costs calculation.

Structural Schema



Constraints

[1] The package tare weight must be less than the maximum package weight.

context ShippingAndPackaging::tareIsLessThanMaximumWeight: Boolean
body : self.typicalPackageTareWeight < self.maximumPackageWeight

Description

The **postal code** and the **country of origin** are used for the system to calculate shipping quotes in some shipping methods.

Maximum package weight is the maximum weight permitted for a single package.

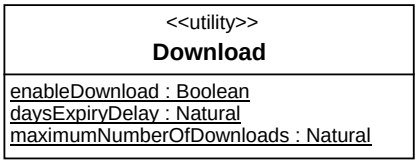
Package tare weight is the typical weight of shipping box and packing material and it is added to the weight of products when computing postage. Larger packages increase their weight as indicated in the **percentage increase for larger packages** attribute instead of using the **typical package tare weight**.

Download configuration

Overview

The system allows customizing the most important general downloadable product properties.

Structural Schema



Description

There is a special type of product *Option* which allows customers downloading it.

The general properties of downloadable products can be customized setting up the following attributes:

- **Enable download:** Determines whether it is possible to download products.
- **Expiry delay:** Specifies the maximum number of days the downloadable file of a product will be available.
- **Maximum number of downloads:** Sets the maximum number of times the customer will be able to download the same product.

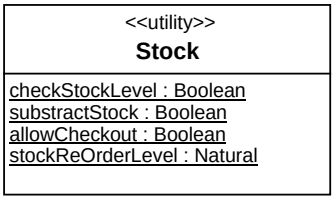
These values are used as default when creating a downloadable product attribute, although it can be redefined then.

Stock configuration

Overview

The system allows configuring some options about the stock administration.

Structural Schema



Description

Checking the stock level can be enabled or disabled by changing the value of **check stock level** attribute.

Moreover, it is possible to indicate whether the system has to decrease the stock when a product is purchased, setting up the attribute **subtract stock**.

The store owner can allow customers checking out products even if there is insufficient stock by activating the *Boolean* attribute **allow checkout**.

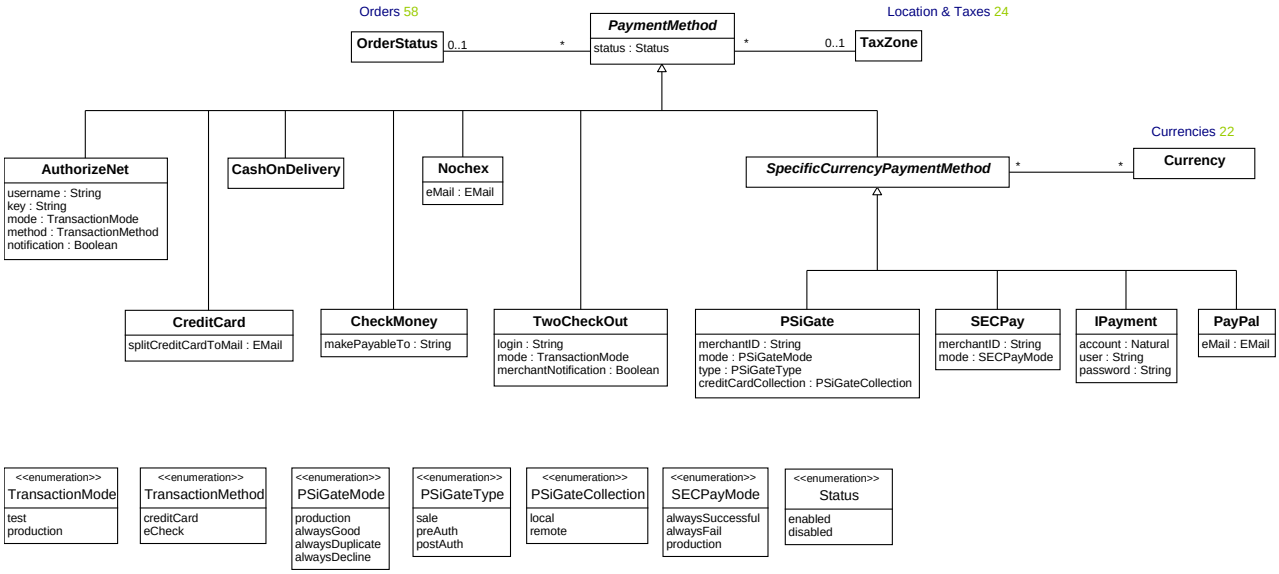
Finally, the attribute **stock reorder level** specifies the minimum inventory that indicates the stock needs to be reordered.

Payment methods

Overview

The system allows operating with different payment methods.

Structural Schema



Constraints

[1] There is at least one enabled payment method

context PaymentMethod::AtLeastOneEnabled: Boolean
body : PaymentMethod.allInstances() -> select (pm | pm.status=Status::enabled) -> size() >= 1

Description

The system allows customers paying through different payment methods, which can be **enabled** or **disabled** by the store administrator.

Some of the payment methods, like *Authorize.net*, *iPayment*, *Nochex*, *PayPal*, *2Checkout*, *PSiGate* or *SECPay,* involve an external company for credit card processing.

There are also a few methods that simply store information for off-line processing.

There are also modules available for *handling cash*, *money order* and *check payments*, which do not involve an external merchant.

Therefore, all the payment methods have specific information about all the necessary data to process the payment.

If the payment method has an **associated *TaxZone***, it is only enabled in zones included in the specified *TaxZone*.

If the payment method specifies an ***OrderStatus***, the status of the orders paid through it, is automatically setup to this status.

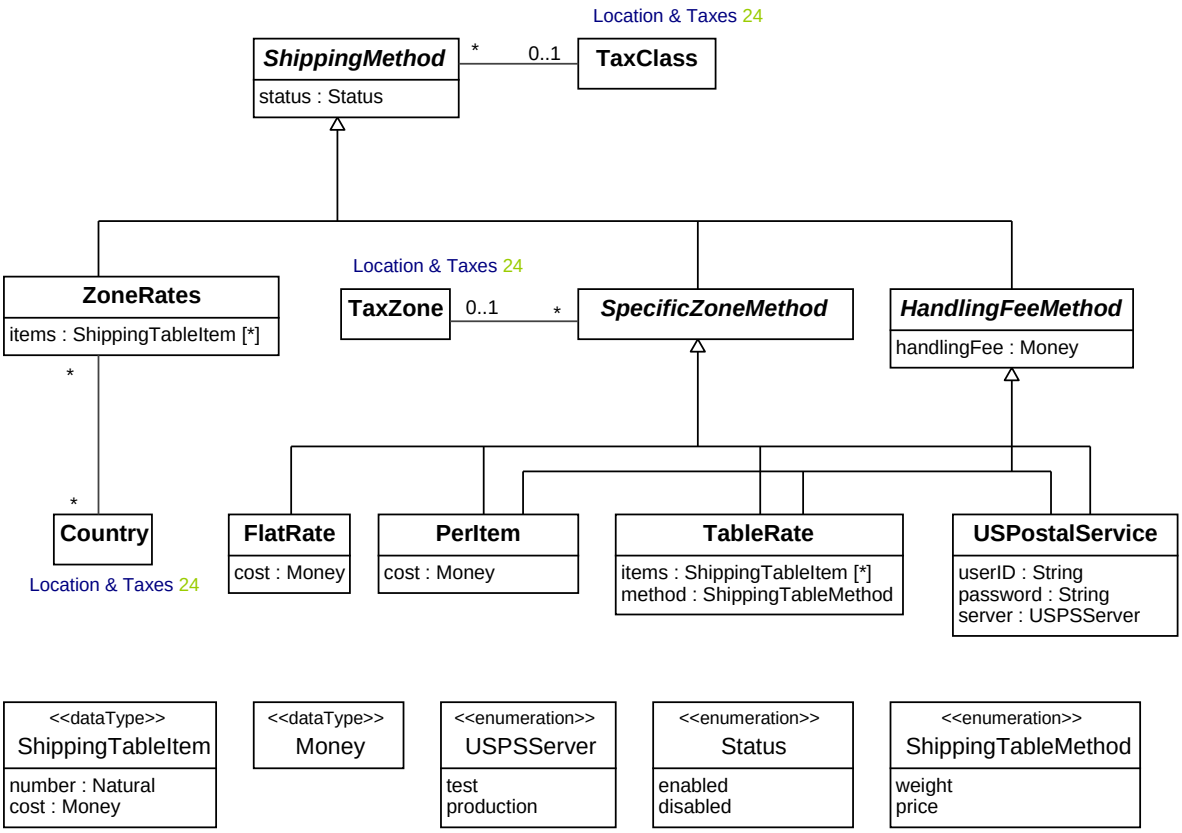
In addition, some payment methods have a set of associated **currencies**. In this case, payment methods are only enabled for operating with these currencies.

Shipping methods

Overview

The system allows operating with different shipping methods.

Structural Schema



Constraints

[1] There is at least one enabled shipping method.

context ShippingMethod::AtLeastOneEnabled: Boolean
body : ShippingMethod.allInstances() -> select (sm | sm.enabled) -> size() >= 1

Description

osCommerce allows customizing shipping methods which are available at checkout time. During the checkout process, the chosen method is used to calculate the final shipping and packaging costs for the order.

Depending on the selected method, the price can be affected by how many products have been ordered, how much they weight or other criteria:

- **Flat rate:** A single price is used on all orders, regardless of how many items are bought, how much everything weights, etc…
- **Per Item:** A single price is multiplied by the number of items in the customer's basket. A flat handling cost may also be added.
- **Table Rate:** Table rate charging sets the price for shipping based on the total weight or the total cost of the products ordered. The weight or price is looked up in a table to find the matching range, and then that price is applied. This is similar to Flat Rate charging, but with different levels.
- **United Parcel Service (UPS):** The UPS shipping method interacts with the UPS website to calculate the total price.
- **United States Postal Service (USPS):** The USPS shipping method interacts with the USPS website to calculate the total price.
- **Zone Rates:** Zone rates shipping method is similar to Table Rate method. The total weight of the customer's order is looked up in a table, and that price is used as the shipping cost.

If the shipping method has an **associated *TaxClass***, it will be applied in the shipping cost.

Specific zone methods can have an **associated *TaxZone***. In this case, the payment method is only applicable in zones included in the specified *TaxZone*.

Similarly, the **associated countries** for *Zone Rates* method represents the countries where it is applicable.

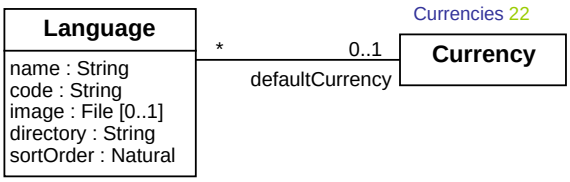
Finally, like payment methods, shipping methods can be **enabled** or **disabled** as desired.

Languages

Overview

osCommerce is a multilingual system able to deal with any number of languages.

Structural Schema



Constraints

[1] A language is identified by its name and by its code

context Language::codeAndNameAreUnique: Boolean
body : Language.allInstances() -> isUnique(name) **and** Language.allInstances() -> isUnique(code)

Description

Languages can be added or deleted as desired and are identified by a **name** and a **code**.

The **directory** indicates to the system the name of the directory which contains its configuration files.

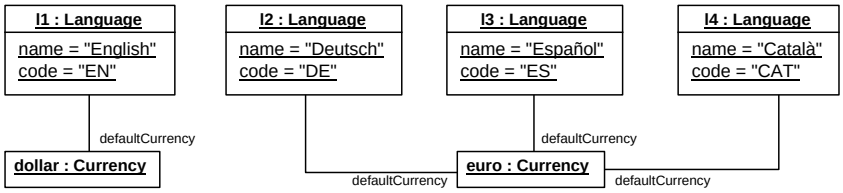
Languages are listed taking into account the **sort order** number.

A language can have an **image**, which is used to identify it visually.

Moreover, languages can have a **default currency**.

If the *Store* attribute *SwitchToDefaultLangaugeCurrency* is true, when a language becomes the current language, its default currency becomes the current currency.

Examples

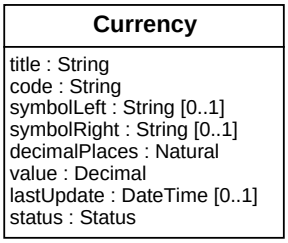

l1 : Language
name = "English"
code = "EN"
l2 : Language
name = "Deutsch"
code = "DE"
l3 : Language
name = "Español"
code = "ES"
l4 : Language
name = "Català"
code = "CAT"
defaultCurrency
dollar : Currency
defaultCurrency
defaultCurrency
euro : Currency
defaultCurrency

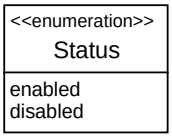
Currencies

Overview

osCommerce allows working with different currencies.

Structural Schema





Constraints

[1] A currency is identified by its title and by its code.

context Currency::codeAndTitleAreUnique: Boolean
body :
Currency.allInstances() -> isUnique(title) **and**
Currency.allInstances() -> isUnique(code)

Description

Currencies can be added or deleted as desired by the store owner and are identified by a **title** and a **code**. The product's price is multiplied by the attribute **value** in order to allowing conversion between currencies.

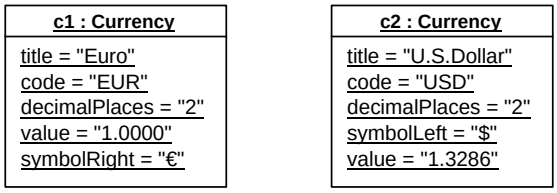
For example, if the value of Euros is 1.0000 and the *value* of Dollars is 1.3286, we can assume that product prices are saved by the system in Euros.

If the current currency is changed to Dollars, all the prices will be multiplied by 1.3286 in order to be expressed in Dollars.

Finally, the **status** of a currency indicates if the online store can currently operate with it.

Examples

osCommerce, by default, allows dealing with two different currencies: Euros and U.S. Dollars. This is the instantiation of these currencies:

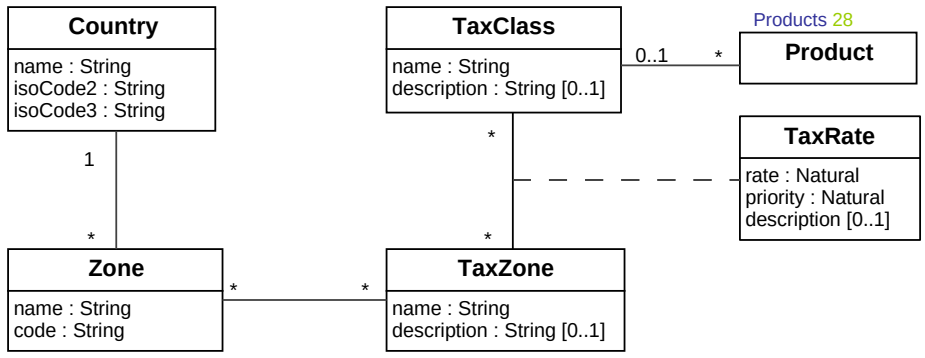


Location & Taxes

Overview

In order to supply a flexible use of taxes, product prices are stored tax free. This allows calculating the final price of products depending on the customer's location and the tax class applied to it.

Structural Schema



Constraints

[1] A *Country* is identified either by its name or its ISO codes.

context Country::nameAndCodesAreUnique: Boolean
body :
Country.allInstances() -> isUnique (name) **and**
Country.allInstances() -> isUnique (isoCode2) **and**
Country.allInstances() -> isUnique (isoCode3)

[2] A *Zone* is identified either by its name and country or its code and country.

context Zone::nameAndCountryAndCodeAndCountryAreUnique: Boolean
body :
Zone.allInstances() -> isUnique (Tuple{n:name, c:country}) **and**
Zone.allInstances() -> isUnique (Tuple{n:code, c:country})

[3] A *TaxZone* is identified by its name.

```
context TaxZone::nameIsUnique: Boolean
  body : TaxZone.allInstances() -> isUnique (name)
```

[4] A *TaxClass* is identified by its name

```
context TaxClass::nameIsUnique: Boolean
  body : TaxClass.allInstances() -> isUnique (name)
```

Description

The final price of products is calculated depending on the customer's location and the tax class applied to a product. In order to be a customizable solution, osCommerce allows setting up different types of taxes and different tax zones where they can be applied.

Tax Classes identify a particular type of tax.

Tax Zones are required to calculate the appropriate tax rate value based on where the purchase is coming from and group *Zones* with the same tax regulation.

Tax Rates specify the tax percentage that is used in a Tax Zone for a *TaxClass*.

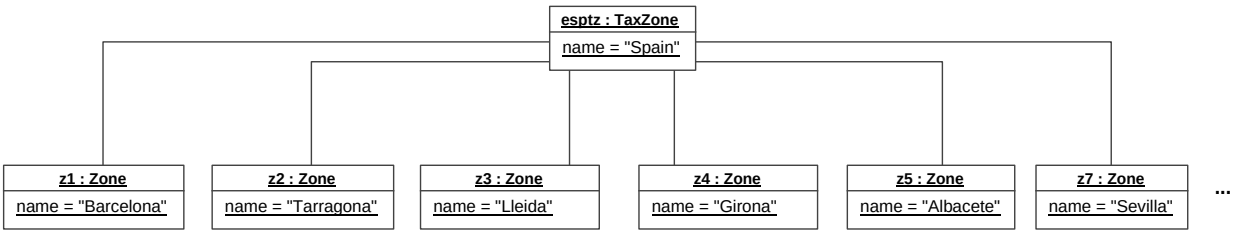
Priorities play an important role in a Tax Class as they state how multiple tax rates in the same class are treated; either adding each rate together when the priorities are the same, or compounding the rates together in the defining priority order.

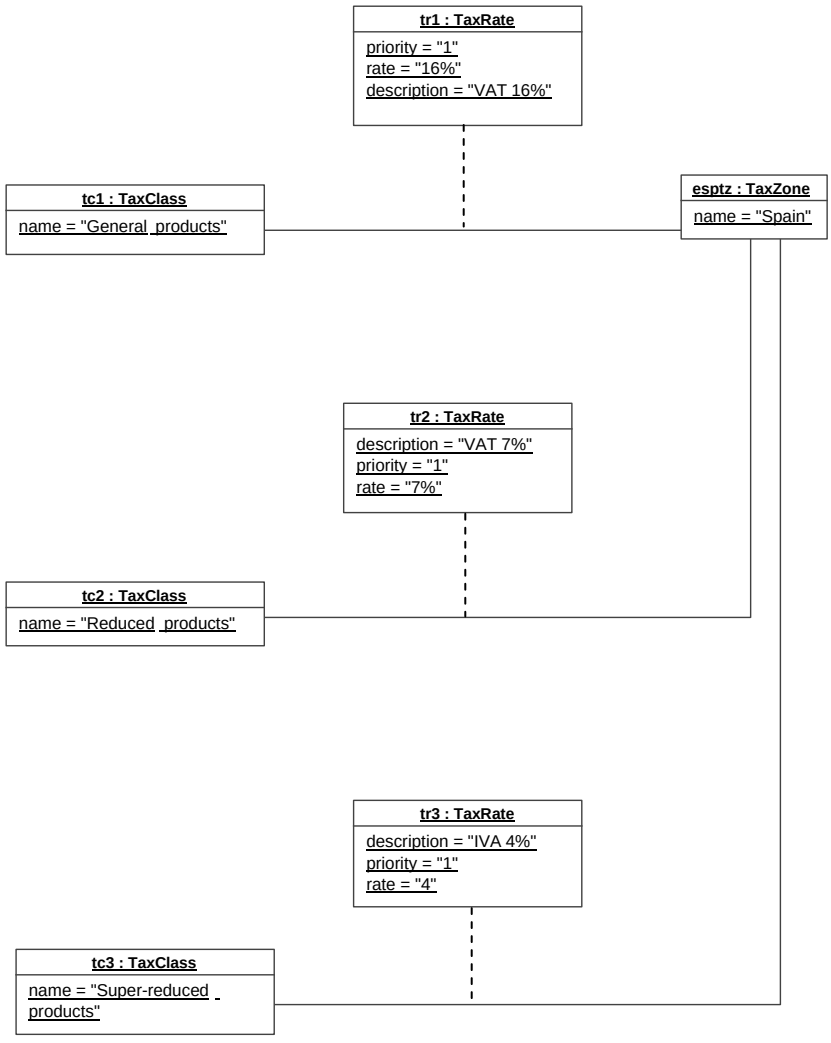
Examples

The Value Added Tax (VAT) in the European Union is a general and indirect consumption tax assessed on the value added to goods and services. Actually, rates are applied vary between Member States and between certain types of products.

In Spain, for example, there are three types of VAT: general VAT (16%), reduced VAT (7%) and super-reduced VAT (4%).

The following instantiation allows dealing with the three types of VAT for a Spanish *online* store:

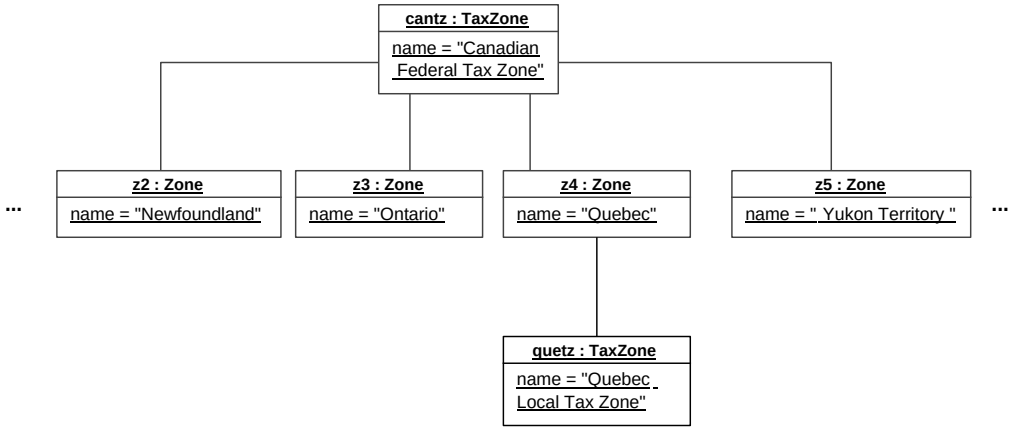


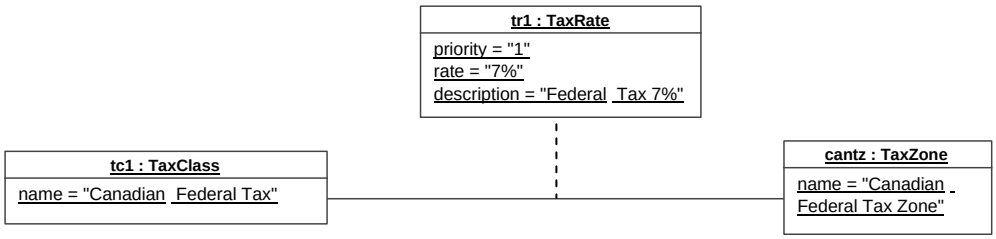


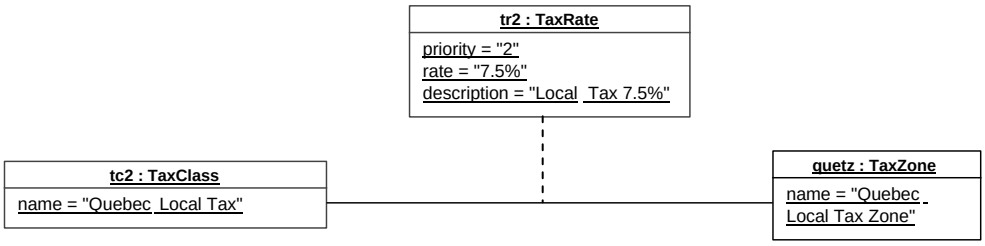
Otherwise, customers located in Quebec must pay a federal tax rate of 7% and a compounded local tax rate of 7.5%.

Note that for a product bought in Quebec, you should pay a 7% and 7.5% compounded tax. That is, you should add to your product's price a 15,025% of taxes (1.075 x 0.07 + 0.075 = 0.15025).

The following instantiation allows dealing with Canadian Federal Tax and Quebec Local Tax:







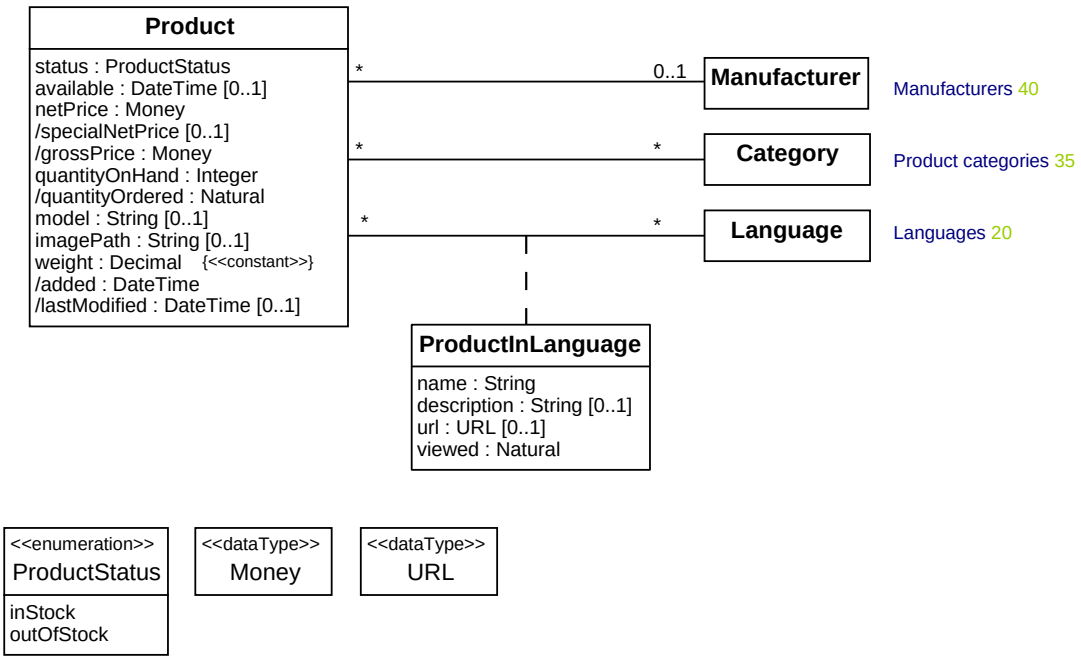
2.5 STORE ADMINISTRATION

Products

Overview

The system must know the information about the products offered by the online store.

Structural Schema



Operations

```
context Product def:
  addTaxes(z:Zone, basePrice:Money) : Money =
    let appliedTaxRates:Set(TaxRate)=
    z.taxZone.taxRate -> select (tr | tr.taxClass = self.taxClass)
    in
      let priorities:Set(Natural) =
        if appliedTaxRate -> isEmpty() then set{}
        else appliedTaxRates -> sortedBy(priority).priority -> asSet()
        endif
      in
```

```
if priorities -> isEmpty() then basePrice
else priorities -> iterate (p:Natural; res:Money = 0 |
                            res +
                            (((appliedTaxRates -> select (tr | tr.priority = p).rate
                             -> sum()) / 100)+1)*basePrice)
endif
```

Derivation Rules

[1] *Product::**grossPrice*** is the product's *netPrice* taking into account the applied taxes.

```
context Product::grossPrice(): Money
body :  self.addTaxes(Store.allInstances() -> any(true).zone, self.netPrice)
```

[2] *Product::**specialNetPrice*** is the special price, if the product is an active special.

```
context Product::specialNetPrice(): Money
body :
  if self.oclIsTypeOf(Special) then
    if self.oclAsType(Special).specialStatus=Status::enabled and
      self.oclAsType(Special).expiryDate < Now()
    then self.oclAsType(Special).specialPrice
    else set{}
    endif
  else set{}
  endif
```

[3] *Product::**added*** is the *DateTime* of product creation.

```
context Product::added(): DateTime
  body : Now()
```

Constraints

[1] A product is identified by a name in a language.

```
context Language::nameIsUnique(): Boolean
 body : self.productInLanguage -> isUnique(name)
```

Description

OsCommerce saves the following information about products:

- **Status:** Indicates whether the product is either in stock or out of stock.
- **Available:** The date since the product will be available.

- **Net price:** The product's price without taxes.
- **Gross price:** The product's price taking into account the taxes applied in the zone of the store. *Derived attribute*.
- **Special net price:** If the product has an active special offer, the current product price is the special price. Otherwise, this attribute is empty. *Derived attribute.*
- **Quantity on hand:** The product's quantity in stock.
- **Quantity ordered:** This attribute is updated by the system and keeps how many products have been sold.
- **Model:** An additional information field for products. It can be used, for example, for specifying the product model number.
- **Image path:** Every product can be associated to an image, which is located in the file indicated by this attribute.
- **Weight:** The product's weight. It is used for calculating the shipping costs in some shipping methods.
- **Added:** The *DateTime* when the product was created. *Derived attribute.*
- **Last modified:** The last time when the product was modified.
- **Manufacturer:** The product's manufacturer.
- **Category:** Products are classified into categories. Therefore, products can belong to categories. In case that a product is not associated to a category, it is assumed that it belongs to the top of categories hierarchy.

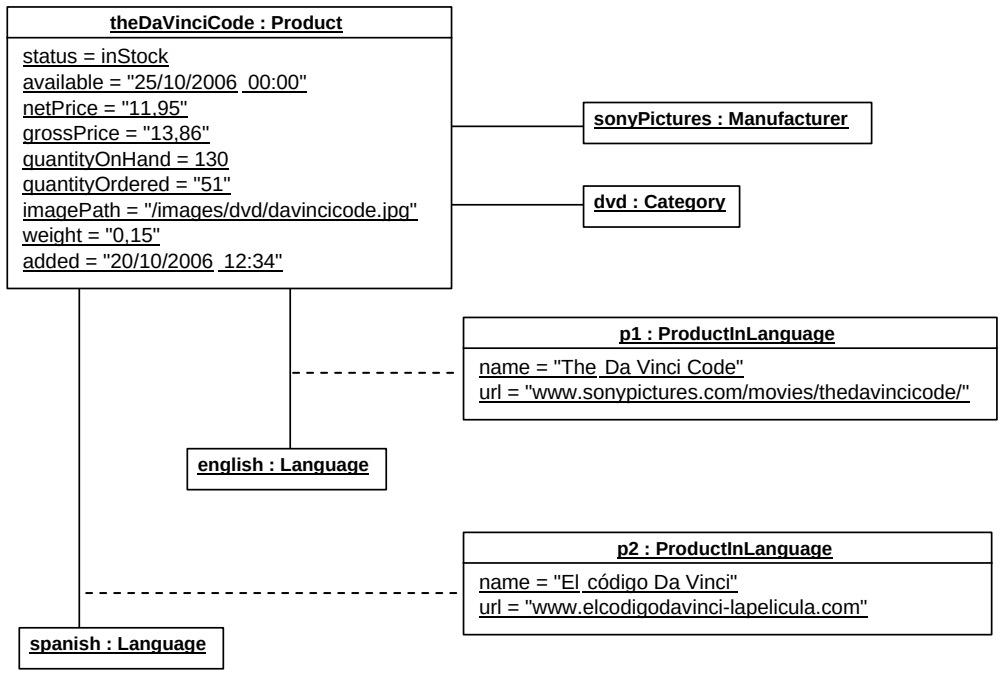
Moreover, the following attributes of a product can have different values in each language:

- **Name:** The name which identifies the product.
- **Description:** The product's description.
- **URL:** The web page where founding more information about the product.
- **Times viewed:** It is updated by the system and gives information about how many times the product has been viewed.

Example

Nowadays, there are some *online* shops, based on *osCommerce*, which offers music and entertainment products, like DVD's.

The next one is an example instantiation of a film, sold on DVD, which can be found as a product in some of those shops:

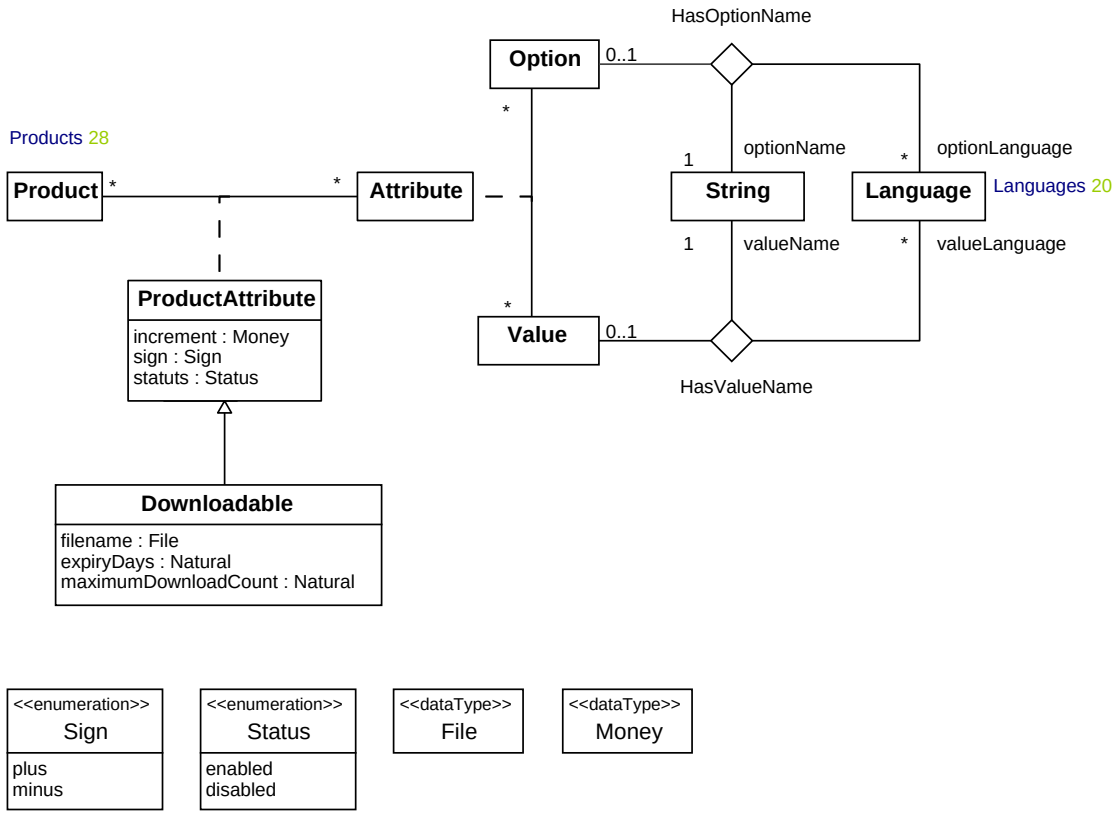


Product attributes and options

Overview

osCommerce allows defining several attributes for each product. Product attributes are used to offer multiple options of a product without needing to create many separate but very similar products.

Structural Schema



Constraints

[1] In each language, each product option has a unique name.

context Language::optionNameIsUnique(): Boolean
body : self.hasOptionName -> isUnique(optionName)

[2] In each language, each product value has a unique name.

```
context Language::valueNameIsUnique(): Boolean
  body : self.hasOptionValue -> isUnique(valueName)
```

Description

Usually, there are products which are sold in different options.

An attribute is an option/value pair such as, for example, *Size/Small, Size/Medium, Color/Red* or *Color /Yellow.*

Product attributes can be enabled or disabled setting up the attribute ***status***.

Moreover, the "base price" of products goes up or goes down according to the attributes chosen by the customer. The amount of money incremented or decremented by choosing a product attribute can be specified by setting up:

- **Sign:** The sign of the increment (plus or minus).
- **Increment:** The amount of money incremented or decremented.

There is a specific type of product attribute which allows products being downloadable. In this case, the system requires information about:

- **File Name**
- **Expiry days:** How many days, since the product was ordered, the download will be enabled.
- **Maximum downloads:** How many times the customer can download the product.

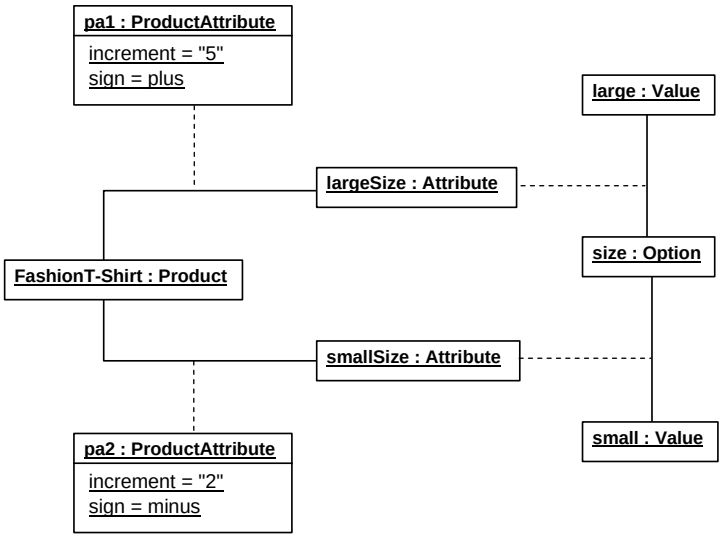
Remember that there are configuration properties which specify a general value for this downloadable product attributes.

Finally, options and values can have different names in each language.

Examples

This is an instantiation of a typical product offered by fashion shops:

A T-Shirt which can be bought in two sizes: large or small.

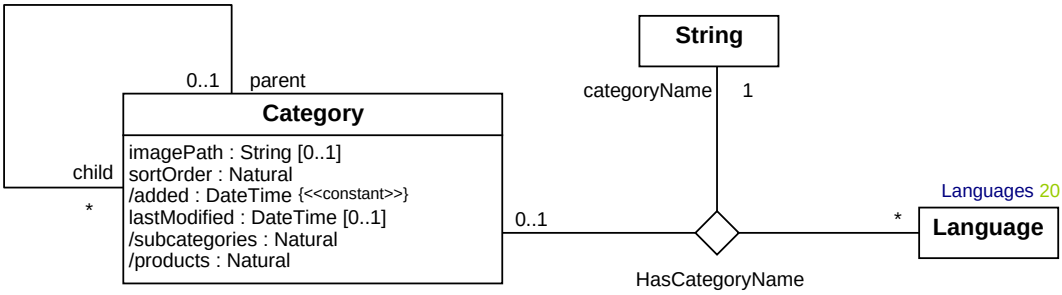


Product categories

Overview

Products are grouped into categories which are arranged hierarchically.

Structural Schema



Operations

context Category **def:**
allParents() : Set(Category) = self.parent -> union(self.parent.allParents())

Derivation Rules

[1] *Category::**added*** is the *DateTime* of category creation.

context Category::added():DateTime
body : Now()

[2] *Category::**subcategories*** is the number of subcategories owned by the category.

context Category::subcategories(): Natural
body : self.child -> size()

[3] *Category::**products*** is the number of products owned by the category.

context Category::products(): Natural
body : Category.allInstances() -> select(c | c.allParents() -> includes(self)).product -> size()

Constraints

[1] In each language, each category has a unique name.

context Language::categoryNameIsUnique(): Boolean
body : self.hasCategoryName -> isUnique(name)

[2] There are no cycles in category hierarchy.

context Category::isAHierarchy(): Boolean
body : not self.allParents() -> includes(self)

Description

OsCommerce groups products into categories which are arranged hierarchically. Categories are identified by a name in each language and have the following attributes:

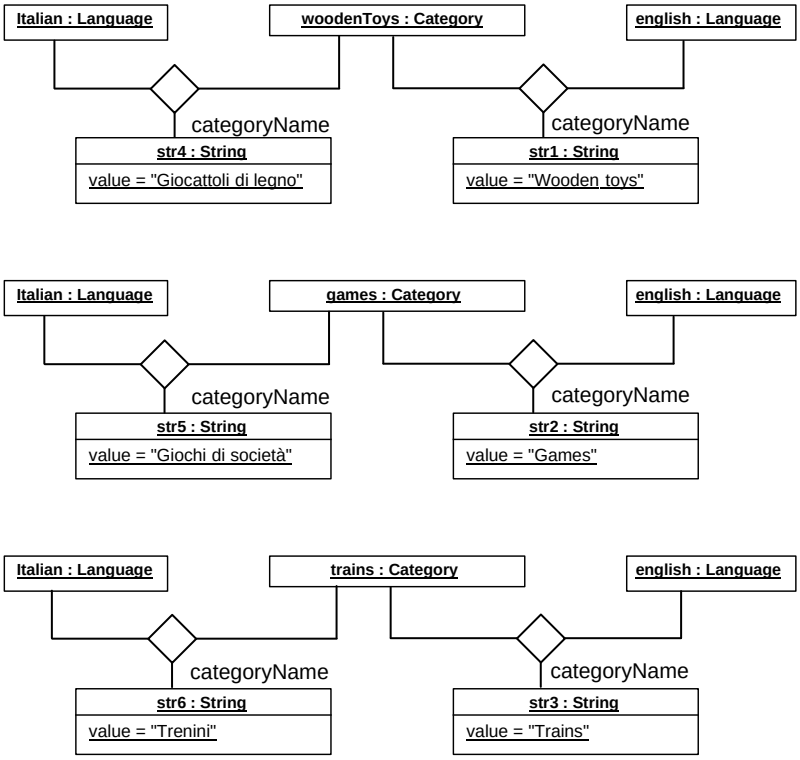
- **Image path:** Categories can be associated to an image, which is located in the file specified by this attribute.
- **Sort order:** The categories of the same hierarchical level are displayed as indicated by their sort order. In case that the sort order is the same, these are displayed alphabetically ordered.
- **Added:** The DateTime when the category was created. *Derived attribute.*
- **Last modified:** The last time when the category was modified.
- **Subcategories:** The quantity of subcategories. *Derived attribute.*
- **Products:** The quantity of products contained in the category.

Examples

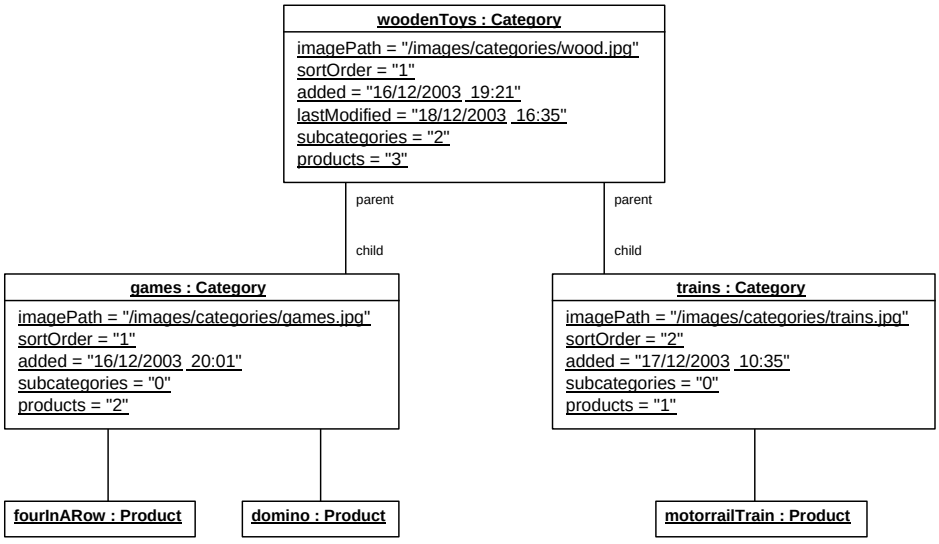
L' Isola dei bambini is an Italian toys shop based on *osCommerce*.

The following is a real example which shows the instantiation of some product categories from that shop. There are three categories: Wooden toys, Games and Trains.

Like products, categories can have different names in each language. In the example, we assume that there are two languages: Italian and English.



Wooden toys are at the top of the categories hierarchy and Games and Trains are subcategories of wooden toys.

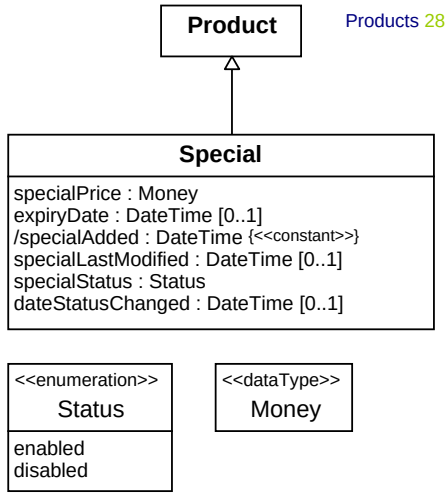


Specials

Overview

osCommerce allows offering specials. That is, lower prices for a set of products can be offered during a specific time period.

Structural Schema



Derivation Rules

[1] *Special::**added*** is the *DateTime* when the special was created

context Special::added():DateTime
body : Now()

Description

Specials are products which are sold, during an interval of time, in a lower price. The information kept by the system about specials is:

- **Special price**: The product's price during the special offer.

- **Expiry date:** The date until the special offer is active.
- **Added:** The DateTime when the *Special* was created. *Derived attribute.*
- **Last modified:** The last time when the *Special* was modified.
- **Status:** Specials can be enabled or disabled by setting this attribute.
- **Date status changed:** The system updates automatically the last time when the status of the *Special* was modified.

Examples

Egyptian Jewellery Online is an online shop based on *osCommerce.* This is an instance corresponding to one of their special offers:

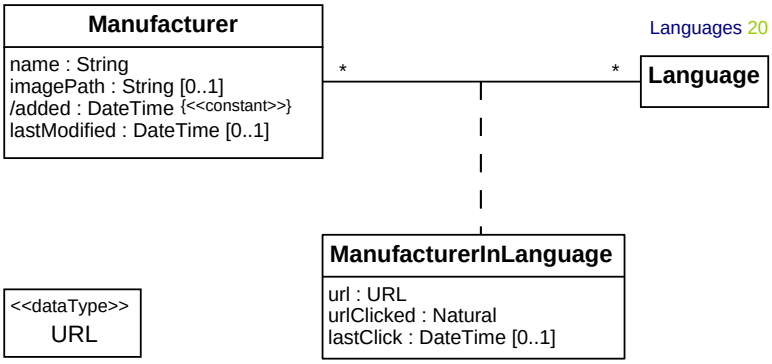
goldWingedHorusNecklace : Special
status = inStock available = "25/10/2006 00:00" netPrice = "1740" grossPrice = "2018,40" quantityOnHand = 2 quantityOrdered = "1" imagePath = "/images/necklaces/gen01.jpg" weight = "0,15" added = "20/10/2006 12:34" specialPrice = "1400" specialStatus = enabled specialAdded = "17/02/2007 15:32"

Manufacturers

Overview

In *osCommerce*, the products in the store are manufactured by manufacturers.

Structural Schema



Derivation Rules

[1] *Manufacturer::**added*** is the *DateTime* when the *Manufacturer* was created.

context Manufacturer::added():DateTime
body : Now()

Constraints

[1] A manufacturer is identified by its name

context Manufacturer::nameIsUnique(): Boolean
body : Manufacturer.allInstances() -> isUnique(name)

[2] Each manufacturer must have a URL in each language

context Manufacturer::aURLInEachLanguage(): Boolean
body : self.language ->size() = Language.allInstances() -> size()

Description

OsCommerce keeps the following information about manufacturers:

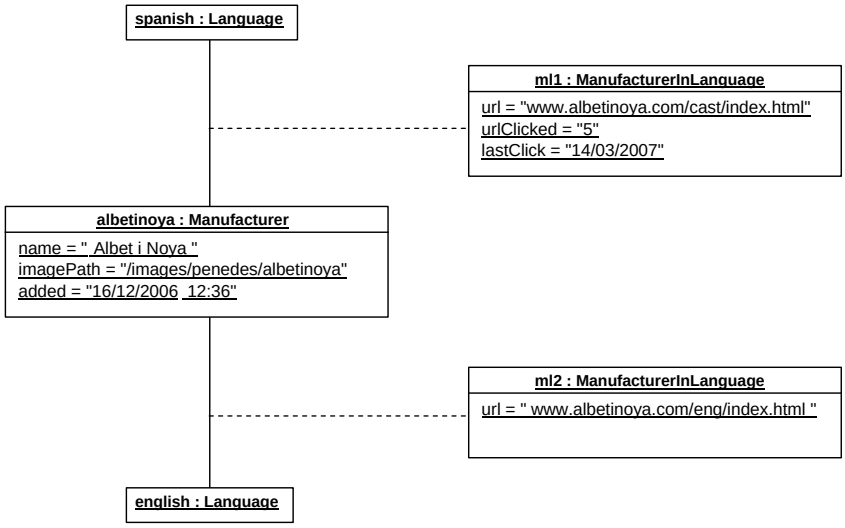
- **Name:** The manufacturer's name.
- **Image path:** Manufacturers can be illustrated by an image, which is located in the file specified by this attribute.
- **Added:** The DateTime when the manufacturer was created. *Derived attribute*.
- **Last modified:** The system updates automatically the last time when the manufacturer information was modified.

Moreover, for each language, each manufacturer must have a **URL**, in order to allow customers to obtain information about it. The system updates automatically **how many times the URL has been clicked** and **when the last clicked was**.

Example

Vinoverde is a German wine online shop. This based on *osCommerce* online store sells wines of the Penedès guarantee of origin, among others.

This is a real instantiation of one of the manufacturers of this famous wine region, whose products are sold in the example store:

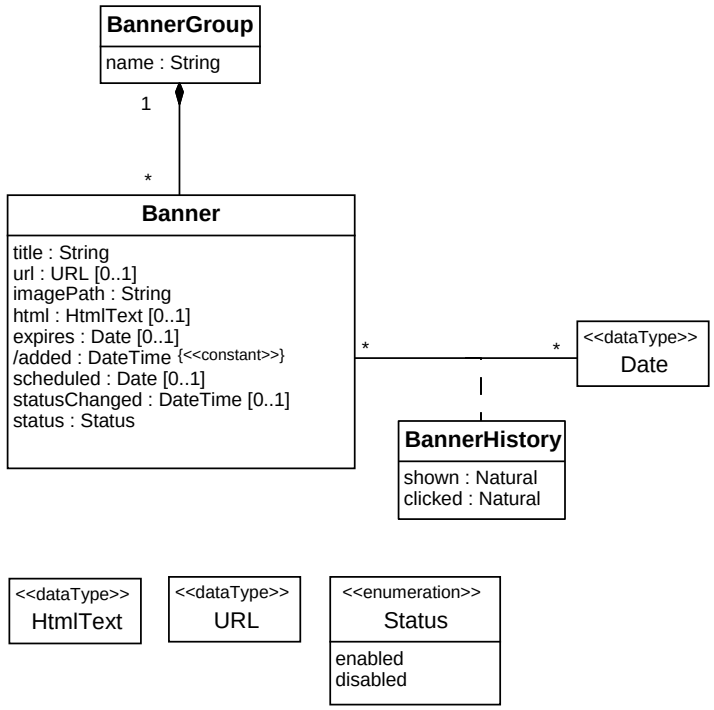


Banners

Overview

osCommerce allows administrating banners published in the *online* store.

Structural Schema



Derivation Rules

[1] *Banner::**added*** is the *DateTime* when the banner was created.

context Banner::added():DateTime
body : Now()

Constraints

[1] A Banner is identified by its title.

context Banner::titleIsUnique: Boolean
body : Banner.allInstances() -> isUnique(title)

[2] A Banner Group is identified by its name.

context BannerGroup::nameIsUnique: Boolean
body : BannerGroup.allInstances() -> isUnique(name)

Description

Banners are images which are shown in the *online* store. *osCommerce* allows administrating the current banners and adding new banners. The system saves the following information about them:

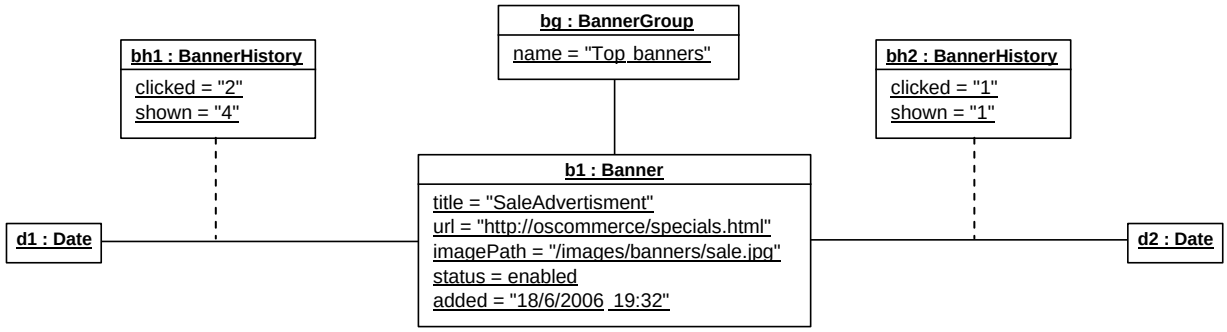
- **Title:** A name which identifies the banner.
- **URL:** The URL where a customer is redirected when the banner is clicked.
- **Image path:** The file where the banner is located.
- **Size:** The banner's size (width and height).
- **Title:** A name which identifies the banner.
- **Html:** HTML based banners can be defined by this attribute.
- **Expires:** When the expiry date is reached, the banner is automatically disabled.
- **Added:** The DateTime when the banner was created. *Derived attribute*.
- **Scheduled:** A future date when the banner is to become active. If no scheduled date is defined, the banner is automatically published when it is created.
- **Status changed:** The system updates automatically the last time when the banner status was modified.
- **Status:** The banner can be enabled or disabled.

Banners grouped into the same ***BannerGroup*** are shown iteratively in the same place.

Moreover, the system maintains **historical information** about how many times the banner has been shown or clicked every day.

Example

This is a possible instantiation of a banner used as an advertisement during the sales:

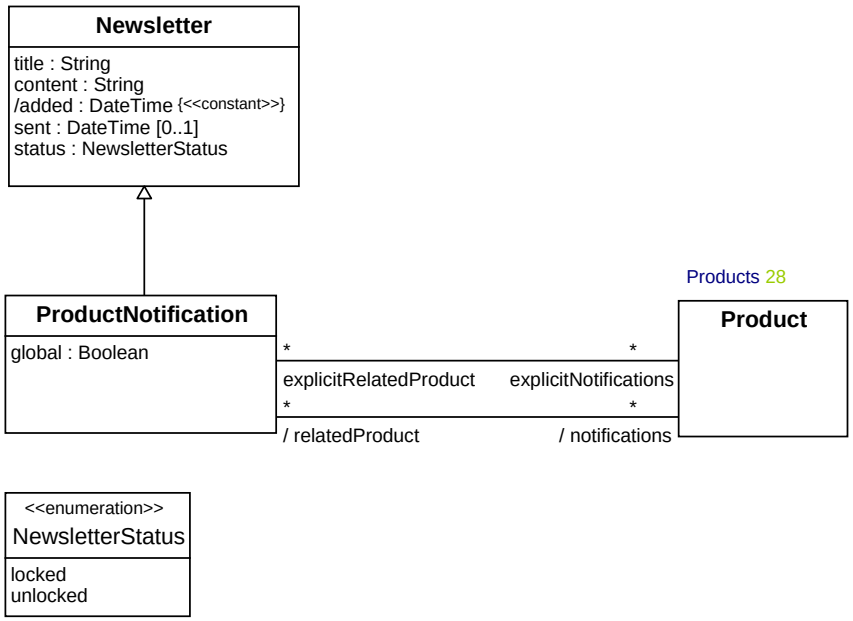


Newsletters

Overview

osCommerce allows store owners sending emails and product notifications to customers.

Structural Schema



Derivation Rules

[1] *ProductNotification::**notifications*** is the set of implied products in the notification.

```
context ProductNotification::notifications():Set(Product)
body :
   if self.global then Product.allInstances()
   else self.explicitNotifications
   endif
```

[2] *ProductNotification::**added*** is the *DateTime* when the newsletter was created.

```
context Newsletter::added():DateTime
  body : Now()
```

Constraints

[1] A Newsletter is identified by its title.

context Newsletter::titleIsUnique: Boolean
body : Newsletter.allInstances() -> isUnique(title)

Description

Newsletters are used to send emails to the customers who gave their email address when they created his account.

The system saves the following information about newsletters:

- **Title:** A name which identifies the newsletter.
- **Content:** The e-mail's content.
- **Added:** The DateTime when the newsletter was created. *Derived attribute*.
- **Sent:** The System sets up, automatically, the date when the newsletter was sent.
- **Status:** Newsletters can be locked or unlocked. If a Newsletter is locked, it cannot be modified by any administrator. Only if the newsletter is locked, it can be sent.
- **Status:** The banner can be enabled or disabled.

Newsletters are sent to all the customers who selected to receive newsletters upon creating their user account.

However, there is a specific type of newsletter, called ***ProductNotification.*** This particular type of newsletter is sent only to customers who selected to be notified about product updates included in the list of products implied in the notification.

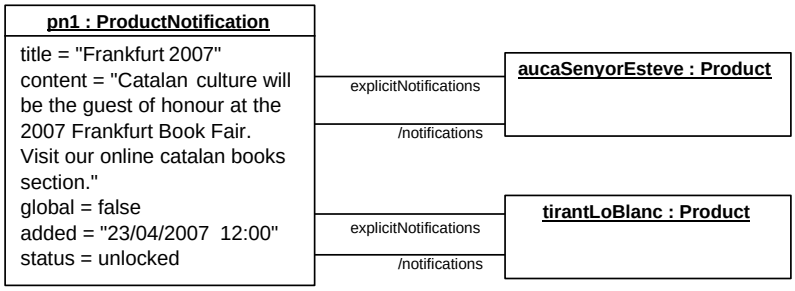
The list of products involved in the notification is represented by the derived association *notifications.*

New implied products are specified through the association *explicitNotifications.* If the attribute *global* is true, all products are implied in the product notification.

Example

Catalan culture will be the guest of honour at the 2007 Frankfurt Book Fair considered to be the most important event in the world of publishing.

Imagine that the administrator of an online bookshop wants to send a newsletter to customers who bought books like *L'auca del senyor Esteve* or *Tirant Lo Blanc*, famous Catalan books, to inform them about this event:



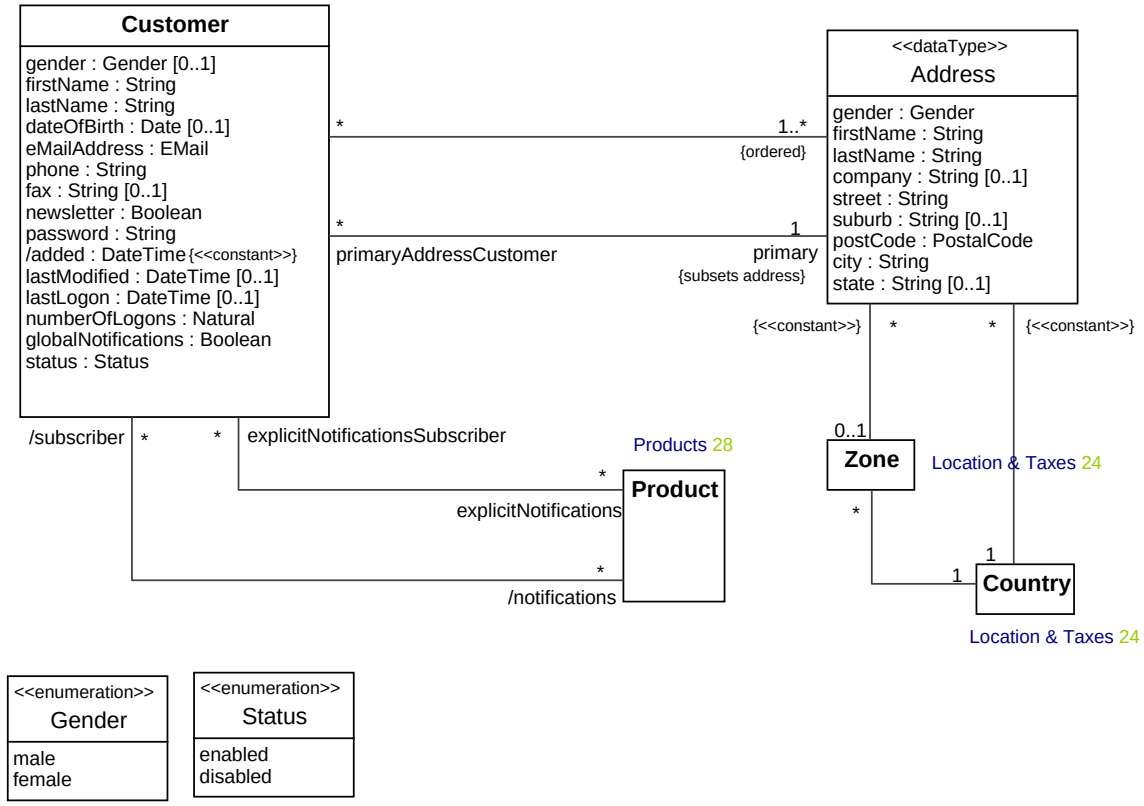
2.6 CUSTOMERS

Customers

Overview

osCommerce keeps information about customers and their addresses, one of which is the primary address.

Structural Schema



Derivation Rules

[1] *Customer::**notifications*** is the set of subscriptions to product notifications.

```
context Customer::notifications():Set(Product)
body :
   if self.globalNotifications then Product.allInstances()
   else self.explicitNotifications
   endif
```

[2] *Customer::**added*** is the *DateTime* of the customer creation.

```
context Customer::added():DateTime
  body : Now()
```

Constraints

[1] Customers are identified by their email address.

```
context Customer::eMailIsUnique(): Boolean
body : Customer.allInstances() -> isUnique(eMailAddress)
```

[2] Addresses have zone if needed.

```
context Country::addressesHaveZoneIfNeeded(): Boolean
body :
   self.zone -> notEmpty() implies self.address -> forAll
   (a | a.state = a.zone.name and self = a.zone.country)
```

Description

osCommerce has the following information about Customers :

- **Gender**
- **First Name**
- **Last Name**
- **Date of Birth**
- **Email address**
- **Phone**

- **Fax**
- **Password**

The System also maintains the *DateTime* of the **last modification**, the *DateTime* of the **last logon** and the customer's **number of logons**. There is a derived attribute (***added***) which indicates when the *Customer* was created.

Additionally, Customers can be subscribed to product notifications. This fact is represented by the association role ***explicitNotifications.*** If the attribute ***globalNotifications*** is true, then the customer will receive notifications for all the products of the store. The derived association ***notifications***, keeps the active subscriptions to product notifications associated to the customer, taking into account the explicit notifications and the attribute *globalNotifications*.

Customers, in *osCommerce*, have one or more **addresses**, one of which is the primary. The primary address is the default shipping and delivering addresses for the orders placed in the store.

The **status** of a customer indicates if the customer is currently active.

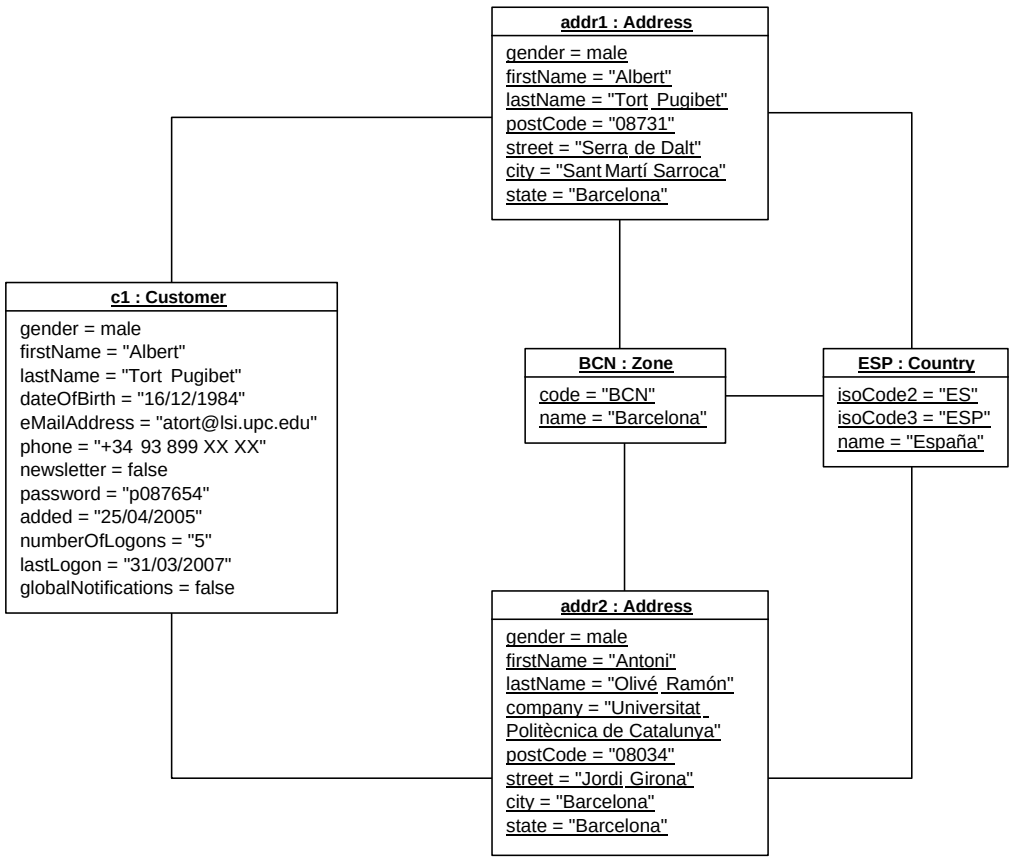
osCommerce has the following information about *Addresses*:

- **Gender**
- **First Name**
- **Last Name**
- **Company**
- **Street**
- **Suburb**
- **Postal Code**
- **City**
- **State**

Moreover, Addresses are located in a Country. If the Country has zones, the address must be located in a zone whose name is the same as the name of the state, and the country of the zone must be the same as the country of the address.

Examples

The following instantiation is an example of a customer with two address book entries. The customer can choose, for each order, which one is the shipping and the billing address:



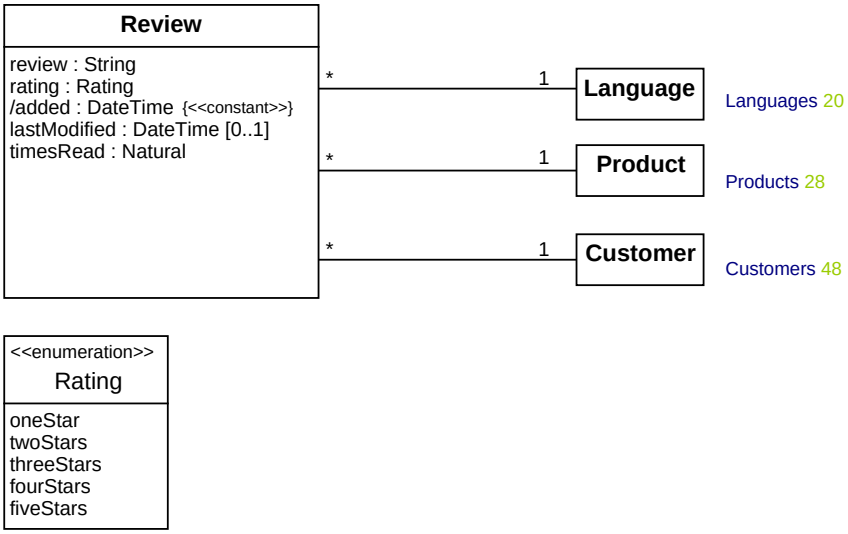
2.7 ONLINE CATALOG

Reviews

Overview

In order to allow users reading evaluations of a product, customers can write reviews.

Structural Schema



Derivation Rules

[1] *Review::**added*** is the *DateTime* of the review creation.

```
context Review::added():DateTime
  body : Now()
```

Description

Reviews are customer evaluations of a product and are written in a **language**.

osCommerce takes into account the following information about reviews:

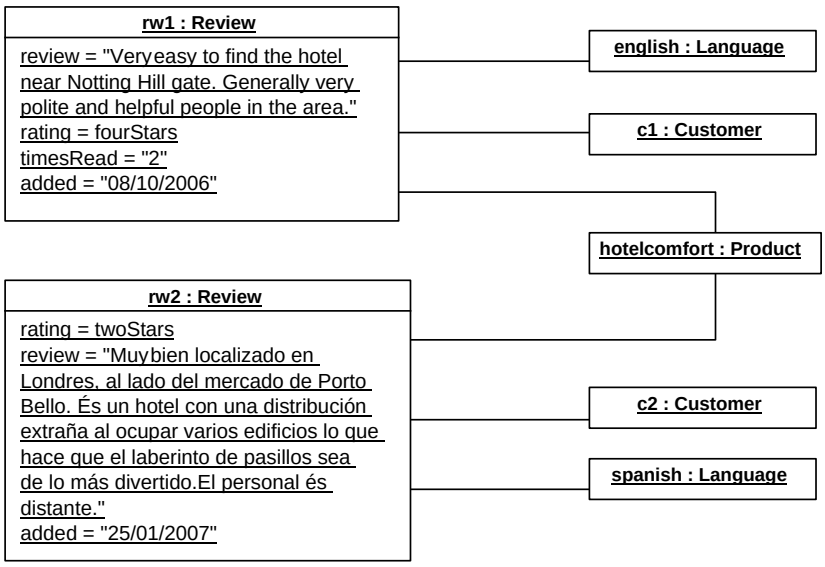
- **Review:** The customer's opinion about the product**.**
- **Rating:** The rating for the product.
- **Added:** The DateTime when the review was created. *Derived attribute.*
- **Last modified:** The system updates automatically the last time when the review information was modified.
- **Review read:** The system updates automatically how many times a review has been read.

Examples

osCommerce is a solution used in some travel and tours online shops. In these online shops, travel packages are sold as products.

Usually, in this kind of shops, users write reviews about their impression about the hotels where they stayed during travels.

While they are surfing the online store, customers can read the reviews in order to obtain more information about products.

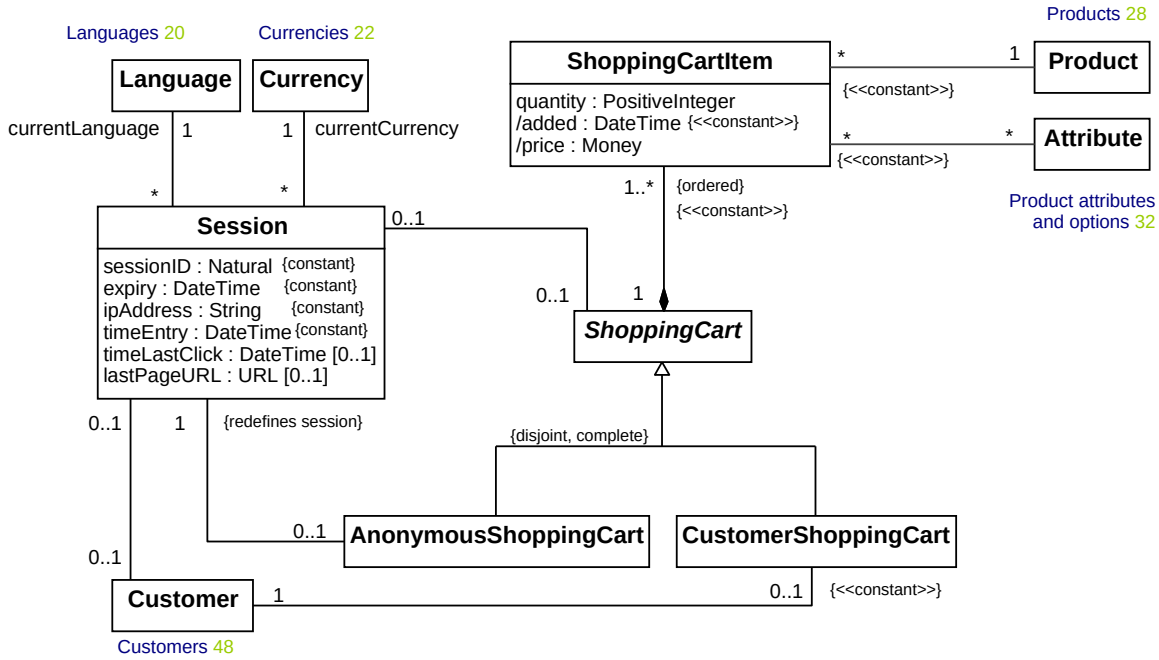


Shopping carts

Overview

Customers can add or remove products from their shopping carts while they are surfing the *online* store.

Structural Schema



Derivation Rules

[1] *ShoppingCartItem::**price*** is the net price for an item taking into account the selected product attributes.

```
context ShoppingCartItem::price():Money
body :
   let netPriceWithSpecial:Money =
            if self.product.specialNetPrice ->notEmpty() then self.product.specialNetPrice
            else self.product.netPrice
            endif
   in
   if  self.attribute -> isEmpty() then self.product.netPriceWithSpecial
   else
      self.attribute.productAttribute -> select (pa | pa.product = self.product) -> collect
      (if sign = Sign::plus
```

```
    then increment
     else –increment
    endif) -> sum() + self.product.netPriceWithSpecial
  endif
```

[2] *ShoppingCartItem::***added** is the *DateTime* when the item was created.

```
context ShoppingCartItem::added():DateTime
  body : Now()
```

Constraints

[1] If a customer shopping cart exists in the context of a session then its customer is the customer of the session

```
context CustomerShoppingCart::sameCustomer(): Boolean
  body : self.session.customer -> notEmpty() implies self.session.customer = self.customer
```

[2] The shopping cart item specifies the selected product attributes, which must be a subset of all the product attributes.

```
context ShoppingCartItem::productHasTheAttributes(): Boolean
  body : self.product.attribute -> includesAll(self.attribute)
```

[3] The shopping cart item specifies only one attribute per option.

```
context ShoppingCartItem::onlyOneAttributePerOption(): Boolean
  body : self.attribute -> isUnique(option)
```

[4] Sessions are identified by its sessionID.

```
context Session::sessionIDIsUnique(): Boolean
  body : Session.allInstances() -> isUnique (sessionID)
```

Description

Shopping carts contains the products chosen by customers from the *online* catalog.

A shopping cart is **anonymous** until de customer logs in. At this moment, if the customer didn't have a previous *CustomerShoppingCart*, it becomes a ***CustomerShoppingCart***. If the customer had a previous customer shopping cart, the anonymous shopping cart is removed from the system and their products are added to the previous *CustomerShoppingCart*.

In summary, if a customer leaves a session with a non-empty customer shopping cart, then the cart will be automatically restored in his next session.

Anonymous shopping carts can only exist in the context of a session, and they are automatically removed when its session expires.

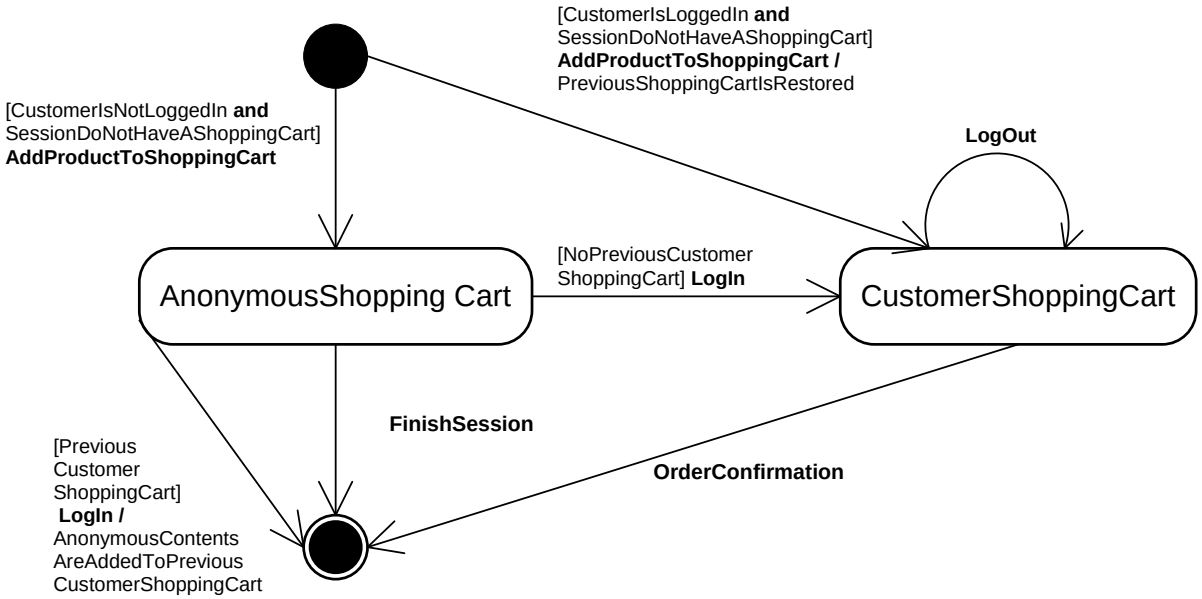
A shopping cart contains a sequence of one or more ***ShoppingCartItem***, each of which is a **quantity** of a product. If the product has attributes then the shopping cart item specifies the **selected attributes** of the product.

Moreover, there is a derived attribute which calculates the **net price for the shopping cart item**, taking into account the net price of the product and the increments or decrements of the attributes. Note that the net price of the product is the special net price if it is an active special offer.

When an order, corresponding to a shopping cart, is confirmed, the shopping cart is removed from the system.

State Transition Diagram

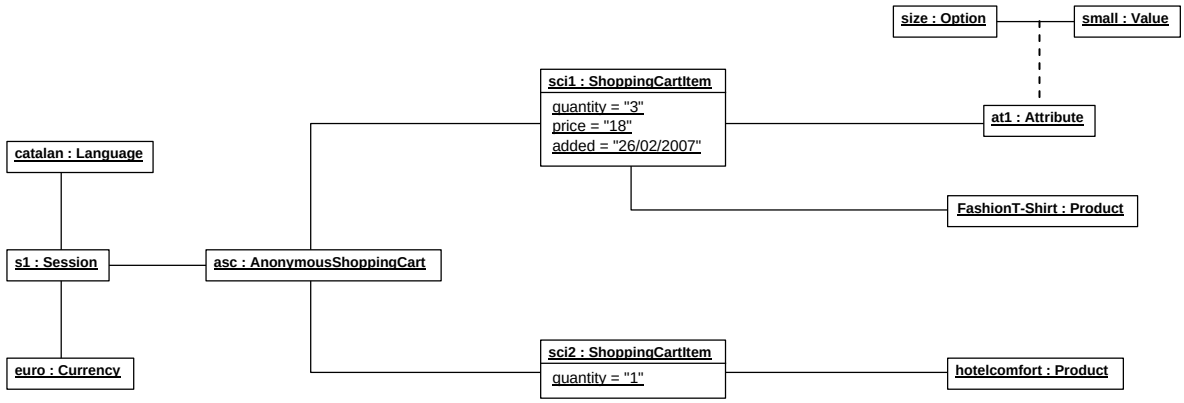
The state of *ShoppingCart* entities can be conceptually modelled by using a state transition diagram.



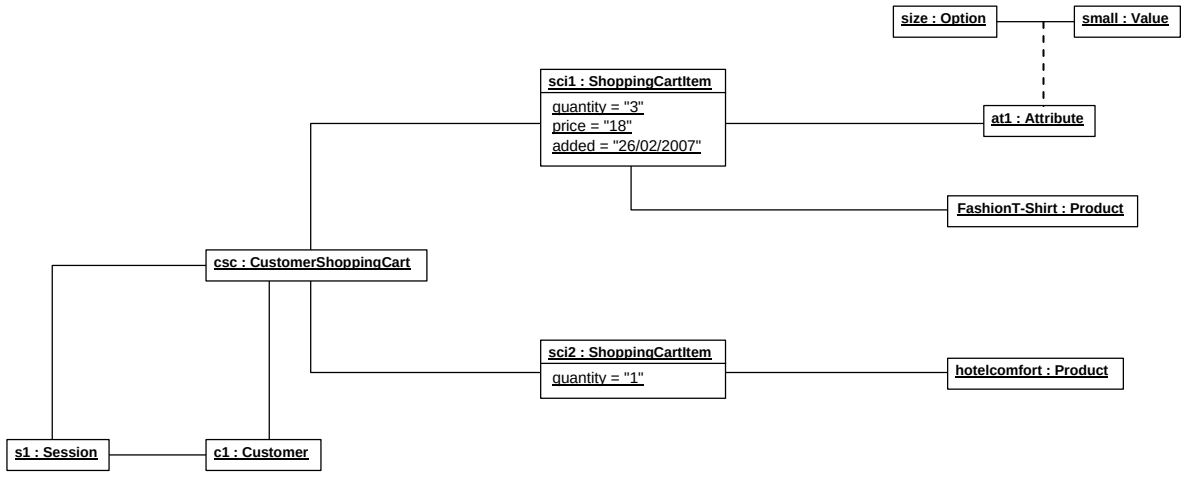
Examples

These are two example instantiations of an anonymous shopping cart and a customer shopping cart in the context of a session.

The first example represents an anonymous shopping cart:



The second example represents a customer shopping cart which will not disappear when the session expires. Thanks to it, the shopping cart will be restored when the customer initiates a new session.

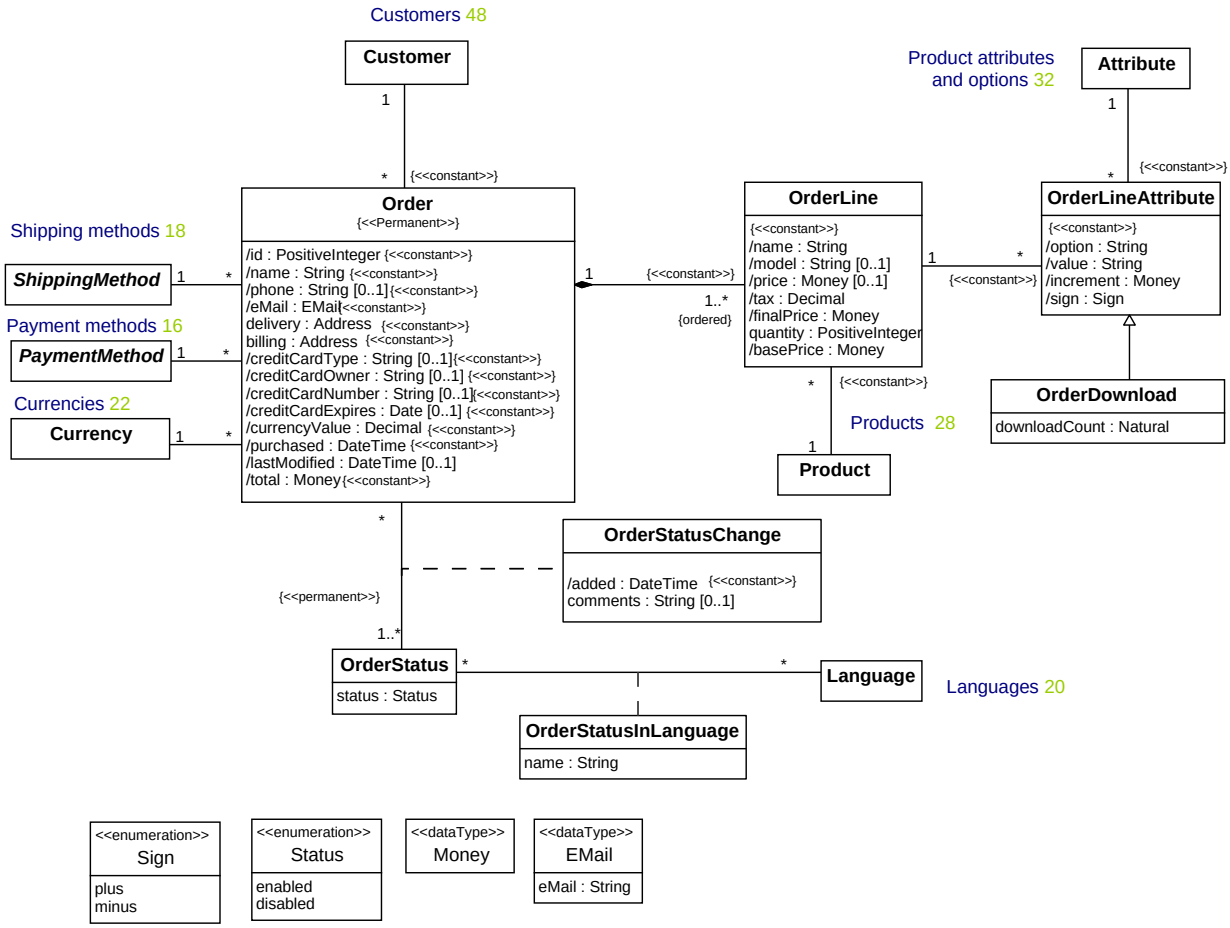


Orders

Overview

Orders are the confirmation that a customer wants to buy the contents of his shopping cart.

Structural Schema



Operations

```
context  ShippingMethod def:
   addTaxes(z:Zone, basePrice:Money) : Money =
      let appliedTaxRates:Set(TaxRate)=
         z.taxZone.taxRate -> select (tr | tr.taxClass = self.taxClass)
       in
          let  priorities:set(Natural) =
```

```
        if appliedTaxRate -> isEmpty() then set{}
        else appliedTaxRate -> sortedBy(priority).priority -> asSet()
        endif
    in
        if priorities -> isEmpty() then basePrice
        else priorities -> iterate (p:Natural; res:Money = 0 |
                res +
                (((appliedTaxRates -> select (tr | tr.priority = p).rate
                -> sum()) / 100)+1)*basePrice
        endif

context ShippingMethod def:
  shippingCosts(totalWeight:Decimal, totalPrice:Money, quantity:PositiveInteger): Money = 0

context FlatRate def:
  shippingCosts(totalWeight:Decimal, totalPrice:Money, quantity:PositiveInteger): Money = self.cost

context PerItem def:
  shippingCosts(totalWeight:Decimal, totalPrice:Money, quantity:PositiveInteger): Money =
    self.cost*quantity

context TableRate def:
  shippingCosts(totalWeight:Decimal, totalPrice:Money, quantity:PositiveInteger): Money =
    if self.method = ShippingTableMethod::weight
    then
      self.items -> select (i | i.number <= (totalWeight*quantity)) -> sortedBy(number) ->last().cost
    else
      self.items -> select (i | i.number <= (totalPrice*quantity)) -> sortedBy(number) ->last().cost
    endif

context USPostalService def:
  shippingCosts(totalWeight:Decimal, totalPrice:Money, quantity:PositiveInteger): Money =
    calculateFromUSPS (self.userID, self.password, self.server, totalWeight, totalPrice, quantity)

context TableRate def:
  shippingCosts(totalWeight:Decimal, totalPrice:Money, quantity:PositiveInteger): Money =
    if self.method = ShippingTableMethod::weight
    then
      self.items -> select (i | i.number <= (totalWeight*quantity)) -> sortedBy(number) ->last().cost
    endif
```

Derivation Rules

[1] *Order::**id*** identifies the order and it is assigned automatically.

```
context Order::id():PositiveInteger
body :
  if Order.allInstances() -> size() = 0 then 0
  else Order.allInstances() -> sortedBy(id) -> last().id + 1
  endif
```

[2] *Order::**primary*** address of an order is that of its customer.

```
context Order::primary():Address
 body : self.customer.primary
```

[3] *Order::**eMailAddress*** of an order is that of its customer.

```
context Order::eMailAddress():EMail
  body : self.customer.eMailAddress
```

[4] *Order::**phone*** of an order is that of its customer.

```
context Order::phone():String
   body : self.customer.phone
```

[5] *Order::**purchased*** is the *DateTime* when the order was created

```
context Order::purchased():DateTime
  body : Now()
```

[6] *Order::**lastModified*** is the last *DateTime* when the status order was modified

```
context Order::lastModified():DateTime
  body : self.orderStatusChange -> sortedBy(added)  -> last().added
```

[7] *Order::**statuts*** is the current status of the order

```
context Order::status():OrderStatus
  body : self.orderStatusChange -> sortedBy(added) -> last().orderStatus
```

[8] *Order::**total*** gives the total amount of an order

```
context Order::total():Money
body :
   let totalWithoutShippingCosts:Money =
      self.orderLine -> collect(finalPrice*quantity) -> sum()
   let totalWeight:Decimal =
      self.orderLine -> collect(product.weight*quantity) -> sum()
   let quantity:PositiveInteger =
      self.orderLine.quantity -> sum()
   let handlingFee:Money =
      if self.shippingMethod.oclIsTypeOf(HandlingFeeMethod)
      then
         self.shippingMethod.oclAsType(HandlingFeeMethod).handlingFee
      else 0
      endif
   in
      let totalWeightIncreased:Decimal =
           if totalWeight* (ShippingAndPackaging.percentageIncreaseForLargerPackages/100) >
              ShippingAndPackaging.typicalPackageTareWeight
           then
             totalWeight * (1 +totalWeight*
             ShippingAndPackaging.percentageIncreaseForLargerPackages/100)
          else totalWeight + ShippingAndPackaging.typicalPackageTareWeight
          endif
       in
          totalWithoutShippingCosts +
               self.shippingMethod.shippingCosts
               (totalWeightIncreased, totalWithoutShippingCosts, quantity) + handlingFee
```

[9] *OrderStatusChange::**added*** is the *DateTime* when the change is done.

```
context OrderStatusChange::added():DateTime
body :  Now()
```

[10] *OrderLine::**name*** is that of its product in the default language

```
context OrderLine::name():String
body :
   self.product.productInLanguage
   ->select(pil | pil.language = Store.allInstances() -> any(true).defaultLanguage).name
```

[11] *OrderLine::**model*** is that of its product

```
context OrderLine::model():String
body :  self.product.model
```

[12] *OrderLine::**basePrice*** is the net price of the product without taking into account the selected attributes.

```
context OrderLine::basePrice():Money
body :
   if self.product.specialNetPrice ->notEmpty()
   then self.product.specialNetPrice
   else self.product.netPrice
   endif
```

[13] *OrderLine::**price*** is the net price of the product with the selected attributes

```
context OrderLine::price():Money
body :
   if  self.orderLineAttribute  -> isEmpty() then self.basePrice
   else
      self.orderLineAttribute -> collect
      (if sign = Sign::plus then increment
      else –increment
      endif) -> sum() + self.basePrice
  endif
```

[14] *OrderLine::**finalPrice*** is the price of the product with the selected attributes and taking into account the shipping costs and the taxes

```
context OrderLine::finalPrice():Money
body :
   if self.billing.zone -> notEmpty() then
      self.product.addTaxes(self.billing.zone, self.price)
   else self.price
   endif
```

[15] *OrderLineAttribute::**option*** is the option name in the default language

```
context OrderLineAttribute::option():String
body :
   self.attribute.option.hasOptionName
      -> select (hon | hon.optionLanguage = Store.allInstances()
      -> any(true).defaultLanguage).optionName
```

[16] *OrderLineAttribute::**value*** is the option value in the default language

```
context OrderLineAttribute::value():String
body :
   self.attribute.value.hasValueName
      -> select (hvn | hon.valueLanguage = Store.allInstances()
      -> any(true).defaultLanguage).valueName
```

[17] *OrderLineAttribute::**increment*** is the increment applied in the product price by the attribute

```
context OrderLineAttribute::increment():Money
body :
   self.attribute.productAttribute
      -> select (pa | pa.product = self.orderLine.product).increment
```

[18] *OrderLineAttribute::**sign*** is the sign of the increment applied in the product price by the attribute

```
context OrderLineAttribute::sign():Sign
body :
   self.attribute.productAttribute
   -> select (pa | pa.product = self.orderLine.product).sign
```

Constraints

[1] A specific zone shipping method with a specific tax zone can only be applied if the delivery address zone is included in the tax zone.

```
context Order::ApplicableZoneShippingMethod: Boolean
body :
   self.shippingMethod.oclIsTypeOf(SpecificZoneMethod) and
   self.shippingMethod.oclAsType(SpecificZoneMethod).taxZone -> notEmpty implies
   self.shippingMethod.oclAsType(SpecificZoneMethod).taxZone.zone
   -> includes(self.delivery.zone)
```

[2] The *Zone Rates* shipping method can only be applied in the specified countries.

```
context Order::ApplicableZoneRatesShippingMethod: Boolean
body :
   self.shippingMethod.oclIsTypeOf(ZoneRates)  implies
   self.shippingMethod.oclAsType(ZoneRates).country -> includes(self.delivery.country)
```

[3] Payment methods with a specified tax zone can only be applied in orders with a billing address located in a zone included in the tax zone.

```
context Order::ApplicableZonesPaymentMethod: Boolean
body :
   self.paymentMethod.taxZone -> notEmpty()  implies
   self.paymentMethod.taxZone.zone -> includes(self.billing.zone)
```

[4] Payment methods with a specified set of applicable currencies can only be applied if the current currency is included in that set.

```
context Order::ApplicableCurrenciesPaymentMethod: Boolean
body :
   self.shippingMethod.oclIsTypeOf(SpecificCurrenciesMethod)  implies
   self.shippingMethod.oclAsType(SpecificCurrenciesMethod).currency
   -> includes(self.currency)
```

[5] Orders are identified by its id

```
context Order::IDIsUnique: Boolean
body :  Order.allInstances() -> isUnique(id)
```

[6] Order status are identified by its name

```
context Order::NameIsUnique: Boolean
body :  OrderStatus.allInstances() -> isUnique(name)
```

Description

When the customer confirms that he/she wants to buy the contents of his shopping cart, the system generates an order. Orders have the following information:

- **Id:** Identifies an order. It is assigned automatically. *Derived Attribute*.
- **Billing address**
- **Delivery address**
- **Email Address:** A copy of the customer's email. *Derived Attribute.*
- **Phone:** A copy of the customer's phone. *Derived Attribute*.
- **Purchased:** The DateTime when the order was created. *Derived Attribute*.
- **Last Modified:** The last time the order was changed. *Derived Attribute*.
- **Status:** The current status of the order. *Derived Attribute*.
- **Total:** The total price of the order, taking into account the shipping costs and the taxes.

An order is made by a customer during *check out* process, and can be in different *OrderStatus* during time.

An order is composed by **order lines** which represents a quantity of a product. The information about order lines is:

- **Name:** A copy of the product's name. *Derived Attribute*.
- **Model:** The current status of the order. *Derived Attribute.*
- **Total:** The total price of the order, taking into account the shipping costs and the taxes.

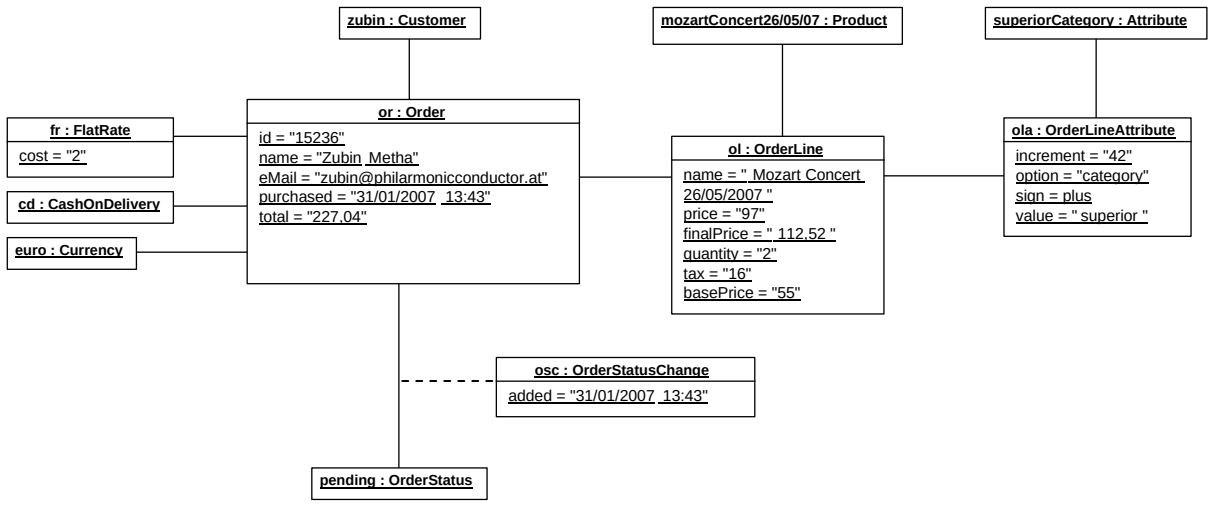
Likewise, order lines can have **order line attributes**, which represents the product attributes chosen for an order line product.

Order line attributes which corresponds to downloadable product attributes have an specific attribute, called ***downloadCount*** that maintains how many times the product has been download by the customer.

Example

The Vienna Mozart Orchestra plays concerts in Vienna in the most famous concert halls, including the Golden Hall in the Musikverein, a well-known building where takes place the most popular New Year's concert.

Imagine that we would like to implement a ticket online shop based on the *osCommerce* solution. The following would be a possible instantiation of an order:



BEHAVIORAL
SCHEMA

3 BEHAVIORAL SCHEMA

3.1 INTRODUCTION

In the next sections we develop the behavioral schema of the *osCommerce* system.

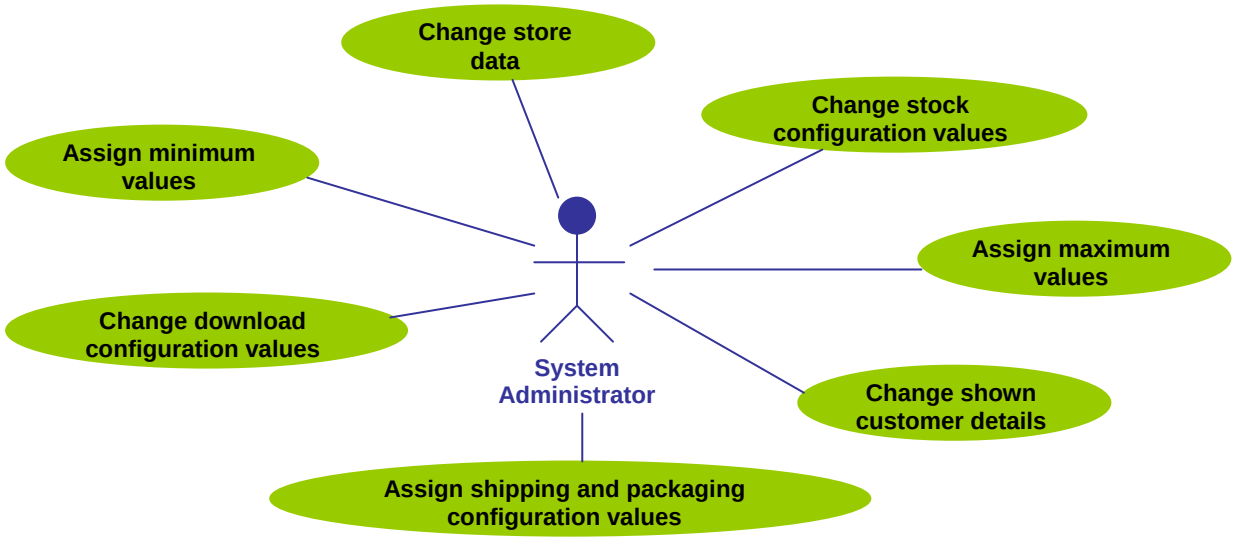
The main purpose of the *osCommerce* behavioral schema is specifying the valid changes in the domain state, as well as the actions that the *osCommerce* system can perform.

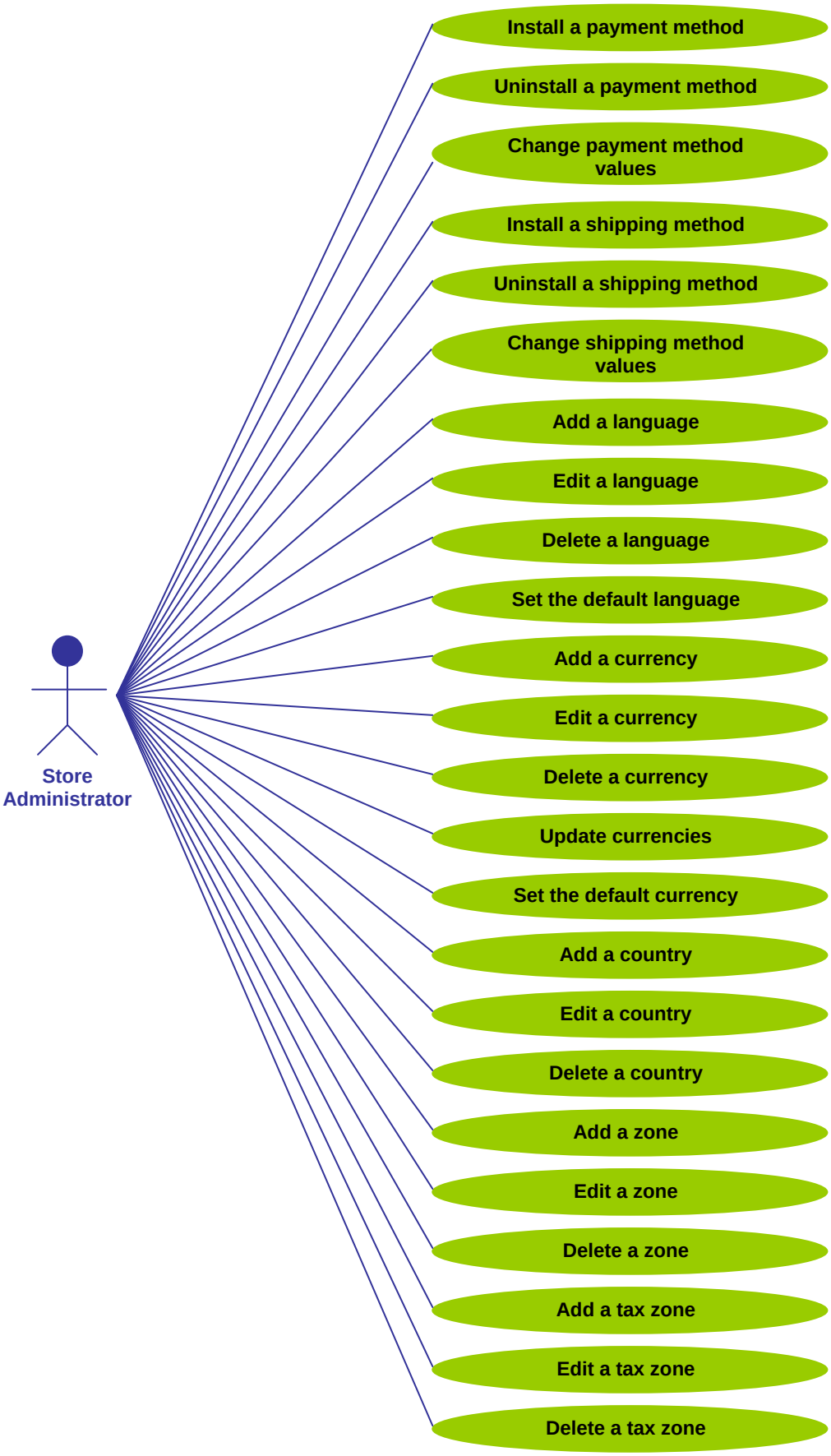
Firstly, we show an overview use cases diagram which gives a general view of the most important functionalities of the system. *Include* relationships are not shown for clarity.

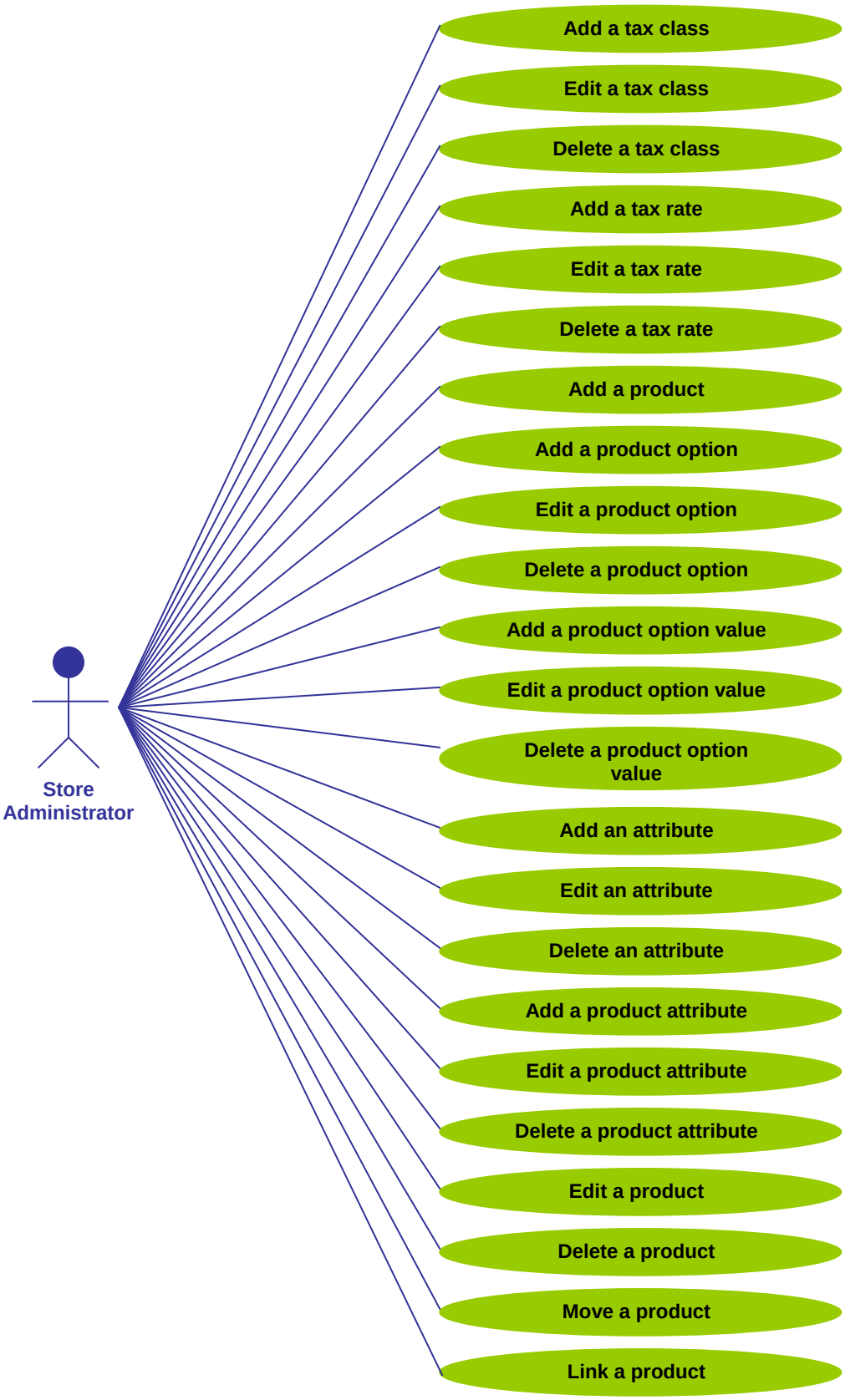
Afterwards, we specify each use case textually, as an interaction of activities between the primary actors and the system.

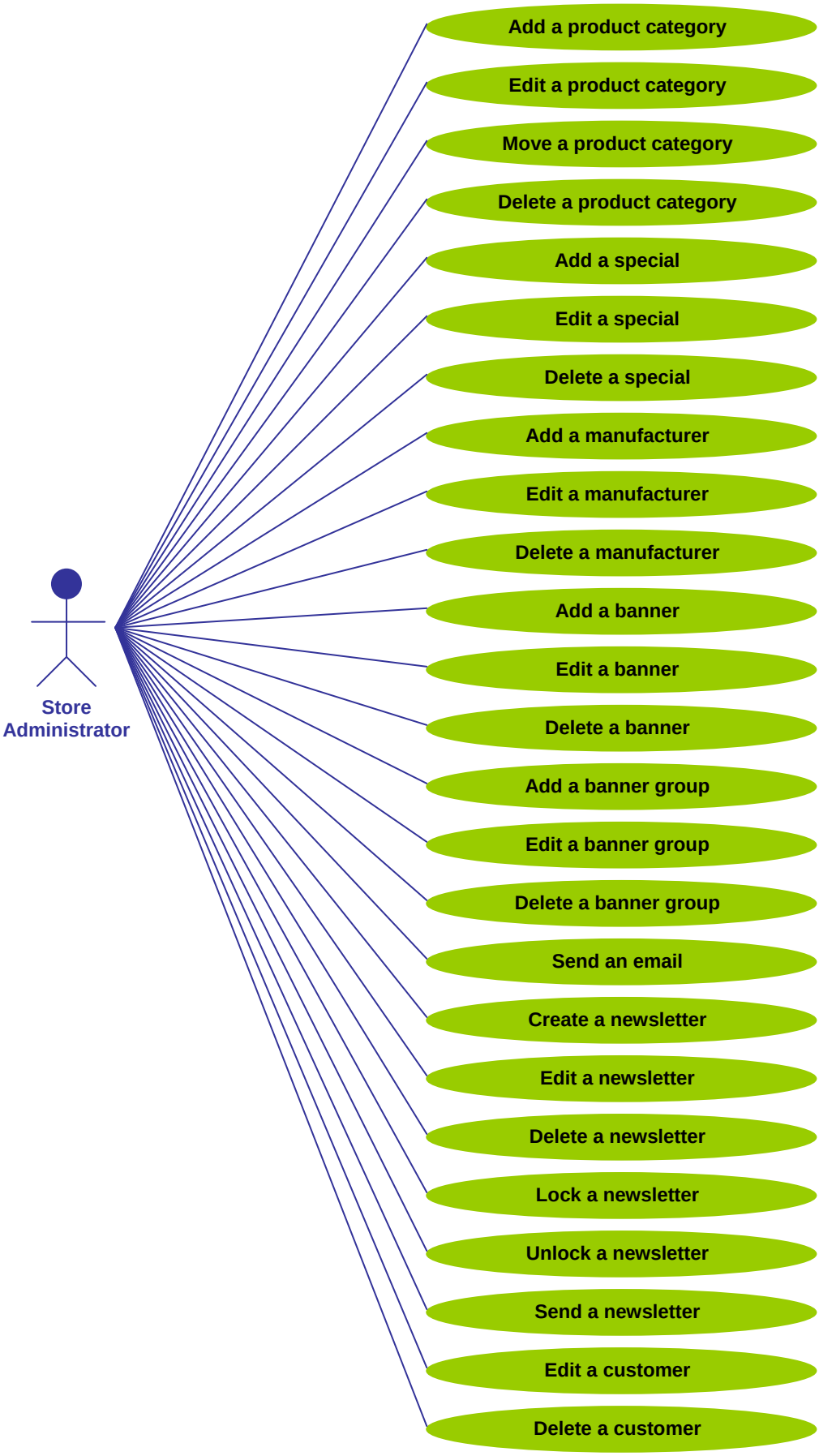
Use cases specification contains the mapping of use cases with the most important events of the system using textual references. Events of the system are presented alphabetically, in order to improve search. Each event is represented by an UML diagram, and its effect and constraints are specified using OCL operations.

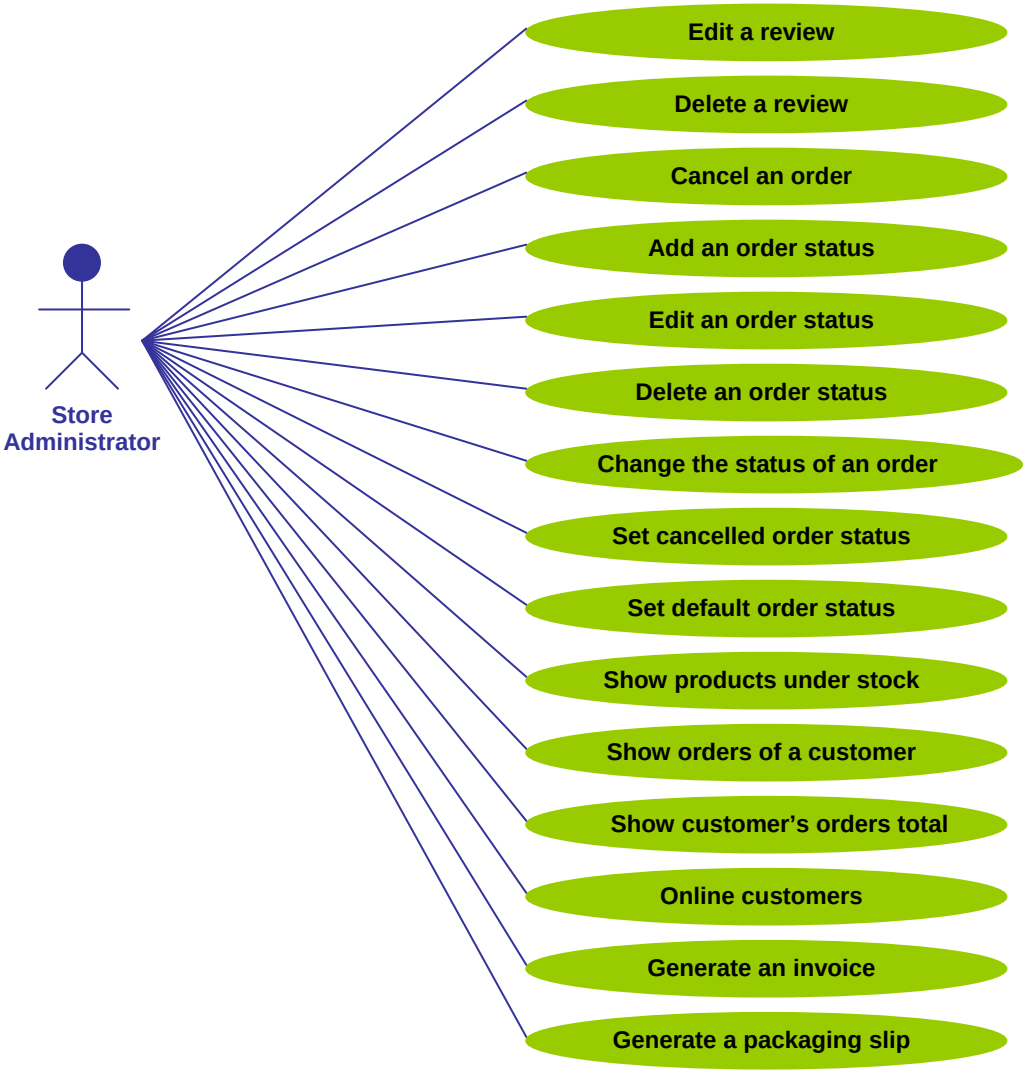
3.2 USE CASE DIAGRAM

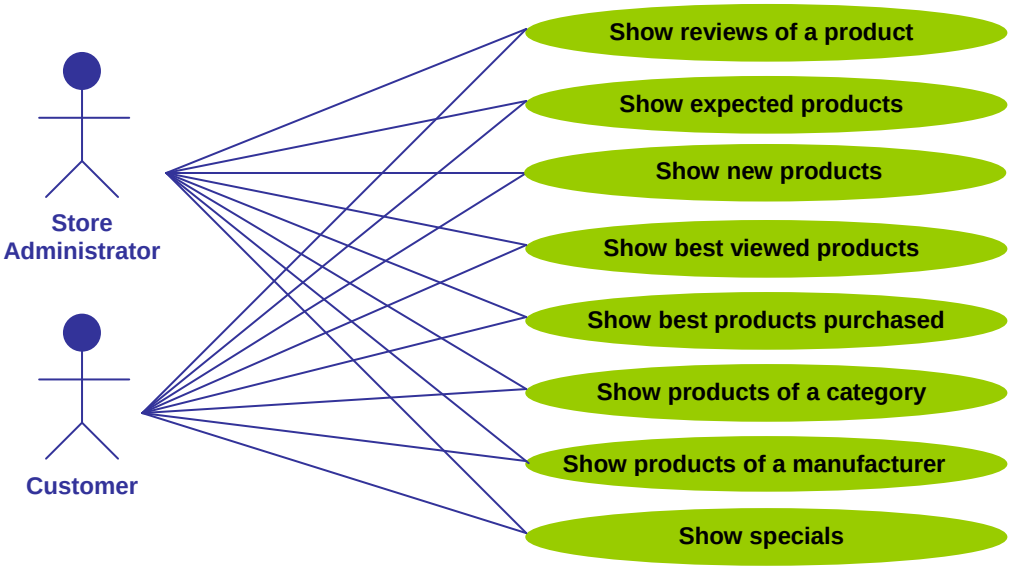


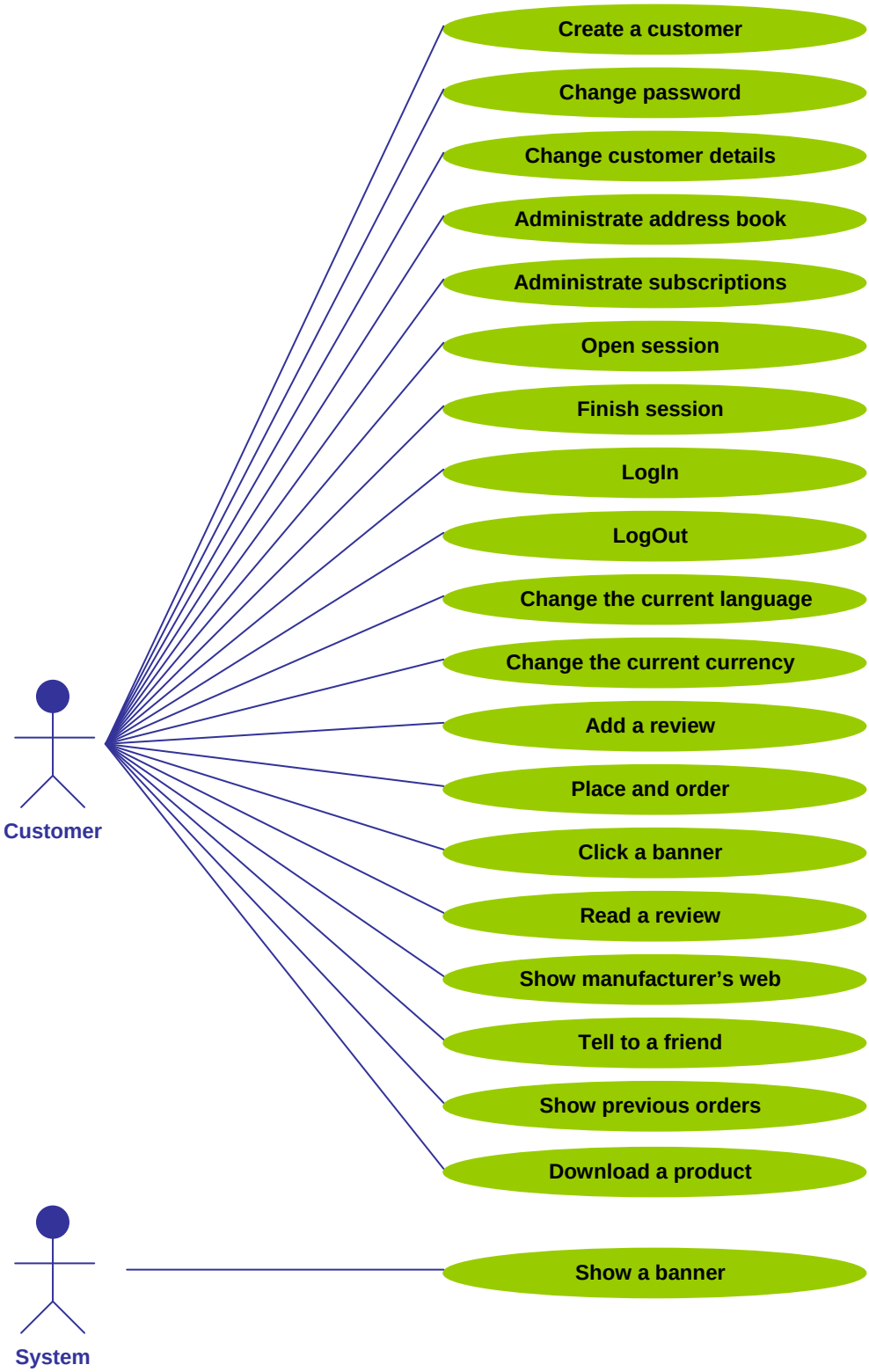

Store Administrator
Install a payment method
Uninstall a payment method
Change payment method values
Install a shipping method
Uninstall a shipping method
Change shipping method values
Add a language
Edit a language
Delete a language
Set the default language
Add a currency
Edit a currency
Delete a currency
Update currencies
Set the default currency
Add a country
Edit a country
Delete a country
Add a zone
Edit a zone
Delete a zone
Add a tax zone
Edit a tax zone
Delete a tax zone


Store Administrator
Add a tax class
Edit a tax class
Delete a tax class
Add a tax rate
Edit a tax rate
Delete a tax rate
Add a product
Add a product option
Edit a product option
Delete a product option
Add a product option value
Edit a product option value
Delete a product option value
Add an attribute
Edit an attribute
Delete an attribute
Add a product attribute
Edit a product attribute
Delete a product attribute
Edit a product
Delete a product
Move a product
Link a product


Store Administrator
Add a product category
Edit a product category
Move a product category
Delete a product category
Add a special
Edit a special
Delete a special
Add a manufacturer
Edit a manufacturer
Delete a manufacturer
Add a banner
Edit a banner
Delete a banner
Add a banner group
Edit a banner group
Delete a banner group
Send an email
Create a newsletter
Edit a newsletter
Delete a newsletter
Lock a newsletter
Unlock a newsletter
Send a newsletter
Edit a customer
Delete a customer


Edit a review
Delete a review
Cancel an order
Add an order status
Edit an order status
Delete an order status
Change the status of an order
Set cancelled order status
Set default order status
Show products under stock
Show orders of a customer
Show customer's orders total
Online customers
Generate an invoice
Generate a packaging slip
Store Administrator


Show reviews of a product
Show expected products
Show new products
Show best viewed products
Show best products purchased
Show products of a category
Show products of a manufacturer
Show specials
Store Administrator
Customer


Create a customer
Change password
Change customer details
Administrate address book
Administrate subscriptions
Open session
Finish session
LogIn
LogOut
Change the current language
Change the current currency
Add a review
Place and order
Click a banner
Read a review
Show manufacturer's web
Tell to a friend
Show previous orders
Download a product
Customer
Show a banner
System

3.3 USE CASE SPECIFICATION

Use case

Change store data

Primary Actor: System administrator

Precondition: None.

Trigger: The system administrator wants to change the initial values of the store data.

Main Success Scenario:

1. The system displays the current values of the store data.
2. The system administrator provides a new value for one of the store attributes:
   - [→*MameChange*]
   - [→*OwnerChange*]
   - [→*EMailAddressChange*]
   - [→*EMailFromChange*]
   - [→*ExpectedSortOrderChange*]
   - [→*ExpectedSortFieldChange*]
   - [→*SendExtraOrderChange*]
   - [→*DisplayCartAfterAddingProductChange*]
   - [→*AllowGuestToTellAFriendChange*]
   - [→*DefaultSearchOperatorChange*]
   - [→*StoreAddressAndPhoneChange*]
   - [→*TaxDecimalPlacesChange*]
   - [→*DisplayPricesWithTaxChange*]
   - [→*SwitchToDefaultLanguageCurrencyChange*]
   - [→*CountryChange*]
   - [→*ZoneChange*]
3. The system validates that the value is correct.
4. The system saves the new value.
5. The system displays the new values of the store data.

   The system administrator repeats steps 2-5 until he is done.

Use case

Assign minimum values

Primary Actor: System administrator

Precondition: None.

Trigger: The system administrator wants to change the minimum values of some attributes.

Main Success Scenario:

1. The system displays the current minimum values.
2. The system administrator provides a new value for one of the minimum values:
   - [→*FirstNameMinimumChange*]
   - [→*LastNameMinimumChange*]
   - [→*DateOfBirthMinimumChange*]
   - [→*EMailAddressMinimumChange*]
   - [→*StreetAddressMinimumChange*]
   - [→*CompanyNameMinimumChange*]
   - [→*PostCodeMinimumChange*]
   - [→*CityMinimumChange*]
   - [→*StateMinimumChange*]
   - [→*TelephoneMinimumChange*]
   - [→*PasswordMinimumChange*]
   - [→*CreditCardOwnerNameMinimumChange*]
   - [→*CreditCardNumberMinimumChange*]
   - [→*ReviewTextMinimumChange*]
3. The system validates that the value is correct.
4. The system saves the new value.
5. The system displays the new current minimum values.

   The system administrator repeats steps 2-5 until he is done.

Use case

Assign maximum values

Primary Actor: System administrator

Precondition: None.

Trigger: The system administrator wants to change the maximum number of address book entries permitted for each customer.

Main Success Scenario:

1. The system displays the current maximum number of address book entries for each customer.
2. The system administrator provides the new maximum value:
   [→*AddressBookEntriesMaximumChange*]
3. The system validates that the value is correct.
4. The system saves the new value.
5. The system displays the new current maximum value.

Use case

Change shown customer details

Primary Actor: System administrator

Precondition: None.

Trigger: The system administrator wants to change whether some customer attributes are shown.

Main Success Scenario:

1. The system displays the current values of customer details configuration (shown or not shown).
2. The system administrator provides the new value for one of the customer details:
   [→*GenderCustomerDetailChange*]
   [→*DateOfBirthCustomerDetailChange*]
   [→*CompanyCustomerDetailChange*]
   [→*SuburbCustomerDetailChange*]
   [→*StateCustomerDetailChange*]
3. The system validates that the value is correct.
4. The system saves the new value.
5. The system displays the new current values of customer details configuration.

   The system administrator repeats steps 2-5 until he is done.

Use case

Assign shipping and packaging configuration values

Primary Actor: System administrator

Precondition: None.

Trigger: The system administrator wants to change the shipping and packaging configuration values.

Main Success Scenario:

1. The system displays the current shipping and packaging configuration values.
2. The system administrator provides the new value for one of the shipping and packaging configurable options:

   [→*PostCodeShippingConfigurationChange*]

   [→*MaximumPackageWeightShippingConfigurationChange*]

   [→*TypicalPackageTareWeightShippingConfigurationChange*]

   [→*PercentageIncreaseForLargerPackagesShippingConfigurationChange*]

   [→*CountryShippingConfigurationChange*]
3. The system validates that the value is correct.
4. The system saves the new value.
5. The system displays the new current shipping and packaging configuration values.

   The system administrator repeats steps 2-5 until he is done.

Use case

Change download configuration values

Primary Actor: System administrator

Precondition: None.

Trigger: The system administrator wants to change the download configuration values.

Main Success Scenario:

1. The system displays the current download configuration values.
2. The system administrator provides the new value for one of the download configuration options:

   [→*EnableDownloadConfigurationChange*]

   [→*DaysExpiryDelayDownloadConfigurationChange*]

   [→*MaximumNumberDownloadConfigurationChange*]
3. The system validates that the value is correct.
4. The system saves the new value.
5. The system displays the new current download configuration values.

   The system administrator repeats steps 2-5 until he is done.

Use case

Change stock configuration values

Primary Actor: System administrator

Precondition: None.

Trigger: The system administrator wants to change the stock configuration values.

Main Success Scenario:

1. The system displays the current stock configuration values.
2. The system administrator provides the new value for one of the stock configuration options:
   [→*CheckLevelStockConfigurationChange*]
   [→*SubstractStockConfigurationChange*]
   [→*AllowCheckoutStockConfigurationChange*]
   [→*ReorderLevelStockConfigurationChange*]
3. The system validates that the value is correct.
4. The system saves the new value.
5. The system displays the new current stock configuration values.

   The system administrator repeats steps 2-5 until he is done.

Use case

Install a payment method

Primary Actor: Store administrator

Precondition: The payment method is not installed yet.

Trigger: The store administrator wants to install a payment method.

Main Success Scenario:

1. The system shows all the available payment methods and which of they are installed.
2. The store administrator selects a non installed payment method.
3. The store administrator provides the data of the payment method:
   [→*InstallAuthorizeNetPaymentMethod*]
   [→*InstallCreditCardPaymentMethod*]
   [→*InstallCashOnDeliveryPaymentMethod*]
   [→*InstallIPaymentPaymentMethod*]
   [→*InstallCheckMoneyPaymentMethod*]
   [→*InstallNochexPaymentMethod*]
   [→*InstallPayPalPaymentMethod*]
   [→*InstallTwoCheckOutPaymentMethod*]
   [→*InstallPSiGatePaymentMethod*]
   [→*InstallSECPaymentMethod*]
4. The system validates that the data is correct.
5. The system uninstalls the new payment method.

Use case

Uninstall a payment method

Primary Actor: Store administrator

Precondition: The payment method is installed.

Trigger: The store administrator wants to uninstall a payment method.

Main Success Scenario:

1. The system shows all the payment methods and which of they are installed.
2. The store administrator selects an installed payment method.
   - [→*UninstallAuthorizeNetPaymentMethod*]
   - [→*UninstallCreditCardPaymentMethod*]
   - [→*UninstallCashOnDeliveryPaymentMethod*]
   - [→*UninstallIPaymentPaymentMethod*]
   - [→*UninstallCheckMoneyPaymentMethod*]
   - [→*UninstallNochexPaymentMethod*]
   - [→*UninstallPayPalPaymentMethod*]
   - [→*UninstallTwoCheckOutPaymentMethod*]
   - [→*UninstallPSiGatePaymentMethod*]
   - [→*UninstallSECPaymentMethod*]
3. The system uninstalls the selected payment method.

Extensions:

2a. The payment method is used in an existing order:

2a1. The system warns the store administrator that the payment method is used in the information of existing orders and that is only possible to disable the payment method.

2a2. The system changes the status of the payment method to disabled:

[→*StatusPaymentMethodChange*]

2a3. The use case ends.

Use case

Change payment method values

Primary Actor: System administrator

Precondition: The payment method is installed.

Trigger: The system administrator wants to change the configuration values of an installed payment method.

Main Success Scenario:

1. The system displays the installed payment methods.
2. The customer selects an installed payment method.
3. The system displays the current values of the payment method.
4. The system administrator provides the new values for the configurable attributes of the payment method:
   - [→*EditAuthorizeNetPaymentMethod*]
   - [→*EditCreditCardPaymentMethod*]
   - [→*EditCashOnDeliveryPaymentMethod*]
   - [→*EditIPaymentPaymentMethod*]
   - [→*EditCheckMoneyPaymentMethod*]
   - [→*EditNochexPaymentMethod*]
   - [→*EditPayPalPaymentMethod*]
   - [→*EditTwoCheckOutPaymentMethod*]
   - [→*EditPSiGatePaymentMethod*]
   - [→*EditSECPaymentMethod*]
5. The system validates that the new values are correct.
6. The system saves the new values.
7. The system displays the new values of the payment method.

Use case

Install a shipping method

Primary Actor: Store administrator

Precondition: The shipping method is not installed yet.

Trigger: The store administrator wants to install a shipping method.

Main Success Scenario:

1. The system shows all the available shipping methods and which of they are installed.
2. The store administrator selects a non installed shipping method.
3. The store administrator provides the data of the shipping method.
   - [→*InstallZoneRatesShippingMethod*]
   - [→*InstallFlatRateShippingMethod*]
   - [→*InstallPerItemShippingMethod*]
   - [→*InstallTableRateShippingMethod*]
   - [→*InstallUSPostalServiceShippingMethod*]
4. The system validates that the data is correct.
5. The system creates an instance of the new shipping method.

Use case

Uninstall a shipping method

Primary Actor: Store administrator

Precondition: The shipping method is installed.

Trigger: The store administrator wants to uninstall a shipping method.

Main Success Scenario:

1. The system shows all the available shipping methods and which of they are installed.
2. The store administrator selects an installed shipping method.
   [→*UninstallZoneRatesShippingMethod*]
   [→*UnistallFlatRateShippingMethod*]
   [→*UninstallPerItemShippingMethod*]
   [→*UninstallTableRateShippingMethod*]
   [→*UninstallUSPostalServiceShippingMethod*]
3. The system deletes the instance of the selected shipping method.

Extensions:

2a. The shipping method is the shipping method used in an existing order:

2a1. The system warns the store administrator that the shipping method is used in the information of existing orders and that is only possible to disable the shipping method.

2a2. The system changes the *enabled* attribute of the shipping method to false:

[→*StatusShippingMethodChange*]

2a3. The use case ends.

Use case

Change shipping method values

Primary Actor: System administrator

Precondition: The shipping method is installed.

Trigger: The system administrator wants to change the configuration values of an installed shipping method.

Main Success Scenario:

1. The system displays the installed shipping methods.
2. The customer selects an installed shipping method.
3. The system displays the current values of the selected shipping method.

4. The system administrator provides the new values for the configurable attributes of the shipping method:

   [→*EditZoneRatesShippingMethod*]

   [→*EditFlatRateShippingMethod*]

   [→*EditPerItemShippingMethod*]

   [→*EditTableRateShippingMethod*]

   [→*EditUSPostalServiceShippingMethod*]

5. The system validates that the new values are correct.
6. The system saves the new values.
7. The system displays the new values of the shipping method.

Use case

Add a language

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a new language.

Main Success Scenario:

1. The store administrator provides the details of the new language:

   [→*NewLanguage*]

2. The system validates that the data is correct.
3. The system saves the new language.

Use case

Edit a language

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a language.

Main Success Scenario:

1. The store administrator selects the language to be edited.
2. The store administrator provides the new details of the selected language:

   [→*EditLanguage*]

3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete a language

Primary Actor: Store administrator

Precondition: There are at least two languages.

Trigger: The store administrator wants to delete a language.

Main Success Scenario:

1. The store administrator selects the language to be deleted.
2. The store administrator confirms that he wants to delete the language:
   [→*DeleteLanguage*]
3. The system deletes the language.

Extensions:

2a. The deleted language is the default language of the store.
   2a1. The system sets any of the available languages as the default language:
      [→*SetDefaultLanguage*]

2b. The deleted language is the current language of any active session.
   2b1. The system sets any of the available languages as the current language:
      [→*SetCurrentLanguage*]

Use case

Set the default language

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to change the default language.

Main Success Scenario:

1. The store administrator selects the language which will become the default language.
2. The system updates the default language:
   [→*SetDefaultLanguage*]

Use case

Add a currency

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a new currency.

Main Success Scenario:

1. The store administrator provides the details of the new currency:
   [→*NewCurrency*]
2. The system validates that the data is correct.
3. The system saves the new currency.

Use case

Edit a currency

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a currency.

Main Success Scenario:

1. The store administrator selects the currency to be edited.
2. The store administrator provides the new details of the selected currency:
   [→*EditCurrency*]
3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete a currency

Primary Actor: Store administrator

Precondition: There are at least two currencies.

Trigger: The store administrator wants to delete a currency.

Main Success Scenario:

1. The store administrator selects the currency to be deleted.

2. The store administrator confirms that he wants to delete the currency:

[→*DeleteCurrency*]

3. The system deletes the currency.

Extensions:

2a. The deleted currency was the default currency.

2a1. The system sets any of the available currencies as the default currency:

[→*SetDefaultCurrency*]

2b. The deleted currency is the current currency of an active session.

2b1. The system sets any of the available currencies as the current currency:

[→*SetCurrentCurrency*]

2c. The currency is the currency of an order:

2c1. The system changes the status of the currency to disable.

[→*CurrencyStatusChange*]

2c2. The use case ends.

Use case

Update currencies

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to update automatically via Internet the change values for currencies.

Main Success Scenario:

1. The system connects to the change information server.
2. The value change is automatically updated for all the currencies:

[→*UpdateCurrencyValueChange*]

Use case

Set the default currency

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to change the default currency.

Main Success Scenario:

1. The store administrator selects the currency which will become the default currency.
2. The system updates the default currency:

   [→*SetDefaultCurrency*]

Use case

Add a country

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a country.

Main Success Scenario:

1. The store administrator provides the details of the new country:

   [→*NewCountry*]
2. The system validates that the data is correct.
3. The system saves the new country.

Use case

Edit a country

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a country.

Main Success Scenario:

1. The store administrator selects the country to be edited.
2. The store administrator provides the new details of the selected country:

   [→*EditCountry*]
3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete a country

Primary Actor: Store administrator

Precondition: The country is not the location of any address.

Trigger: The store administrator wants to delete a country.

Main Success Scenario:

1. The store administrator selects the country to be deleted.
2. The system warns the store administrator of the number of zones which are part of the country to be deleted.
3. The store administrator confirms that he wants to delete the country and their zones:

   [→*DeleteCountry*]
4. The system deletes the country and their zones.

Use case

Add a zone

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a zone.

Main Success Scenario:

1. The store administrator provides the details of the new zone:

   [→*NewZone*]
2. The system validates that the data is correct.
3. The system saves the new zone.

Use case

Edit a zone

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a zone.

Main Success Scenario:

1. The store administrator selects the zone to be edited.
2. The store administrator provides the new details of the selected zone:

   [→*EditZone*]
3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete a zone

Primary Actor: Store administrator

Precondition: The zone is not the location of any address.

Trigger: The store administrator wants to delete a zone.

Main Success Scenario:

1. The store administrator selects the zone to be deleted.
2. The store administrator confirms that he wants to delete the zone:
   [→*DeleteZone*]
3. The system deletes the zone.

Use case

Add a tax zone

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a tax zone.

Main Success Scenario:

1. The store administrator provides the details of the new tax zone:
   [→*NewTaxZone*]
2. The system validates that the data is correct.
3. The system saves the new tax zone.

Use case

Edit a tax zone

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a tax zone.

Main Success Scenario:

1. The store administrator selects the tax zone to be edited.
2. The store administrator provides the new details of the selected tax zone:

[→*EditTaxZone*]

3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete a tax zone

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a tax zone.

Main Success Scenario:

1. The store administrator selects the tax zone to be deleted.
2. The store administrator confirms that he wants to delete the tax zone:

   [→*DeleteTaxZone*]

3. The system deletes the tax zone and all the associated tax rates.

Use case

Add a tax class

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a tax class.

Main Success Scenario:

1. The store administrator provides the details of the new tax class:

   [→*NewTaxClass*]

2. The system validates that the data is correct.
3. The system saves the new tax class.

Use case

Edit a tax class

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a tax class.

Main Success Scenario:

1. The store administrator selects the tax class to be edited.
2. The store administrator provides the new details of the selected tax class:

   [→*EditTaxClass*]
3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete a tax class

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a tax class.

Main Success Scenario:

1. The store administrator selects the tax class to be deleted.
2. The system informs the store administrator about how many products are associated to the deleted tax class.
3. The store administrator confirms that he wants to delete the tax class:

   [→*DeleteTaxClass*]
4. The system deletes the tax class and all the associated tax rates.

Extensions:

2a. The store administrator don't want to delete the tax class.

   2a1. The use case ends.

Use case

Add a tax rate

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a tax rate.

Main Success Scenario:

1. The store administrator provides the details of the new tax rate:

   [→*NewTaxRate*]

2. The system validates that the data is correct.
3. The system saves the new tax rate.

Use case

Edit a tax rate

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a tax rate.

Main Success Scenario:

1. The store administrator selects the tax rate to be edited.
2. The store administrator provides the new details of the selected tax rate:
   [→*EditTaxRate*]
3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete a tax rate

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a tax rate.

Main Success Scenario:

1. The store administrator selects the tax rate to be deleted.
2. The store administrator confirms that he wants to delete the tax rate:
   [→*DeleteTaxRate*]
3. The system deletes the tax rate.

Use case

Add a product

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a product to the store catalog.

Main Success Scenario:

1. The store administrator selects the product category.
2. The store administrator provides the product data:
   [→*NewProduct*]
3. The system validates that the data is correct.
4. The system saves the new product.
5. The store administrator provides a product attribute:
   [→*NewProductAttribute*]
6. The system validates that the product attribute is correct.
7. The system saves the new product attribute.

   The store administrator repeats steps 5-7 until he is done.

Extensions:

5a. The product does not have product attributes:
   5a1. The use case ends.

5b. The product option is new:
   5b1. Add a product option.

5c. The product option value is new:
   5c1. Add a product option value.

Use case

Add a product option

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a product option to the store catalog.

Main Success Scenario:

1. The store administrator provides the product option data:
   [→*NewProductOption*]
2. The system validates that the data is correct.
3. The system saves the new product option.

Use case

Edit a product option

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a product option.

Main Success Scenario:

1. The store administrator selects the product option to be edited.
2. The store administrator provides the new details of the selected product option:
   [→*EditProductOption*]
3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete a product option

Primary Actor: Store administrator

Precondition: The product option has not products or values linked to it.

Trigger: The store administrator wants to delete a product option.

Main Success Scenario:

1. The store administrator selects the product option to be deleted.
2. The system asks for the confirmation of the store administrator.
3. The store administrator confirms that he wants to delete the product option:
   [→*DeleteProductOption*]
4. The system deletes the product option.

Use case

Add a product option value

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a value to a product option.

Main Success Scenario:

1. The store administrator selects the product option.
2. The store administrator provides the product option value data:
   [→*NewProductOptionValue*]
3. The system validates that the data is correct.
4. The system saves the new product option value.

Use case

Edit a product option value

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a product option value.

Main Success Scenario:

1. The store administrator selects the product option value to be edited.
2. The store administrator provides the new details of the selected product option value:
   [→*EditProductOptionValue*]
3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete a product option value

Primary Actor: Store administrator

Precondition: The product option value has not products linked to it.

Trigger: The store administrator wants to delete a product option value.

Main Success Scenario:

1. The store administrator selects the product option value to delete.
2. The system asks for the confirmation of the store administrator.
3. The store administrator confirms that he wants to delete the product option value:
   [→*DeleteProductOptionValue*]
4. The system deletes the product option value.

Use case

Add a product attribute

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to assign an attribute to a product.

Main Success Scenario:

1. The store administrator selects the product.
2. The store administrator provides the attribute and the product attribute data (increment and sign):
   [→*NewProductAttribute*]
   [→*NewDownloadableProductAttribute*]
3. The system validates that the data is correct.
4. The system saves the new product attribute.

Extensions:

2a. The product option is new:
   2a1. Add a product option.
2b. The product option value is new:
   2b1. Add a product option value.

Use case

Edit a product attribute

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a product attribute.

Main Success Scenario:

1. The store administrator selects the product attribute to be edited.
2. The store administrator provides the new details for the product attribute:
   [→*AttributeChange*]
   [→*IncrementAndSignAttributeChange*]
   [→*EditDownloadableAttribute*]
3. The system validates that the data is correct.
4. The system saves the changes.

The system repeats steps 2-4 until he is done.

Use case

Delete a product attribute

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a product attribute.

Main Success Scenario:

1. The store administrator selects the product attribute to be deleted.
2. The system asks for the confirmation of the store administrator.
3. The store administrator confirms that he wants to delete the product attribute:
   [→*DeleteProductAttribute*]
4. The system deletes the product attribute.

Extensions:

3a. The product attribute is part of an existing order line:
   3a1. The system changes the status of the product attribute to disable.
      [→*ProductAttributeStatusChange*]
   3a2. The use case ends

Use case

Edit a product

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a product.

Main Success Scenario:

1. The store administrator selects the product to be edited.
2. The store administrator provides the new values for the attributes of the product:
   [→*EditProduct*]
3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete a product

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a product.

Main Success Scenario:

1. The store administrator selects the product to be deleted.
2. The system asks for the confirmation of the store administrator.
3. The store administrator confirms that he wants to delete the product:
   [→*DeleteProduct*]
4. The system deletes the product and their product attributes.

Extensions:

3a. The product is part of an order:
   3a1. The system changes the status of the product to disable.
      [→*ProductStatusChange*]
   3a2. The use case ends.

Use case

Move a product

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to change the category of a product.

Main Success Scenario:

1. The store administrator selects the product to be moved.
2. The store administrator indicates the new category of the selected product, if any:
   [→*MoveProduct*]
3. The system validates that the data is correct.
4. The system saves the new placement.

Use case

Link a product

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to link a product to another category.

Main Success Scenario:

1. The store administrator selects the product to be linked.
2. The store administrator indicates the new category of the selected product, if any :

   [→*LinkProduct*]
3. The system links the product.

Use case

Add a product category

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a category.

Main Success Scenario:

1. The store administrator provides the details of the new product category, including its parent category, if any:

   [→*NewCategory*]
2. The system validates that the data is correct.
3. The system saves the new category.

Use case

Edit a product category

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a category.

Main Success Scenario:

1. The store administrator selects the category to be edited.

2. The store administrator provides the new details of the selected category:

   [→*EditCategory*]

3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Move a product category

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to change the placement of a category in the category hierarchy.

Main Success Scenario:

1. The store administrator selects the category to be moved.
2. The store administrator indicates the new parent category, if any:

   [→*MoveCategory*]

3. The system validates that the data is correct.
4. The system saves the new placement.

Use case

Delete a product category

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a category.

Main Success Scenario:

1. The store administrator selects the category to be deleted.
2. The system warns the store administrator of the number of subcategories and products linked to the category to be deleted.
3. The store administrator confirms that he wants to delete the category:

   [→*DeleteCategory*]

4. The system deletes the selected category and its subcategories. The products linked to the deleted category or its subcategories are removed from the system if they do not participate in any orders. The system changes the status of the products which participate in orders to out of stock.

Use case

Add a special

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a special.

Main Success Scenario:

1. The store administrator selects the product which will be offered in a special price.
2. The store administrator provides the details of the special:
   [→*NewSpecial*]
3. The system validates that the data is correct.
4. The system saves the new special.

Use case

Edit a special

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a special.

Main Success Scenario:

1. The store administrator selects the special to be edited.
2. The store administrator provides the new details of the selected special:
   [→*EditSpecial*]
3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete a special

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a special.

Main Success Scenario:

1. The store administrator selects the special to be deleted.
2. The system asks for the confirmation of the store administrator.
3. The store administrator confirms that he wants to delete the special:

   [→*DeleteSpecial*]
4. The system deletes the special.

Use case

Add a manufacturer

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a manufacturer.

Main Success Scenario:

1. The store administrator provides the details of the new manufacturer:

   [→*NewManufacturer*]
2. The system validates that the data is correct.
3. The system saves the new manufacturer.

Use case

Edit a manufacturer

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a manufacturer.

Main Success Scenario:

1. The store administrator selects the manufacturer to be edited.
2. The store administrator provides the new details of the selected manufacturer:

   [→*EditManufacturer*]
3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete a manufacturer

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a manufacturer.

Main Success Scenario:

1. The store administrator selects the manufacturer to delete.
2. The system warns the store administrator of the number of products linked to the manufacturer to be deleted.
3. The store administrator confirms that he wants to delete the manufacturer:

   [→*DeleteManufacturer*]
4. The system deletes the manufacturer and, if requested, changes the status of the products manufactured by it to out of stock.

Use case

Add a banner

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a new banner.

Main Success Scenario:

1. The store administrator provides the details of the new banner:

   [→*NewBanner*]
2. The system validates that the data is correct.
3. The system saves the new banner.

Use case

Edit a banner

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a banner.

Main Success Scenario:

1. The store administrator selects the banner to be edited.
2. The store administrator provides the new details of the selected banner:

   [→*EditBanner*]
3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete a banner

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a banner.

Main Success Scenario:

1. The store administrator selects the banner to be deleted.
2. The store administrator confirms that he wants to delete the banner:

   [→*DeleteBanner*]
3. The system deletes the banner.

Use case

Add a banner group

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a new banner group.

Main Success Scenario:

1. The store administrator provides the details of the new banner group:

   [→*NewBannerGroup*]
2. The system validates that the data is correct.
3. The system saves the new banner.

Use case

Edit a banner group

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a banner group.

Main Success Scenario:

1. The store administrator selects the banner group to be edited.
2. The store administrator provides the new details of the selected banner group:
   [→*EditBannerGroup*]
3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete a banner group

Primary Actor: Store administrator

Precondition: The banner group doesn't contain any banners.

Trigger: The store administrator wants to delete a banner.

Main Success Scenario:

1. The store administrator selects the banner group to be deleted.
2. The store administrator confirms that he wants to delete the banner group:
   [→*DeleteBannerGroup*]
3. The system deletes the banner.

Use case

Send an email

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to send an email to customers.

Main Success Scenario:

1. The store administrator selects the addressee customer, or one of the predefined set of addressee customers (all the customers or all the newsletter subscribers).
2. The store administrator specifies the sender address.
3. The store administrator provides the subject and the message.
4. The store administrator confirms that he wants to send the email.
5. The system sends the email.

Use case

Create a newsletter

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to create a new newsletter.

Main Success Scenario:

1. The store administrator selects the type of the newsletter (newsletter or product nofitification).
2. The store administrator provides the title and the content of the newsletter:

   [→*NewNewsletter*]

   [→*NewProductNotification*]
3. The system validates that the data is correct.
4. The system saves the newsletter.

Use case

Edit a newsletter

Primary Actor: Store administrator

Precondition: The newsletter is unlocked.

Trigger: The store administrator wants to edit a newsletter.

Main Success Scenario:

1. The store administrator selects the newsletter to be edited.
2. The store administrator provides the new details of the selected newsletter:

   [→*EditNewsletter*]

   [→*EditProductNotification*]
3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete a newsletter

Primary Actor: Store administrator

Precondition: The newsletter is unlocked.

Trigger: The store administrator wants to delete a newsletter.

Main Success Scenario:

1. The store administrator selects the newsletter to be deleted.
2. The store administrator confirms that he wants to delete the newsletter:
   [→*DeleteNewsletter*]
3. The system deletes the newsletter.

Use case

Lock a newsletter

Primary Actor: Store administrator

Precondition: The newsletter is unlocked.

Trigger: The store administrator wants to indicate to the other administrators that a newsletter is pending to be delivered.

Main Success Scenario:

1. The store administrator selects the newsletter to be locked.
   [→*LockNewsletter*]
2. The system saves the change.

Use case

Unlock a newsletter

Primary Actor: Store administrator

Precondition: The newsletter is locked.

Trigger: The store administrator wants to indicate to the other administrators that a newsletter ceases to be locked.

Main Success Scenario:

1. The store administrator selects the newsletter to be unlocked.
   [→*UnlockNewsletter*]

2. The system saves the change.

Use case

Send a newsletter

Primary Actor: Store administrator

Precondition: The newsletter is locked.

Trigger: The store administrator wants to send a newsletter.

Main Success Scenario:

1. The store administrator selects the newsletter which will be sent.
2. The system sends the newsletter to all the newsletter subscribers.
3. The system saves that the newsletter has been sent.

   [→*SendNewsletter*]

Extensions:

2a. The newsletter is a product notification.

   2a1. The store administrator selects which products are implied in the notification.

   2a2. The systems sends the newsletter to customers who are subscribed to some of the notificated products.

   2a3. The use case continues at step 3.

Use case

Create a customer

Primary Actor: Customer

Precondition: None.

Trigger: A customer wants to open an account in the store.

Main Success Scenario:

1. The customer provides the required customer data:

   [→*NewCustomer*]
2. The system validates the customer data.
3. The system saves the new account.

Use case

Change password

Primary Actor: Customer

Precondition: The customer is logged in.

Trigger: A customer wants to change his password.

Main Success Scenario:

1. The customer provides the old password.
2. The customer provides the new password twice.
   [→*PasswordChange*]
3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Change customer details

Primary Actor: Customer

Precondition: The customer is logged in.

Trigger: A customer wants to change its customer details.

Main Success Scenario:

1. The customer provides the new customer details.
   [→*EditCustomerDetails*]
2. The system validates that the data is correct.
3. The system saves the changes.

Use case

Administrate address book

Primary Actor: Customer

Precondition: The customer is logged in and the number of addresses is less than the maximum number of address entries permitted.

Trigger: A customer wants to view or change the address book.

Main Success Scenario:

1. The system displays the current address book entries of the customer.
2. The customer selects an address book entry to be edited :
   [→*EditCustomerAddress*]
3. The system validates that the data is correct.
4. The system saves the changes and displays the new address book.

   The customer repeats steps 1-4 until he is done.

Extensions:

2a. The customer doesn't want to change the address book:
   2a1. The use case ends.

2b. The customer wants to add a new address book entry:
   2b1. The customer provides the required data:
      [→*NewCustomerAddress*]
   2b2. The use case continues at step 3.

2c. The customer wants to delete an address book entry:
   2c1. The customer selects the address book entry:
      [→*DeleteCustomerAddress*]
   2c2. The use case continues at step 3.

2d. The customer wants to change the default address book entry:
   2d1. The customer selects the new default address book entry:
      [→*PrimaryCustomerAddressChange*]
   2d2. The use case continues at step 3.

Use case

Administrate subscriptions

Primary Actor: Customer

Precondition: The customer is logged in.

Trigger: A customer wants to view or change their product notification subscriptions.

Main Success Scenario:

1. The system displays the details of the current product notification subscriptions of the customer.
2. The customer adds a new product subscription:
   [→*NewProductNotificationSubscription*]
3. The system validates that the data is correct.
4. The system saves the changes and displays the new product notification subscriptions.

   The customer repeats steps 1-4 until he is done.

Extensions:

2a. The customer doesn't want to change their product notification subscriptions:

2a1. The use case ends.

2b. The customer wants to be subscribed or unsubscribed to all product notifications:

[→ *EditGlobalNotifications*]

2c. The customer wants to delete a product notification subscription:

2c1. The customer selects the product:

[→*DeleteProductNotificationSubscription*]

2c2. The use case continues at step 3.

Use case

Edit a customer

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a customer.

Main Success Scenario:

1. The store administrator selects the customer to be edited.
2. The store administrator provides the new details of the selected customer:

   [→*EditCustomer*]
3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete a customer

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a customer.

Main Success Scenario:

1. The store administrator selects the customer to be deleted.
2. The system asks for the confirmation of the store administrator.
3. The store administrator confirms that he wants to delete the customer:

   [→*DeleteCustomer*]
4. The system deletes the customer and their addresses, reviews, notification subscriptions and shopping carts.

Extensions:

3a. The customer has orders:

3a1. The system changes the status of the customer to disable.

[→*CustomerStatusChange*]

3a2. The system deletes customer's addresses, reviews, notification subscriptions and shopping carts.

3a3. The use case ends.

Use case

Open session

Primary Actor: Customer

Precondition: None.

Trigger: A customer starts using the system.

Main Success Scenario:

1. The system creates an anonymous session :

[→*NewSession*]

Use case

Finish session

Primary Actor: Customer

Precondition: None.

Trigger: A customer finishes using the system.

Main Success Scenario:

1. The system deletes the current session.

[→*DeleteSession*]

Extensions:

1a. The customer is logged in and the session has a non empty shopping cart.

1a1. The shopping cart is saved.

Use case

Log in

Primary Actor: Customer

Precondition: The customer is not logged in yet.

Trigger: A customer logs in the system.

Main Success Scenario:

1. The customer introduces their identification data.
2. The system validates the identification data.
3. The customer becomes the owner of the current session.

   [→*LogIn*]

Extensions:

3a. The customer has a shopping cart from a previous session.

   3a1. The previous shopping cart is restored.

      [→*RestorePreviousShoppingCart*]

Use case

LogOut

Primary Actor: Customer

Precondition: The customer is logged in.

Trigger: A customer logs out from the system.

Main Success Scenario:

1. The current session becomes anonymous.

   [→*LogOut*]

Extensions:

1a. The customer has a non empty shopping cart.

   1a1. The shopping cart is saved.

Use case

Change the current language

Primary Actor: Customer

Precondition: None.

Trigger: A customer wants to change the current language of the session.

Main Success Scenario:

1. The store administrator selects the language which will become the current language.
2. The system updates the current language.
   [→*SetCurrentLanguage*]

Use case

Change the current currency

Primary Actor: Customer

Precondition: None.

Trigger: A customer wants to change the current currency of the session.

Main Success Scenario:

1. The store administrator selects the currency which will become the current currency.
2. The system updates the current currency.
   [→*SetCurrentCurrency*]

Use case

Add a review

Primary Actor: Customer

Precondition: None.

Trigger: A customer wants to write a review of a product.

Main Success Scenario:

1. The customer selects a product.
2. The customer provides the content and the rate of the review:
   [→*NewReview*]
3. The system validates that the data is correct.

4. The system saves the review.

Extensions:

2a. The customer is not logged in:

2a1. The customer logs in:

[→*LogIn*]

2a2. The use case continues at step 2.

Use case

Edit a review

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit a review.

Main Success Scenario:

1. The store administrator selects the review to be edited.
2. The store administrator provides the modified text and the new rating of the selected review.

[→*EditReview*]

3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete a review

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to delete a review.

Main Success Scenario:

1. The store administrator selects the review to be deleted.
2. The system asks for the confirmation of the store administrator.
3. The store administrator confirms that he wants to delete the review:

[→*DeleteReview*]

4. The system deletes the review.

Use case

Place an order

Primary Actor: Customer

Precondition: None.

Trigger: A customer wants to place and order.

Main Success Scenario:

1. At any time before step 10 the customer logs in:

   [→*LogIn*]

The system adds the contents of the anonymous shopping cart to the customer shopping cart.

2. The system displays the contents of the shopping cart.
3. The customer browses the product catalog.

   [→*ReadProductInfo*]

4. The customer selects a product to buy:

   [→*AddProductToShoppingCart*]

5. The system adds the product to the shopping cart.
6. The system displays the contents of the shopping cart.
7. The customer changes the contents of the shopping cart:

   [→*UpdateShoppingCart*]

8. The system updates the shopping cart.
9. The system displays the contents of the updated shopping cart.

   The customer repeats steps 3,4 and 7 as necessary to build his order.

10. The customer checks out the order.
11. The system shows the shipping address and the available shipping methods.
12. The customer selects the preferred shipping method.
13. The system shows the billing address and the available payment methods.
14. The customer selects the preferred payment method.
15. The system displays a summary of the order.
16. The customer confirms the order:

    [→*OrderConfirmation*]

17. The system saves the order.
18. The system sends an email to the customer and to the store extra order emails with the information about the order.

Extensions:

1a. The customer is new:

1a1. Create customer.

5a. The configurable option *Display cart after adding a product* is disabled

The customer repeats steps 3 and 4 as necessary.

5a1. The customer continues with the checkout procedure at step 9.

16a. The customer wants to change the contents of the shopping cart:

16a1. The customer changes the contents of the shopping cart:

[→*UpdateShoppingCart*]

16a2. The customer continues with the checkout procedure at step 11.

11a, 16a. The customer wants to change the shipping address:

11a1. The system shows the know addresses of the customer.

11a2. The customer selects a different shipping address.

11a3. The customer continues with the checkout procedure at step 11.

13a, 16b. The customer wants to change the billing address:

13a1. The system shows the know addresses of the customer.

13a2. The customer selects a different billing address.

13a3. The customer continues with the checkout procedure at step 13.

16c. The customer wants to change the shipping method:

16c1. The customer selects the new shipping method.

16c2. The customer continues with the checkout procedure at step 13.

16d. The customer wants to change the payment method:

16d1. The customer selects the new payment method.

16d2. The customer continues with the checkout procedure at step 15.

11a2a,16a2a. The customer wants to define a new shipping address:

11a2a1. The customer gives the new address:

[→*NewCustomerAddress*]

11a2a2. The system saves the address.

11a2a3. The customer continues with the checkout procedure at step 11.

13a2a,16b2a. The customer wants to define a new billing address:

13a2a1. The customer gives the new address:

[→*NewCustomerAddress*]

13a2a2. The system saves the address.

13a2a3. The customer continues with the checkout procedure at step 13.

Use case

Cancel an order

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to cancel an order.

Main Success Scenario:

1. The store administrator selects the order to be cancelled.
2. The system asks for the confirmation of the store administrator.
3. The store administrator confirms that he wants to cancel the order:
   [→*CancelOrder*]
4. The system sets the order status to cancelled.

Use case

Add an order status

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to add a new order status.

Main Success Scenario:

1. The store administrator provides the details of the new order status:
   [→*NewOrderStatus*]
2. The system validates that the data is correct.
3. The system saves the new order status.

Use case

Edit an order status

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to edit an order status.

Main Success Scenario:

1. The store administrator selects the order status to be edited.
2. The store administrator provides the new details of the selected order status:
   [→*EditOrderStatus*]
3. The system validates that the data is correct.
4. The system saves the changes.

Use case

Delete an order status

Primary Actor: Store administrator

Precondition: The deleted order status is not the current status of any order.

Trigger: The store administrator wants to delete an order status.

Main Success Scenario:

1. The store administrator selects the order status to be deleted.
2. The store administrator confirms that he wants to delete the order status:
   [→*DeleteOrderStatus*]
3. The system deletes the order status.

Extensions:

2a. The order status has been an status of an order:
   2a1. The system changes the status of the order status to disable.
      [→*ProductStatusChange*]
   2a2. The use case ends.

Use case

Change the status of an order

Primary Actor: Store administrator

Precondition: None.

Trigger: The store administrator wants to change the status of an order.

Main Success Scenario:

1. The system shows the orders and their status.
2. The store administrator selects the order which will be edited.
3. The system shows the applicable order status.
4. The store administrator selects the new status.
   [→*UpdateOrderStatus*]
5. The system validates that the data is correct.
6. The system saves the changes.

Use case

Set cancelled order status

Primary Actor: Store administrator

Precondition: The order status is not yet the cancelled status.

Trigger: The store administrator wants to indicate to the system which order status is used to indicate that an order is cancelled.

Main Success Scenario:

1. The store administrator selects an order status.
2. The system register that the selected order status represents cancelled orders.
   [→*SetCancelledOrderStatus*]

Use case

Set default order status

Primary Actor: Store administrator

Precondition: The order status is not yet the default status.

Trigger: The store administrator wants to indicate to the system which order status is assign when an order is created.

Main Success Scenario:

1. The store administrator selects an order status.
2. The system register that the selected order status is the default order status.
   [→*SetDefaultOrderStatus*]

Use case

Show a banner

Primary Actor: System.

Precondition: None.

Trigger: The system shows a banner.

Main Success Scenario:

1. The system shows a banner.
   [→*ShowBanner*]

Use case

Click a banner

Primary Actor: Customer

Precondition: None.

Trigger: The customer clicks on a banner.

Main Success Scenario:

1. The customer clicks on a banner.
   [→*ClickBanner*]
2. The system redirects the online store to the banner's web page.

Use case

Read a review

Primary Actor: Customer

Precondition: None.

Trigger: A customer wants to read a review of a product.

Main Success Scenario:

1. The system shows a summary of the reviews of the product.
2. The customer selects a review.
   [→*ReadReview*]
3. The system shows the selected review.

Use case

Download a product

Primary Actor: Customer

Precondition: The customer is logged in.
The customer purchased the product.
Download is enabled by the system.
The download is not expired and the number of download has not been exceeded.

Trigger: A customer wants to download a purchased product.

Main Success Scenario:

1. The customer selects the purchased product to be downloaded.

   [→*ProductDownload*]

2. The system allows the customer downloading the product.

Use case

Show manufacturer's web

Primary Actor: Customer

Precondition: None.

Trigger: A customer wants to be redirected to the manufacturer's web page.

Main Success Scenario:

1. The customer selects a manufacturer.

   [→ClickManufacturer]

2. The customer is redirected to the the manufacturer's web page.

Use case

Show products under stock

Primary Actor: Store administrator.

Precondition: None.

Trigger: The store administrator wants to obtain which products have to be reordered.

Main Success Scenario:

1. The system shows the set of products under stock :

   [→*ShowUnderStockProducs*]

Use case

Show expected products

Primary Actor: Store administrator, Customer.

Precondition: None.

Trigger: The store administrator or the customer wants to obtain which products will be in stock soon.

Main Success Scenario:

1. The system shows the set of expected products:

   [→*ShowExpectedProducs*]

Use case

Show orders of a customer

Primary Actor: Store administrator.

Precondition: None.

Trigger: The store administrator wants to obtain the orders of a customer.

Main Success Scenario:

1. The system shows the list of customers.
2. The store administrator selects a customer.
3. The system shows the orders of the selected customer:

   [→*ShowOrdersOfCustomer*]

Use case

Show previous orders

Primary Actor: Customer.

Precondition: The customer is logged in.

Trigger: A Customer wants to visualize their orders.

Main Success Scenario:

1. The system shows the previous orders made by the customer:

   [→*ShowOrdersOfCustomer*]

Use case

Show best viewed products

Primary Actor: Store administrator, Customer.

Precondition: None.

Trigger: The store administrator or the customer wants to visualize the most viewed products.

Main Success Scenario:

1. The system shows the products in stock ordered by the number of times which has been visualized by the customers.

   [→*ShowBestViewedProducts*]

Use case

Show best products purchased

Primary Actor: Store administrator, Customer.

Precondition: None.

Trigger: The store administrator or the customer wants to visualize the most purchased products.

Main Success Scenario:

1. The system shows the products in stock ordered by the number of times which has been purchased by the customers.

   [→*ShowBestPurchasedProducts*]

Use case

Show customer's orders total

Primary Actor: Store administrator.

Precondition: None.

Trigger: The store administrator wants to visualize the total amount of money spent by each customer in the online store.

Main Success Scenario:

1. The system shows the customers and the total price of their orders.

   [→*ShowCustomersOrdersTotal*]

Use case

Online customers

Primary Actor: Store administrator.

Precondition: None.

Trigger: The store administrator wants to visualize the customers who are online.

Main Success Scenario:

1. The system shows the online customers.

[→*ShowOnlineCustomers*]

Use case

Show specials

Primary Actor: Store administrator, Customer.

Precondition: None.

Trigger: The store administrator or the customer wants to visualize products on offer.

Main Success Scenario:

1. The system shows the products on offer.

[→*ShowSpecials*]

Use case

Show products of a category

Primary Actor: Store administrator, customer.

Precondition: None.

Trigger: The store administrator or the customer wants to visualize the products contained in a category.

Main Success Scenario:

1. The store administrator or the customer selects a category.
2. The system shows the products of the selected category.

[→*ShowProductsOfCategory*]

Use case

Show products of a manufacturer

Primary Actor: Store administrator, customer.

Precondition: None.

Trigger: The store administrator or the customer wants to visualize the products produced by a manufacturer.

Main Success Scenario:

1. The store administrator or the customer selects a manufacturer.
2. The system shows the products manufactured by the selected manufacturer.
   [→*ShowProductsOfManufacturer*]

Use case

Show new products

Primary Actor: Customer.

Precondition: None.

Trigger: The customer wants to visualize the last launched products.

Main Success Scenario:

1. The system shows the last products on sale.
   [→*ShowNewProducts*]

Use case

Show reviews of a product

Primary Actor: Store administrator, customer.

Precondition: None.

Trigger: The store administrator or the customer wants to visualize the reviews of a product.

Main Success Scenario:

1. The store administrator or the customer selects a product.
2. The system shows the reviews of the selected product
   [→*ShowReviewsOfProduct*]

Use case

Tell to a friend

Primary Actor: Customer.

Precondition: None.

Trigger: A Customer wants to send the current web page to a friend with a comment by email.

Main Success Scenario:

1. The customer provides his name, his friend's name, his friend's email and the message about the web page.
2. The system sends the email with a link to the current web page.

Use case

Generate an invoice

Primary Actor: Store administrator.

Precondition: None.

Trigger: The store administrator wants to generate an invoice corresponding to an order.

Main Success Scenario:

1. The store administrator selects the order.
2. The system shows a printable invoice.

Use case

Generate a packaging slip

Primary Actor: Store administrator.

Precondition: None.

Trigger: The store administrator wants to generate a packaging slip of an order.

Main Success Scenario:

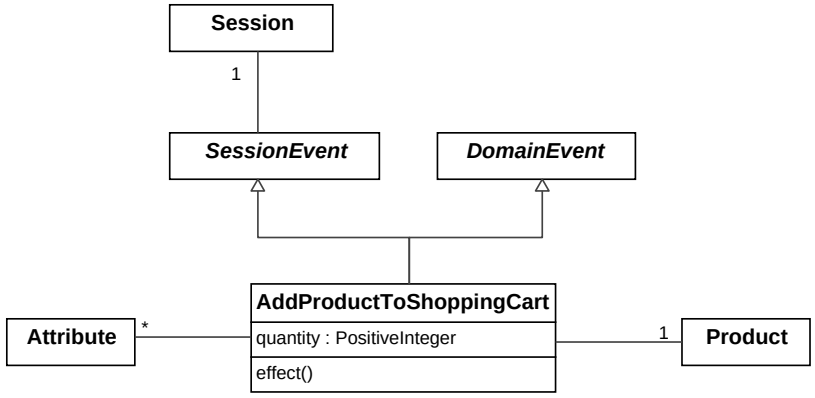
1. The store administrator selects the order.
2. The system shows a printable packaging slip.

3.4 EVENTS SPECIFICATION

Event

AddProductToShoppingCart

Event diagram



Initial Integrity Constraints

```
context AddProductToShoppingCart::AttributesAreFromProduct(): Boolean
  body : self.product.attribute -> includesAll(self.attribute)

context AddProductToShoppingCart::AttributesAreOfDifferentOptions(): Boolean
  body : self.attribute -> isUnique(option)
```

Effect

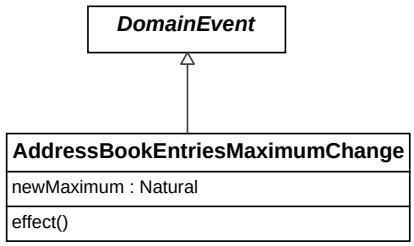
```
context  AddProductToShoppingCart::effect()
   post ShoppingCartItemIsCreated :
      sci.oclIsNew and
      sci.oclIsTypeOf(ShoppingCartItem) and
      sci.quantity = self.quantity and
      sci.product = self.product and
      sci.attribute = self.attribute and
      if self.session.shoppingCart -> notEmpty() then
         --The session has a shopping cart
         self.session.shoppingCart.shoppingCartItem -> includes(sci)
      else
         --The session does not have a shopping cart
         if self.session.customer -> isEmpty() then
         --The session is Anonymous
            sc.oclIsNew()  and
            sc.oclIsTypeOf(AnonymousShoppingCart)  and
```

```
        self.session.shoppingCart = sc and
        sc.shoppingCartItem -> includes(sci)
    else
     --The customer has logged in
        if self.session.customer.customerShoppingCart -> notEmpty()  then
            --The customer has a previous shopping cart
           self.session.shoppingCart = self.session.customer.customerShoppingCart and
           self.session.shoppingCart.shoppingCartItem -> includes(sci)
        else
            --The customer does not have a previous shopping cart
            csc.oclIsNew()  and
            csc.oclIsTypeOf(CustomerShoppingCart)  and
            self.session.shoppingCart = csc and
            csc.shoppingCartItem -> includes(sci)
       endif
    endif
endif
```

Event

AddressBookEntriesMaximumChange

Event diagram



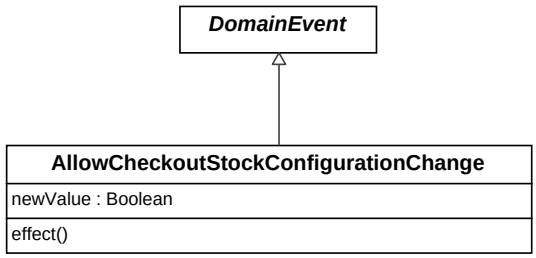
Effect

```
context  AddressBookEntriesMaximumChange::effect()
 post :  MaximumValues.addressBookEntries = self.newMaximum
```

Event

AllowCheckoutStockConfigurationChange

Event diagram



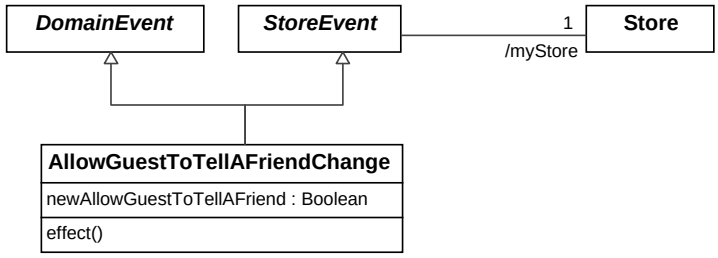
Effect

context AllowCheckoutStockConfigurationChange::effect()
post : Stock.allowCheckout= self.newValue

Event

AllowGuestToTellAFriendChange

Event diagram



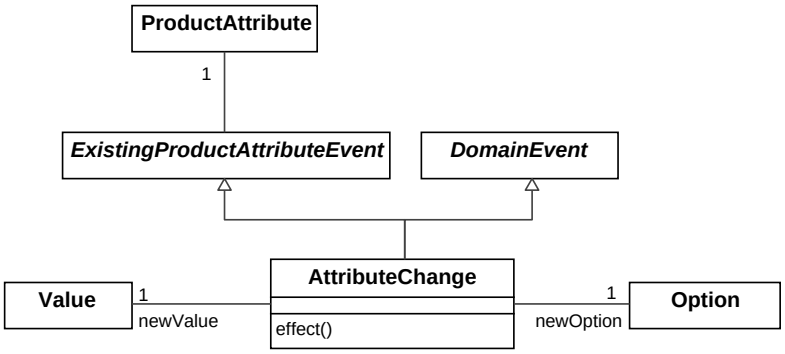
context StoreEvent::myStore():Store
body : Store.allInstances() -> any(true)

Effect

context AllowGuestToTellAFriendChange::effect()
post : myStore.allowGuestToTellAFriend = self.newAllowGuestToTellAFriend

Event
AttributeChange

Event diagram

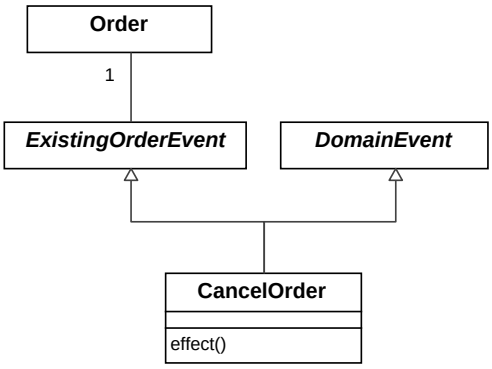


Effect

context AttributeChange::effect()
post :
self.productAttribute.attribute.value = self.newValue **and**
self.productAttribute.attribute.option = self.newOption

Event
CancelOrder

Event diagram



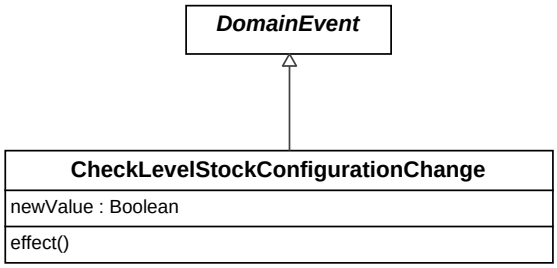
Effect

```
context CancelOrder::effect()
  post:
      self.order.orderStatusChange -> sortedBy(added) -> last().orderStatus =
      Store.allInstances() ->any(true).cancelledStatus
```

Event

CheckLevelStockConfigurationChange

Event diagram



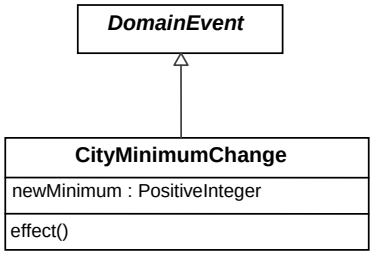
Effect

```
context CheckLevelStockConfigurationChange::effect()
  post : Stock.checkStockLevel= self.newValue
```

Event

CityMinimumChange

Event diagram

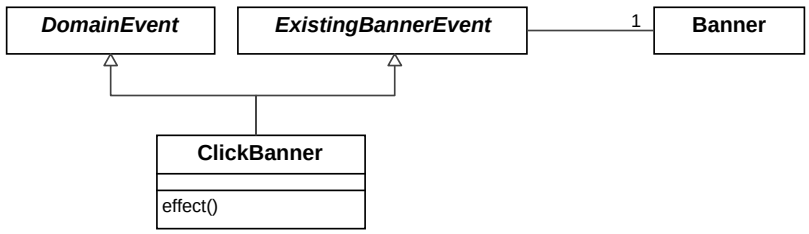


Effect

```
context CityMinimumChange::effect()
 post : MinimumValues.city = self.newMinimum
```

Event
ClickBanner

Event diagram

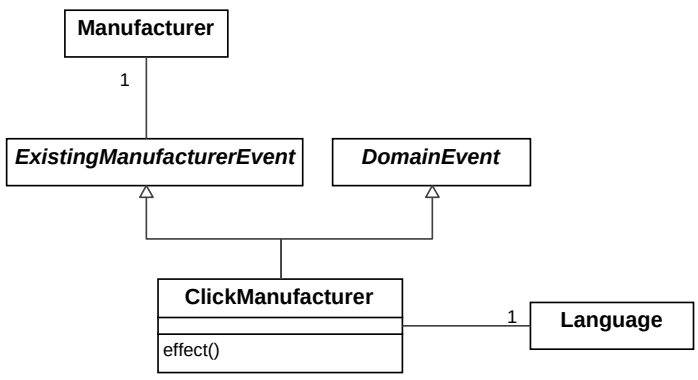


Effect

```
context ClickBanner::effect()
 post :
   BannerHistory.allInstances() -> one
     (bh | bh.banner = self.banner and
           bh.date = Today() and
           bh.clicked = bh@pre.clicked + 1)
```

Event
ClickManufacturer

Event diagram



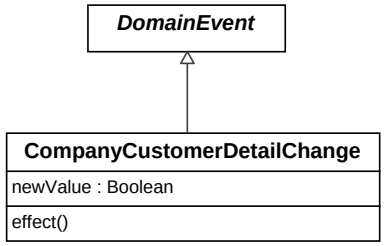
Effect

```
context ClickManufacturer::effect()
 post :
   let manufacturerLanguageRead:ManufacturerInLanguage =
         ManufacturerInLanguage.allInstances() -> select
           (mil | mil.manufacturer = self.manufacturer and
                  mil.language = self.language)
    in
       manufacturerLanguageRead.urlClicked = manufacturerLanguageRead@pre.urlClicked + 1 and
       manufacturerLanguageRead.lastClick = Now()
```

Event

CompanyCustomerDetailChange

Event diagram



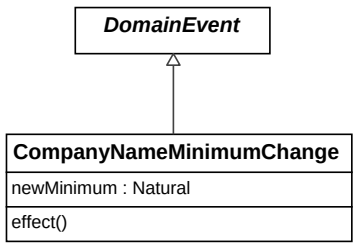
Effect

```
context CompanyCustomerDetailChange::effect()
 post : CustomerDetails.company = self.newValue
```

Event

CompanyNameMinimumChange

Event diagram

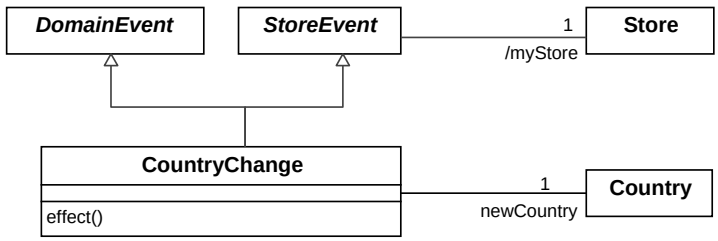


Effect

context CompanyNameMinimumChange::effect()
post : MinimumValues.companyName = self.newMinimum

Event
CountryChange

Event diagram

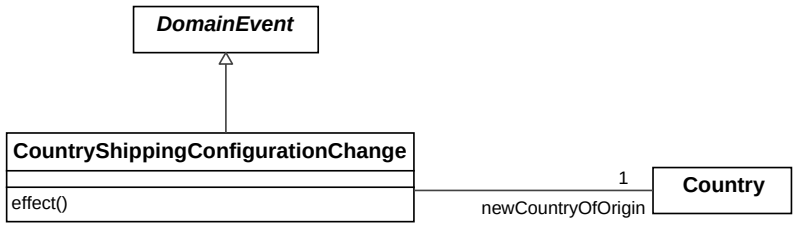


Effect

context CountryChange::effect()
post : myStore.country = self.newCountry

Event
CountryShippingConfigurationChange

Event diagram



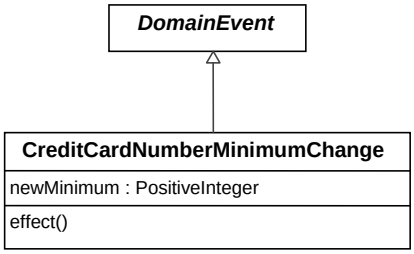
Effect

context CountryShippingConfigurationChange::effect()
post : ShippingAndPackaging.countryOfOrigin = self.newCountryOfOrigin

Event

CreditCardNumberMinimumChange

Event diagram



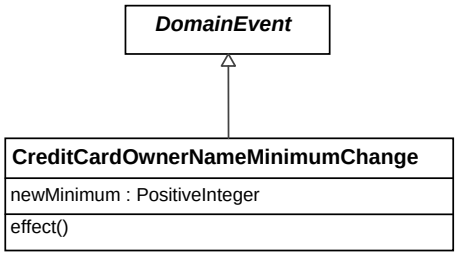
Effect

context CreditCardNumberMinimumChange::effect()
post : MinimumValues.creditCardNumber = self.newMinimum

Event

CreditCardOwnerNameMinimumChange

Event diagram

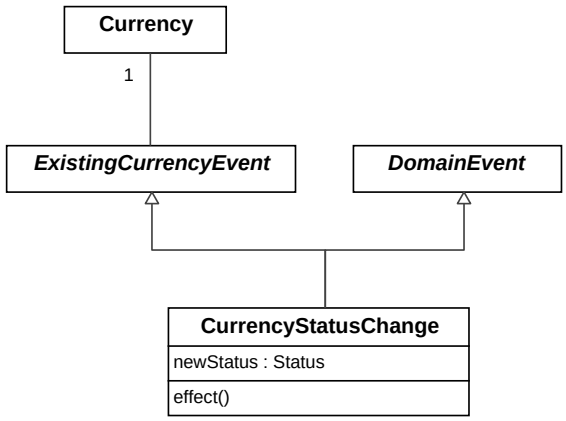


Effect

context CreditCardOwnerNameMinimumChange::effect()
post : MinimumValues.creditCardOwnerName = self.newMinimum

Event
CurrencyStatusChange

- **Event diagram**

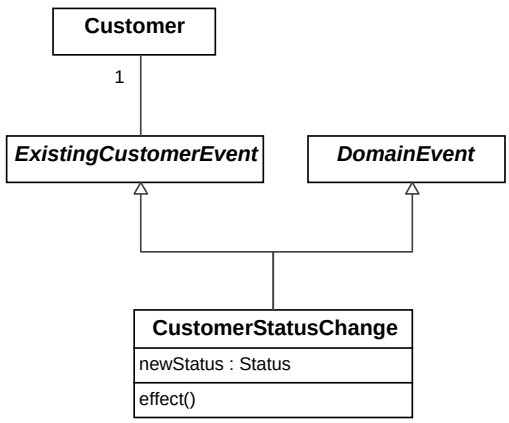


- **Effect**

context CurrencyStatusChange::effect()
post : self.currency.status = self.newStatus

Event
CustomerStatusChange

- **Event diagram**

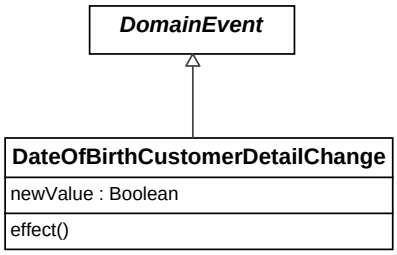


Effect

context CustomerStatusChange::effect()
post : self.customer.status = self.newStatus

Event
DateOfBirthCustomerDetailChange

Event diagram

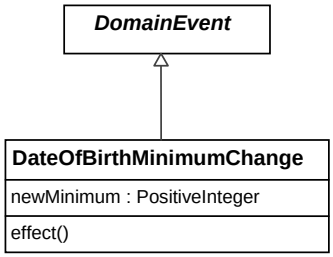


Effect

context DateOfBirthCustomerDetailChange::effect()
post : CustomerDetails.dateOfBirth = self.newValue

Event
DateOfBirthMinimumChange

Event diagram

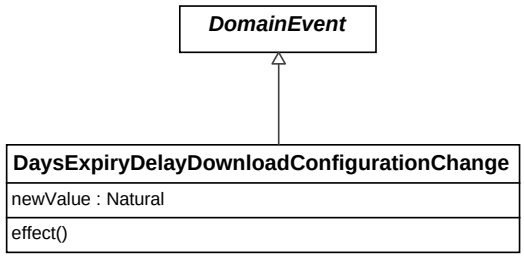


Effect

context DateOfBirthMinimumChange::effect()
post : MinimumValues.dateOfBirth = self.newMinimum

Event
DaysExpiryDelayDownloadConfigurationChange

Event diagram

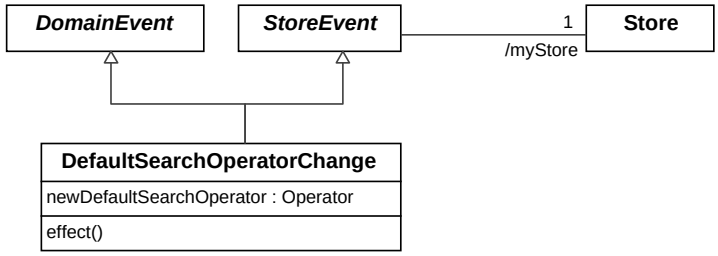


Effect

context DaysExpiryDelayDownloadConfigurationChange::effect()
post : Download.daysExpiryDelay= self.newValue

Event
DefaultSearchOperatorChange

Event diagram

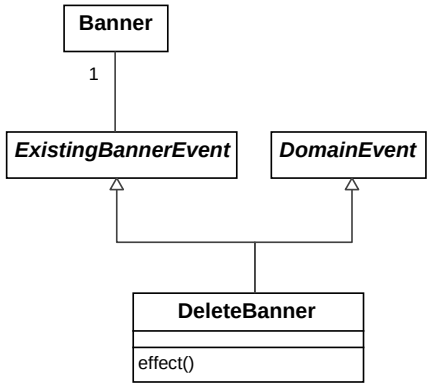


Effect

context DefaultSearchOperatorChange::effect()
post : myStore.defaultSearchOperator = self.newDefaultSearchOperator

Event
DeleteBanner

Event diagram

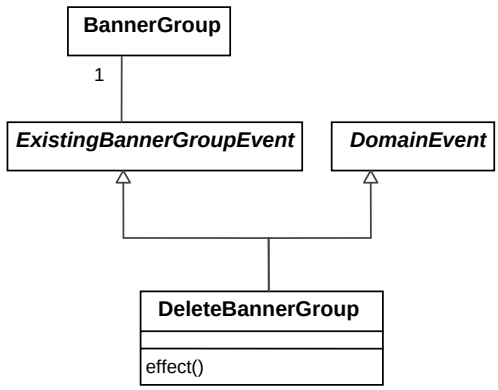


Effect

context DeleteBanner::effect()
post : not self.banner@pre.oclIsKindOf(OclAny)

Event
DeleteBannerGroup

Event diagram



Initial Integrity Constraints

```
context DeleteBannerGroup::BannerGroupIsEmpty():Boolean
  body : self.bannerGroup.banner -> isEmpty()
```

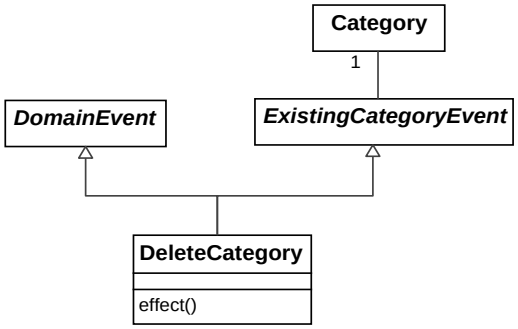
Effect

```
context DeleteBannerGroup::effect()
  post : not self.bannerGroup@pre.oclIsKindOf(OclAny)
```

Event

DeleteCategory

Event diagram



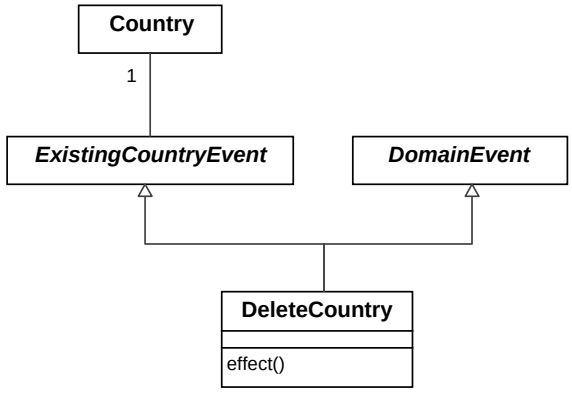
Effect

```
context DeleteCategory::effect()
  post deleteCategoryAndSubcategories:
     not self.category@pre.oclIsKindOf(OclAny) and
     not self.category@pre.child@pre -> forAll(c | c.oclIsKindOf(OclAny))
  post deleteProductsOfCategory:
      self.category@pre.product@pre -> forAll(p |
         if p.orderLine -> notEmpty() then p.status = ProductStatus::outOfStock
         else p@pre.oclIsKindOf(OclAny)
         endif )
  post deleteProductsOfChildCategory:
      self.category@pre.child@pre.product@pre -> forAll(p |
         if p.orderLine -> notEmpty() then p.status = ProductStatus::outOfStock
         else p.oclIsKindOf(OclAny)
         endif )
```

Event

DeleteCountry

Event diagram



Initial Integrity Constraints

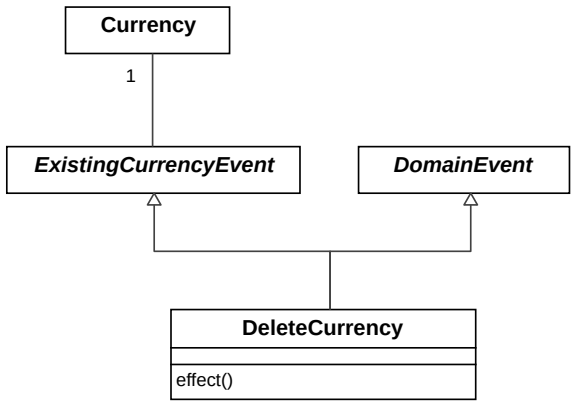
```
context DeleteCountry::CountryIsNotALocation():Boolean
  body :
    Store.allInstances() -> any(true).country <> self.country and
    Address.allInstances().country -> excludes(self.country)
```

Effect

```
context DeleteCountry::effect()
  post : not self.country@pre.oclIsKindOf(OclAny)
```

Event
DeleteCurrency

Event diagram



Initial Integrity Constraints

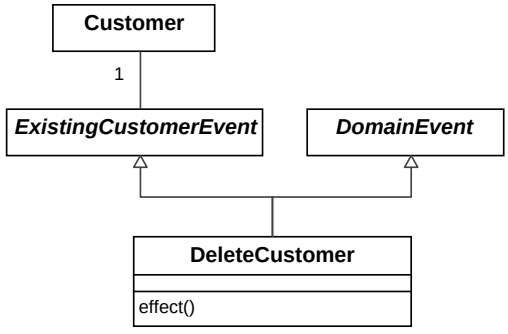
context DeleteCurrency::AtLeastTwoCurrencies(): Boolean
body : Currency.allInstances() -> size() >= 2

Effect

context DeleteCurrency::effect()
post: **not** self.currency@pre.oclIsKindOf(OclAny)

Event
DeleteCustomer

Event diagram



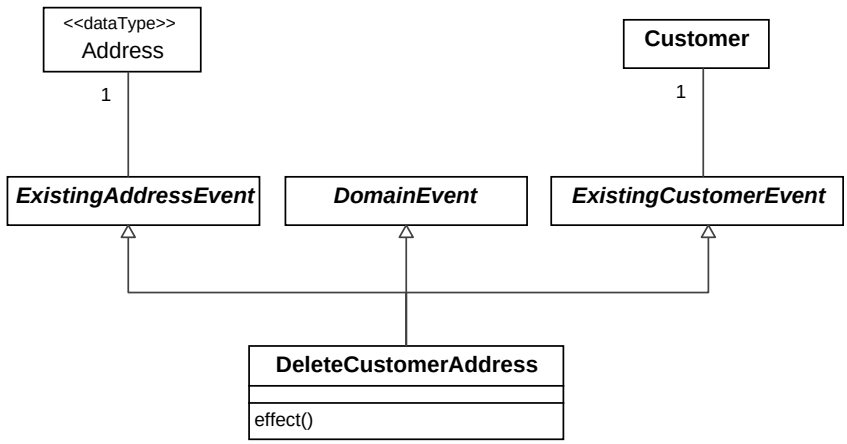
Effect

```
context  DeleteCustomer::effect()
  post deleteCustomer:
    not customer@pre.oclIsKindOf(OclAny)
  post deleteReviewsAndShoppingCart:
    not customer@pre.review@pre -> forAll (r | r.oclIsKindOf(OclAny)) and
    customer@pre.customerShoppingCart->notEmpty()
    implies
    not customer@pre.customerShoppingCart@pre.oclIsKindOf(OclAny))
```

Event

DeleteCustomerAddress

Event diagram



Initial Integrity Constraints

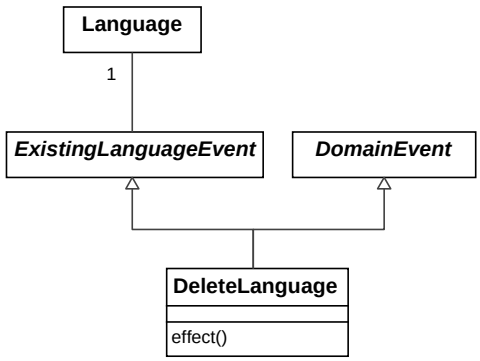
```
context DeleteCustomerAddress::AddressOfCustomer(): Boolean
  body :  self.customer.address -> includes(self.address)

context DeleteCustomerAddress::AtLeastTwoAddresses(): Boolean
  body :  self.customer.address.size() >= 2
```

Effect

```
context  DeleteCustomerAddress::effect()
  post :  self.customer.address -> excludes(self.address)
```

Event
DeleteLanguage

Event diagram



Initial Integrity Constraints

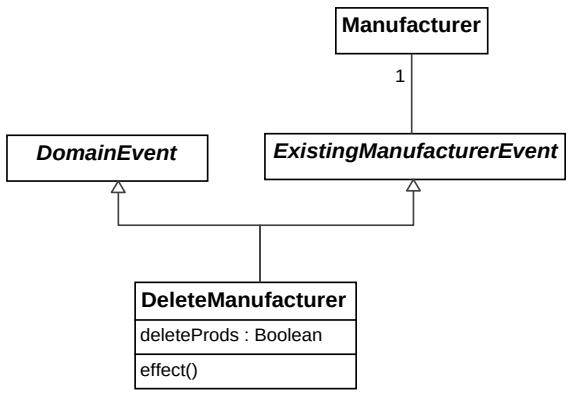
context DeleteLanguage::AtLeastTwoLanguages(): Boolean
body : Language.allInstances() -> size() >= 2

Effect

context DeleteLanguage::effect()
post: **not** self.language@pre.oclIsKindOf(OclAny)

Event
DeleteManufacturer

Event diagram



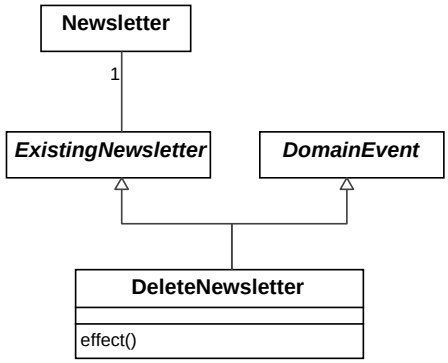
Effect

```
context  DeleteManufacturer::effect()
  post deleteManufacturer:
    not self.manufacturer@pre.oclIsKindOf(OclAny)
  post changeProductsToOutOfStock:
    deleteProds implies
      manufacturer@pre.product@pre ->
        forAll(status = ProductStatus::outOfStock)
```

Event

DeleteNewsletter

Event diagram



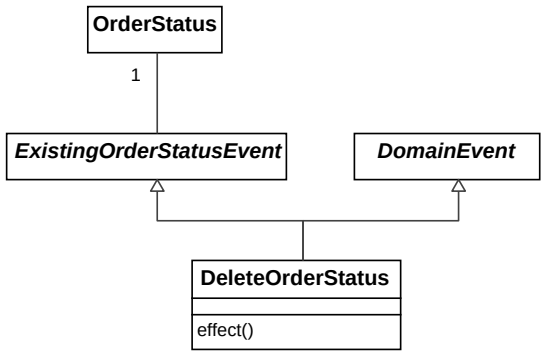
Effect

```
context  DeleteNewsletter::effect()
  post :  not self.newsletter@pre.oclIsKindOf(OclAny)
```

Event

DeleteOrderStatus

Event diagram



Initial Integrity Constraints

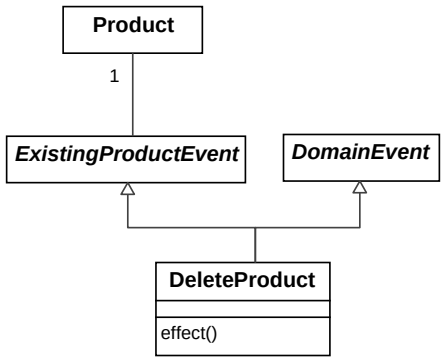
```
context DeleteOrderStatus:: IsNotTheCurrentStatusOfAnyOrder(): Boolean
 body :
    Order.allInstances() -> forAll (o | o.orderStatusChange -> sortedBy(added)
                                      -> last().orderStatus <> self.orderStatus)
```

Effect

```
context  DeleteOrderStatus::effect()
  post :  not self.orderStatus@pre.oclIsKindOf(OclAny)
```

Event
DeleteProduct

Event diagram

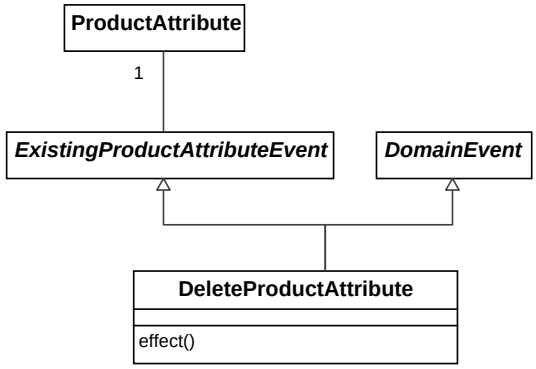


Effect

```
context DeleteProduct::effect()
  post: not product@pre.oclIsKindOf(OclAny)
```

Event
DeleteProductAttribute

Event diagram

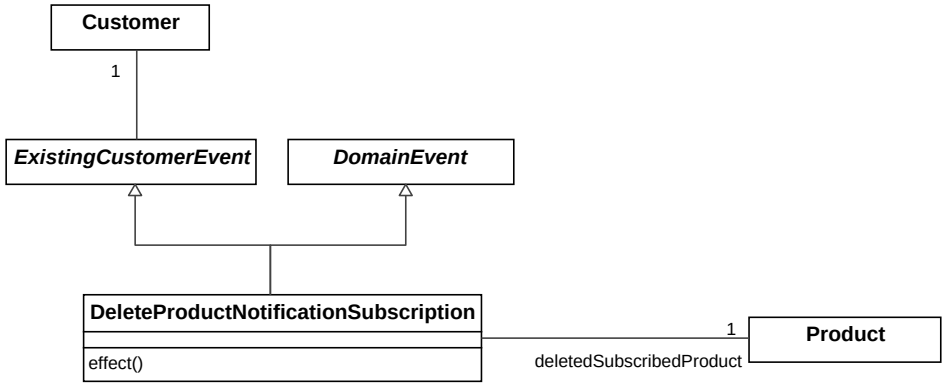


Effect

context DeleteProductAttribute::effect()
post: **not** productAttribut@pre.oclIsKindOf(OclAny)

Event
DeleteProductNotificationSubscription

Event diagram

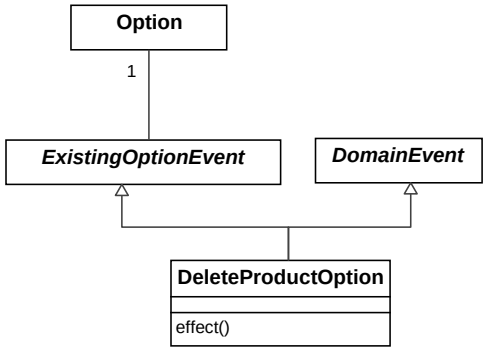


Effect

context DeleteProductNotificationSubscription::effect()
post : customer.explicitNotifications -> excludes(self.deletedSubscribedProduct)

Event
DeleteProductOption

Event diagram



Initial Integrity Constraints

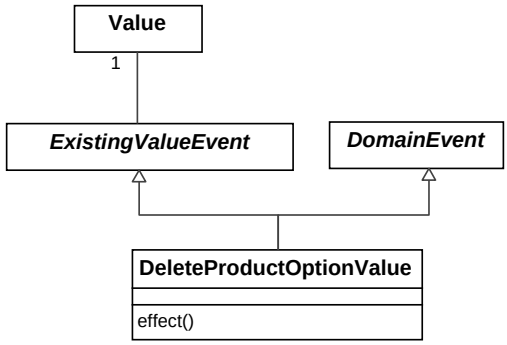
context DeleteProductOption::HasNotProductsOrValues():Boolean
body : self.option.value -> isEmpty() **and** self.option.attribute.product -> isEmpty()

Effect

context DeleteProductOption::effect()
post : **not** self.option@pre.oclIsKindOf(OclAny)

Event
DeleteProductOptionValue

Event diagram



Initial Integrity Constraints

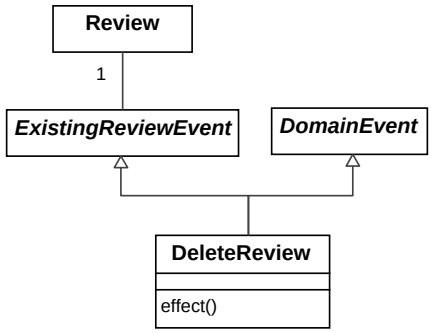
context DeleteProductOptionValue::HasNotProducts():Boolean
body : self.value.attribute.product -> isEmpty()

Effect

context DeleteProductOptionValue::effect()
post : **not** self.value@pre.oclIsKindOf(OclAny)

Event
DeleteReview

- **Event diagram**

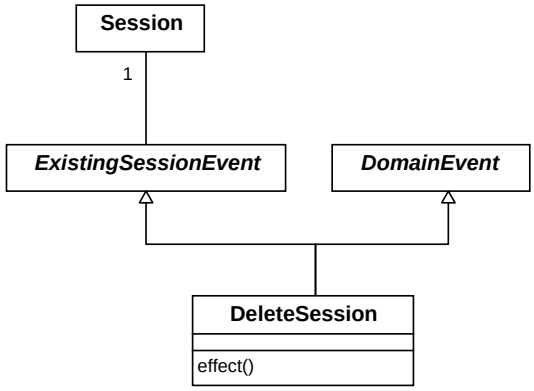


- **Effect**

```
context DeleteReview::effect()
  post : not self.review@pre.oclIsKindOf(OclAny)
```

Event
DeleteSession

- **Event diagram**



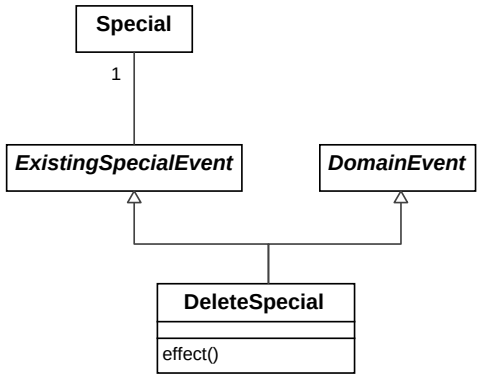
- **Effect**

```
context DeleteSession::effect()
  post : not self.session@pre.oclIsKindOf(OclAny)
```

Event

DeleteSpecial

Event diagram



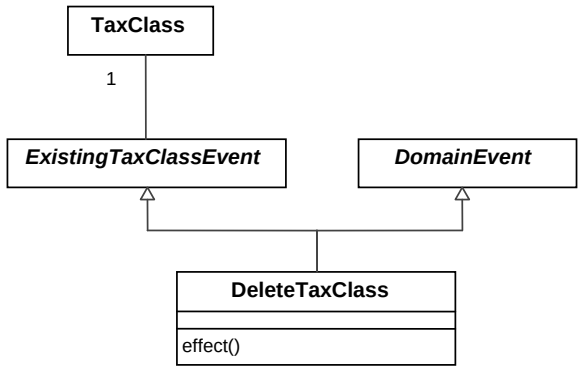
Effect

```
context DeleteSpecial::effect()
  post :
    special.oclIsTypeOf(Product) and
    not special.oclIsTypeOf(Special)
```

Event

DeleteTaxClass

Event diagram



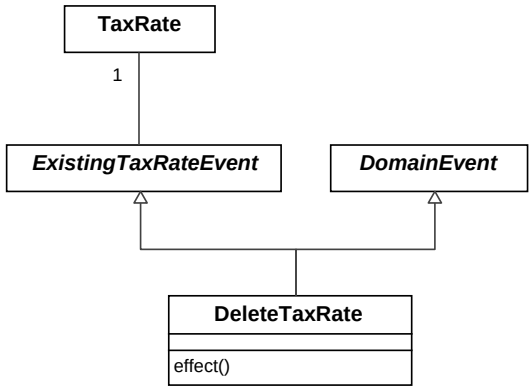
Effect

```
context  DeleteTaxClass::effect()
  post deleteTaxClass:
    not self.taxClass@pre.oclIsKindOf(OclAny)
  post deleteAssociatedTaxRates:
    self.taxClass@pre.taxRate@pre -> forAll(tr | tr.oclIsKindOf(OclAny))
```

Event

DeleteTaxRate

Event diagram



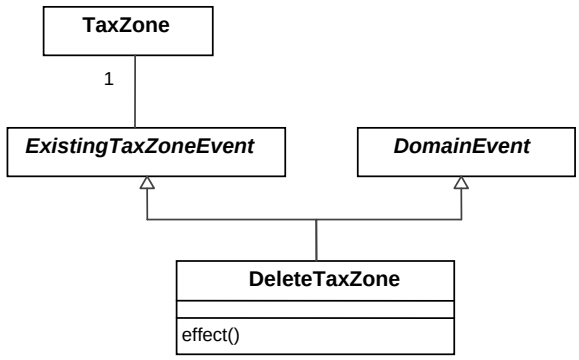
Effect

```
context  DeleteTaxRate::effect()
  post :  not self.taxRate@pre.oclIsKindOf(OclAny)
```

Event

DeleteTaxZone

Event diagram



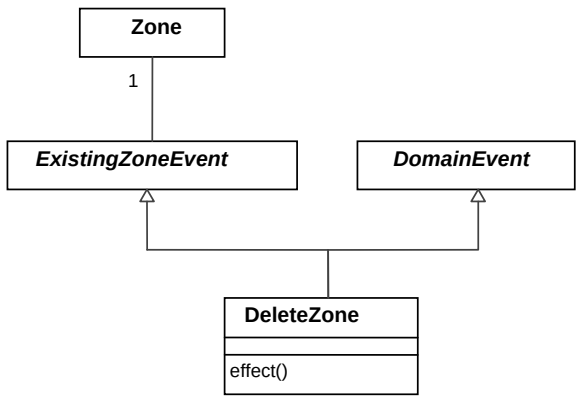
Effect

```
context  DeleteTaxZone::effect()
  post deleteTaxZone:
    not self.taxZone@pre.oclIsKindOf(OclAny)
  post deleteAssociatedTaxRates:
    self.taxZone@pre.taxRate@pre -> forAll(tr | tr.oclIsKindOf(OclAny))
```

Event

DeleteZone

Event diagram



Initial Integrity Constraints

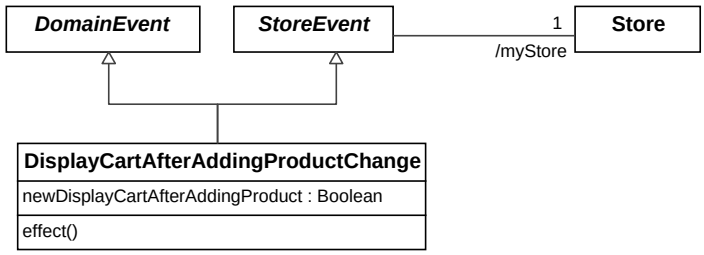
context DeleteZone::ZoneIsNotALocation():Boolean
body : Store.allInstances() -> any(true).zone <> self.zone **and**
Address.allInstances().zone -> excludes(self.zone)

Effect

context DeleteZone::effect()
post : **not** self.zone@pre.oclIsKindOf(OclAny)

Event
DisplayCartAfterAddingProductChange

Event diagram

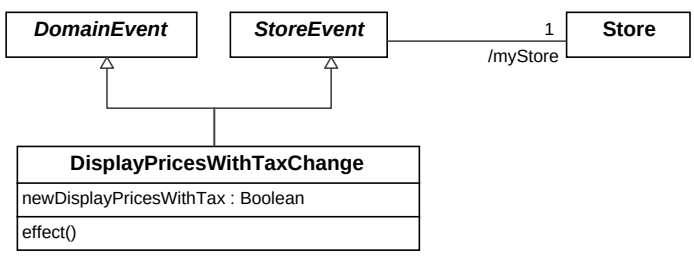


Effect

context DisplayCartAfterAddingProductChange::effect()
post : myStore.displayCartAfterAddingProduct = self.newDisplayCartAfterAddingProduct

Event
DisplayPricesWithTaxChange

Event diagram



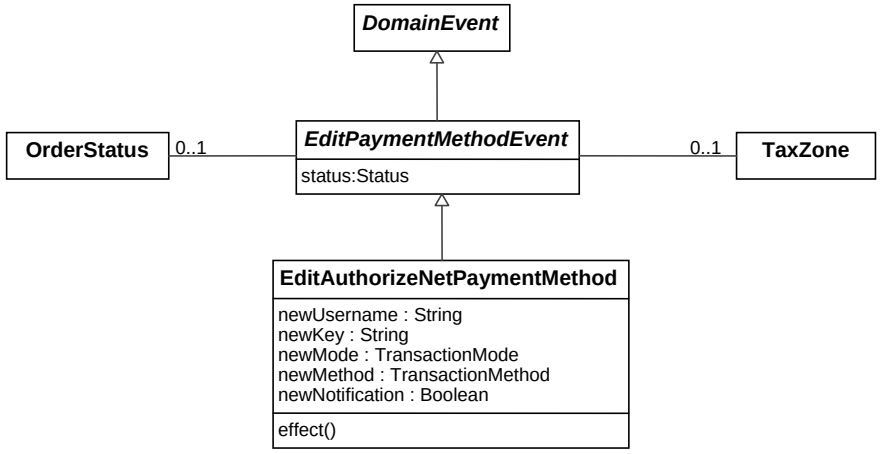
Effect

```
context DisplayPricesWithTaxChange::effect()
 post : myStore.displayPricesWithTax = self.newDisplayPricesWithTax
```

Event

EditAuthorizeNetPaymentMethod

Event diagram



Initial Integrity Constraints

```
context EditAuthorizeNetPaymentMethod::PaymentMethodIsInstalled():Boolean
 body : AuthorizeNet.allInstances() -> notEmpty()
```

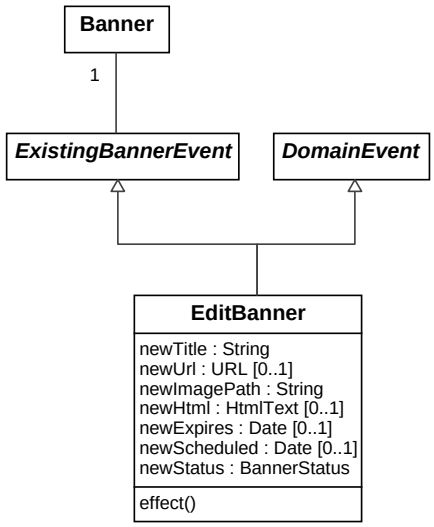
Effect

```
context EditAuthorizeNetPaymentMethod::effect()
 post :
   let pm:AuthorizeNet = AuthorizeNet.allInstances() -> any(true)
    in
   pm.username=self.newUsername and
   pm.key=self.newKey and
   pm.mode=self.newMode and
   pm.method=self.newMethod and
   pm.notification=self.newNotification and
   pm.orderStatus=self.orderStatus and
   pm.status=self.status and
   pm.taxZone=self.taxZone
```

Event

EditBanner

Event diagram



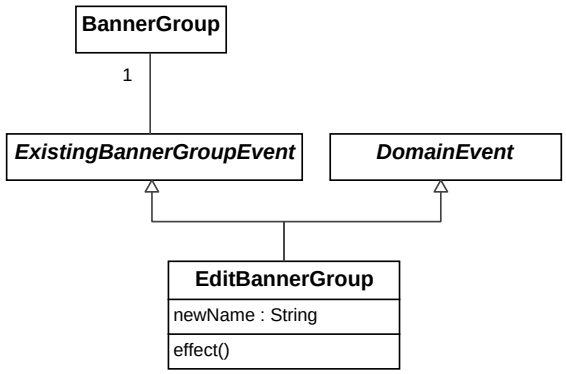
Effect

```
context  EditBanner::effect()
  post :
     self.banner.title = self.newTitle and
     self.banner.url = self.newUrl and
     self.banner.imagePath = self.newImagePath and
     self.banner.html = self.newHtml and
     self.banner.expires = self.newExpires and
     self.banner.scheduled = self.newScheduled and
     self.banner.status = self.newStatus
  post :
     self.banner@pre.status <> self.newStatus implies self.banner.statusChanged = Now()
```

Event

EditBannerGroup

Event diagram



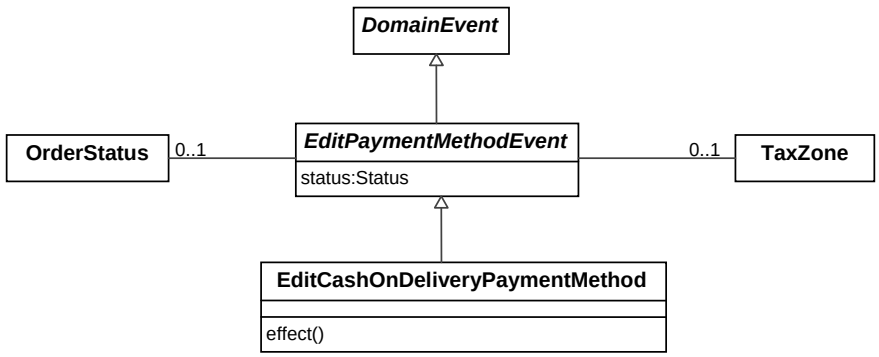
Effect

context EditBannerGroup::effect()
post : self.bannerGroup.name = self.newName

Event

EditCashOnDeliveryPaymentMethod

Event diagram



Initial Integrity Constraints

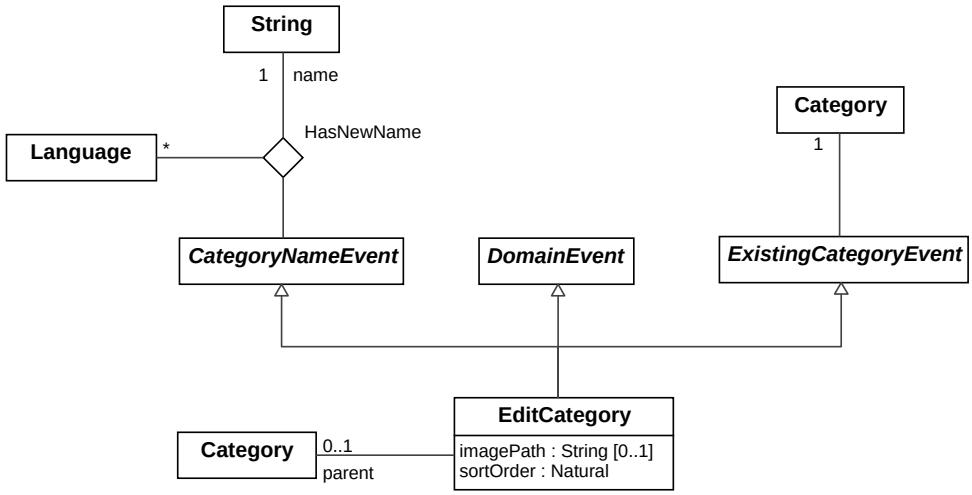
```
context EditCashOnDeliveryPaymentMethod::PaymentMethodIsInstalled():Boolean
  body : CashOnDelivery.allInstances() -> notEmpty()
```

Effect

```
context EditCashOnDeliveryPaymentMethod::effect()
  post :
    let pm:CashOnDelivery = CashOnDelivery.allInstances() -> any(true) in
    pm.orderStatus=self.orderStatus and
    pm.status=self.status and
    pm.taxZone=self.taxZone
```

Event
EditCategory

Event diagram



Effect

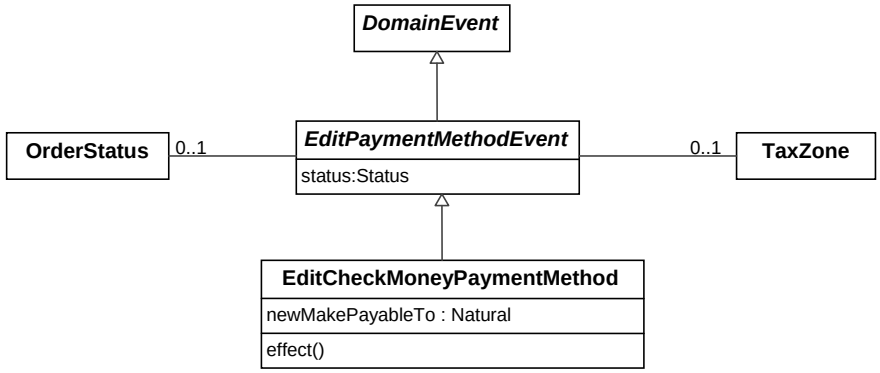
```
context EditCategory::effect()
  post :
    self.category.imagePath = self.imagePath and
    self.category.sortOrder = self.sortOrder and
    self.category.parent = self.parent and
    Language.allInstances() ->
      forAll(l|
```

self.hasNewName->select(language=l).name=
self.category.hasCategoryName->select(language=l).categoryName)
post :
self.category.lastModified = Now()

Event

EditCheckMoneyPaymentMethod

Event diagram



Initial Integrity Constraints

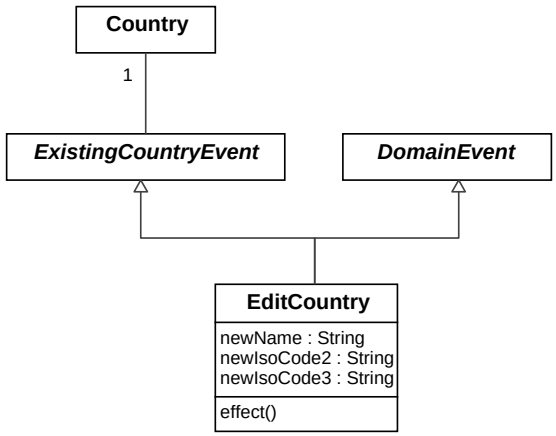
context EditCheckMoneyPaymentMethod::PaymentMethodIsInstalled():Boolean
body : CheckMoney.allInstances() -> notEmpty()

Effect

context EditCheckMoneyPaymentMethod::effect()
post :
let pm: CheckMoney = CheckMoney.allInstances() -> any(true) **in**
pm.makePayableTo=self.newMakePayableTo **and**
pm.orderStatus=self.orderStatus **and**
pm.status=self.status **and**
pm.taxZone=self.taxZone

Event
EditCountry

Event diagram

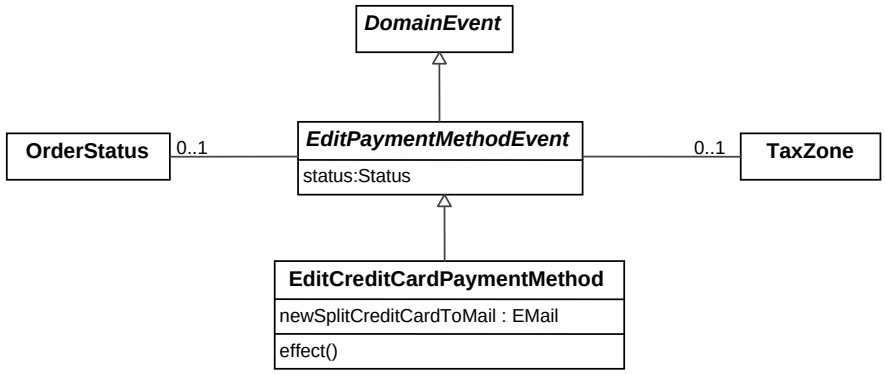


Effect

context EditCountry::effect()
post :
country.name = self.newName **and**
country.isoCode2 = self.newIsoCode2 **and**
country.isoCode3 = self.newIsoCode3

Event
EditCreditCardPaymentMethod

Event diagram



Initial Integrity Constraints

```
context EditCreditCardPaymentMethod::PaymentMethodIsInstalled():Boolean
  body : CreditCard.allInstances() -> notEmpty()
```

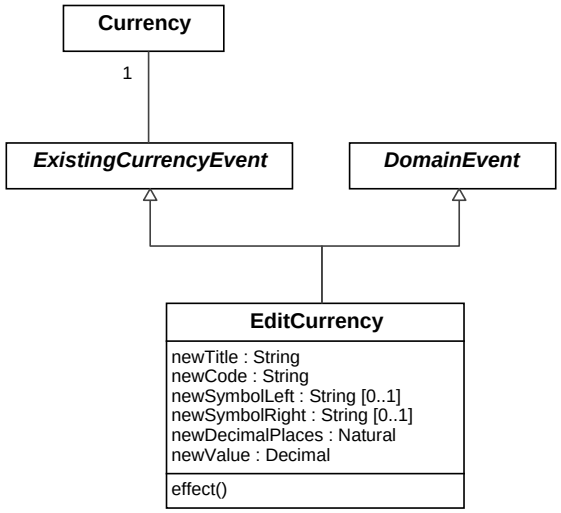
Effect

```
context EditCreditCardPaymentMethod::effect()
  post :
    let pm:CreditCard = CreditCard.allInstances() -> any(true) in
    pm.splitCreditCardToMail=self.newSplitCreditcardToMail and
    pm.status=self.status and
    pm.orderStatus=self.orderStatus and
    pm.taxZone=self.taxZone
```

Event

EditCurrency

Event diagram

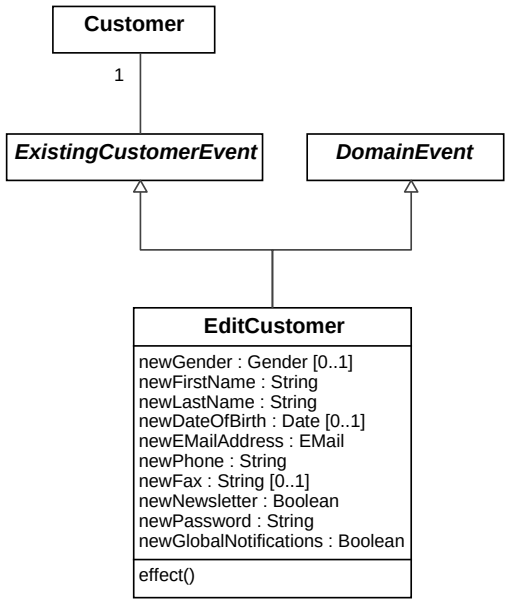


Effect

```
context EditCurrency::effect()
  post :
    currency.title = self.newTitle and
    currency.code = self.newCode and
    currency.symbolLeft = self.newSymbolLeft and
    currency.symbolRight = self.newSymbolRight and
    currency.decimalPlaces = self.newDecimalPlaces and
    currency.value = self.newValue
```

Event
EditCustomer

Event diagram



Initial Integrity Constraints

context EditCustomer::firstNameRight(): Boolean
body : self.newFirstName.size() >= MinimumValues.firstName

context EditCustomer::lastNameRight(): Boolean
body : self.newLastName.size() >= MinimumValues.lastName

context EditCustomer::dateOfBirthRight(): Boolean
body :
CustomerDetails.dateOfBirth **implies**
self.newDateOfBirth->notEmpty() **and**
self.newDateOfBirth.size() >= MinimumValues.dateOfBirth

context EditCustomer::genderRight(): Boolean
body : CustomerDetails.gender **implies** self.newGender->notEmpty()

context EditCustomer::eMailRight(): Boolean
body : self.newEMailAddress.size() >= MinimumValues.eMailAddress

context EditCustomer::telephoneRight(): Boolean
body : self.newTelephone.size() >= MinimumValues.telephoneNumber

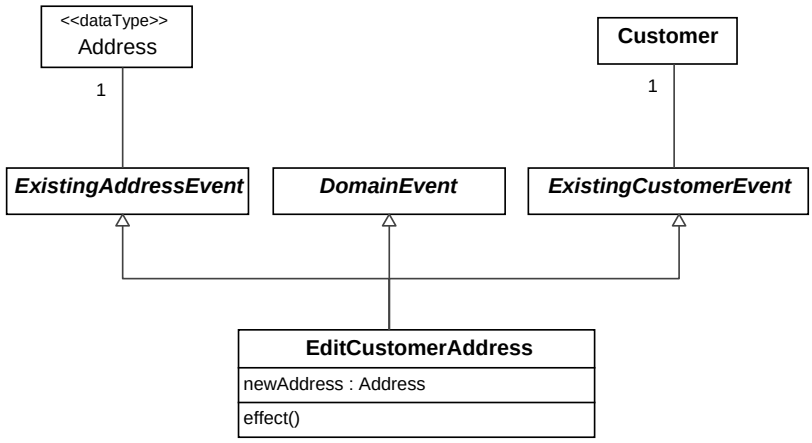
Effect

context EditCustomer::effect()
post :
customer.gender = self.newGender **and**
customer.firstName = self.newFirstName **and**
customer.lastName = self.newLastName **and**
customer.dateOfBirth = self.newDateOfBirth **and**
customer.eMailAddress = self.newEMailAddress **and**
customer.phone = self.newPhone **and**
customer.fax = self.newFax **and**
customer.newsletter = self.newNewsletter **and**
customer.password = self.newPassword **and**
customer.globalNotifications = self.newGlobalNotifications **and**
post :
customer.lastModified = Now()

Event

EditCustomerAddress

Event diagram



Initial Integrity Constraints

context EditCustomerAddress::AddressOfCustomer(): Boolean
body : self.customer.address -> includes(self.address)

context EditCustomerAddress::firstNameRight(): Boolean
body : self.newAdress.firstName.size() >= MinimumValues.firstName

context EditCustomerAddress::lastNameRight(): Boolean
body : self. newAdress.lastName.size() >= MinimumValues.lastName

context EditCustomerAddress::genderRight(): Boolean
body : CustomerDetails.gender **implies** self. newAdress.gender->notEmpty()

context EditCustomerAddress::suburbRight(): Boolean

body : CustomerDetails.suburb **implies** self. newAdress.suburb->notEmpty()

context EditCustomerAddress::streetAddressRight(): Boolean
body : self.newAdress.street.size() >= MinimumValues.streetAddress

context EditCustomerAddress::companyRight(): Boolean
body :
CustomerDetails.company **implies**
self.newAdress.company -> notEmpty() **and**
self.newAdress.company.size() >= MinimumValues.companyName

context EditCustomerAddress::postCodeRight(): Boolean
body : self.newAdress.postCode.size() >= MinimumValues.postCode

context EditCustomerAddress::cityRight(): Boolean
body : self.newAdress.city.size() >= MinimumValues.city

context EditCustomerAddress::stateRight(): Boolean
body :
CustomerDetails.state **implies**
self.newAdress.state -> notEmpty() **and**
self.newAdress.state.size() >= MinimumValues.state

context EditCustomerAddress::addressesHaveZoneIfNeeded(): Boolean
body :
self.newAdress.zone -> notEmpty() **implies**
self.newAdress.state = self.newAdress.zone.name **and**
self.newAdress.country = self.newAdress.zone.country

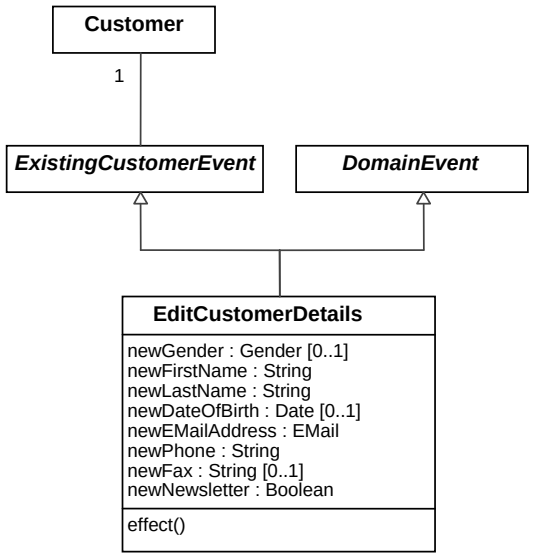
Effect

context EditCustomerAddress::effect()
post :
self.customer.address -> excludes(self.address) **and**
self.customer.address ->includes(self.newAddress)

Event

EditCustomerDetails

Event diagram



Initial Integrity Constraints

context EditCustomerDetails::firstNameRight(): Boolean
body : self.newFirstName.size() >= MinimumValues.firstName

context EditCustomerDetails::lastNameRight(): Boolean
body : self.newLastName.size() >= MinimumValues.lastName

context EditCustomerDetails::dateOfBirthRight(): Boolean
body :
CustomerDetails.dateOfBirth **implies**
self.newDateOfBirth->notEmpty()
self.newDateOfBirth.size() >= MinimumValues.dateOfBirth

context EditCustomerDetails::genderRight(): Boolean
body : CustomerDetails.gender **implies** self.newGender->notEmpty()

context EditCustomerDetails::eMailRight(): Boolean
body : self.newEMailAddress.size() >= MinimumValues.eMailAddress

context EditCustomerDetails::telephoneRight(): Boolean
body : self.newTelephone.size() >= MinimumValues.telephoneNumber

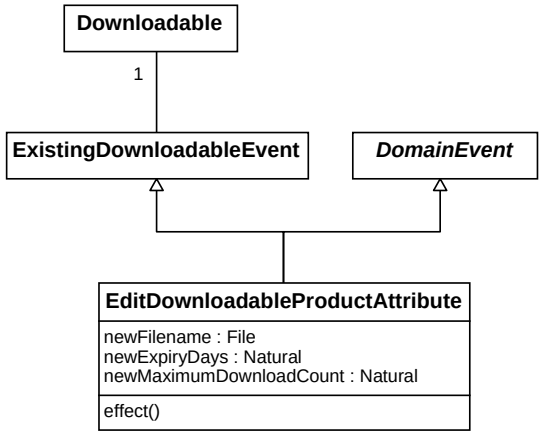
Effect

context EditCustomerDetails::effect()
post :
customer.gender = self.newGender **and**
customer.firstName = self.newFirstName **and**
customer.lastName = self.newLastName **and**
customer.dateOfBirth = self.newDateOfBirth **and**
customer.eMailAddress = self.newEMailAddress **and**
customer.phone = self.newPhone **and**
customer.fax = self.newFax **and**
customer.newsletter = self.newNewsletter

Event

EditDownloadableAttribute

Event diagram



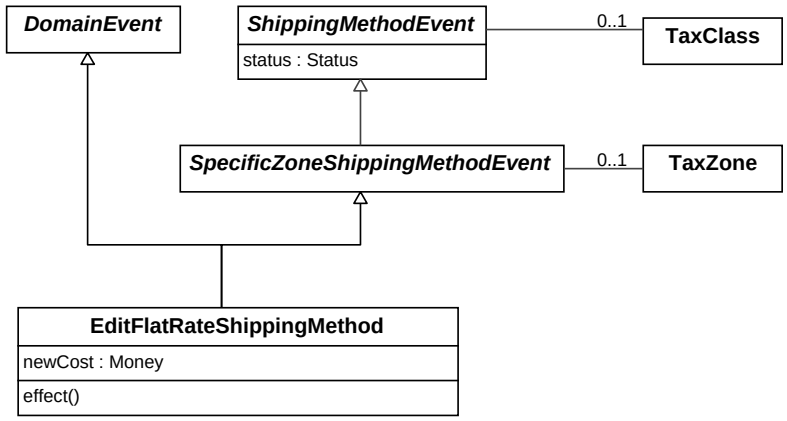
Effect

context EditDownloadableProductAttribute::effect()
post :
self.downloadable.filename = self.newFilename **and**
self.downloadable.expiryDays = self.newExpiryDays **and**
self.downloadable.maximumDownloadCount = self.newMaximumDonwloadCount

Event

EditFlatRateShippingMethod

Event diagram



Initial Integrity Constraints

context EditFlatRateShippingMethod::PaymentMethodIsInstalled():Boolean
body : FlatRate.allInstances() -> notEmpty()

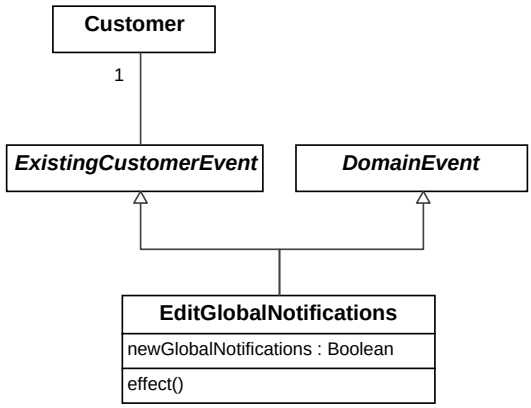
Effect

context EditFlatRateShippingMethod::effect()
post :
let sm: FlatRate= FlatRate.allInstances() -> any(true) **in**
sm.cost=self.newCost **and**
sm.taxZone=self.taxZone **and**
sm.taxClass=self.taxClass **and**
sm.status = self.status

Event

EditGlobalNotifications

Event diagram



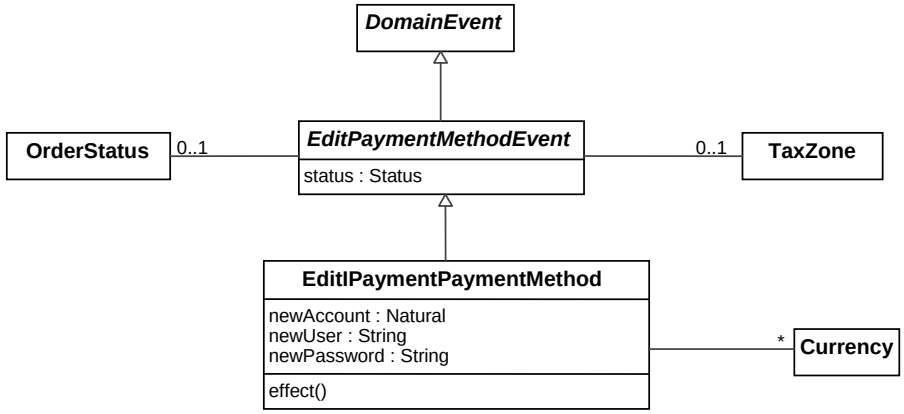
Effect

context EditGlobalNotifications::effect()
post : self.customer.globalNotifications = self.newGlobalNotifications

Event

EditIPaymentPaymentMethod

Event diagram



Initial Integrity Constraints

```
context EditIPaymentPaymentMethod::PaymentMethodIsInstalled():Boolean
  body : IPayment.allInstances() -> notEmpty()
```

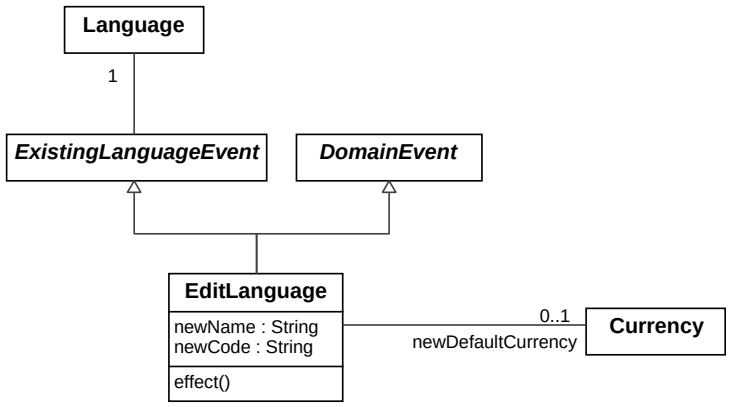
Effect

```
context EditIPaymentPaymentMethod::effect()
  post :
    let pm:IPayment = IPayment.allInstances() -> any(true) in
    pm.account=self.newAccount and
    pm.user=self.newUser and
    pm.password=self.newPassword and
    pm.status=self.status and
    pm.orderStatus=self.orderStatus and
    pm.taxZone=self.taxZone
```

Event

EditLanguage

Event diagram



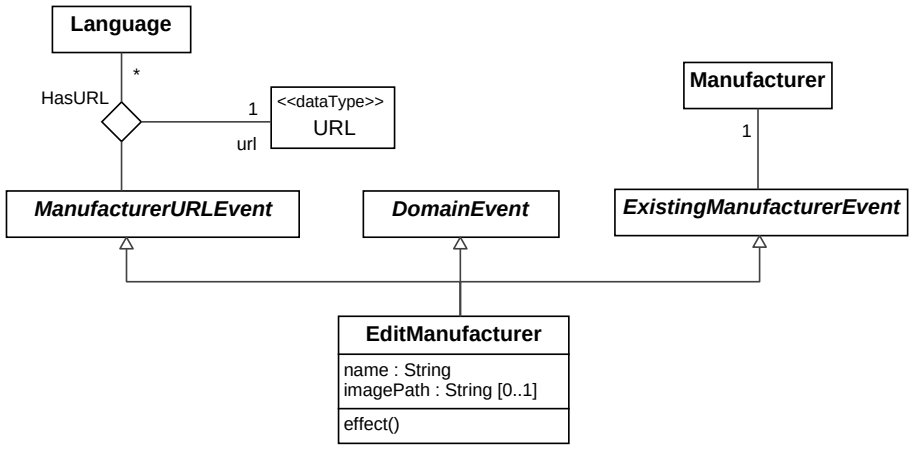
Effect

```
context EditLanguage::effect()
  post :
    self.language.name = self.newName and
    self.language.code = self.newCode and
    self.language.defaultCurrency = self.newDefaultCurrency
```

Event

EditManufacturer

Event diagram



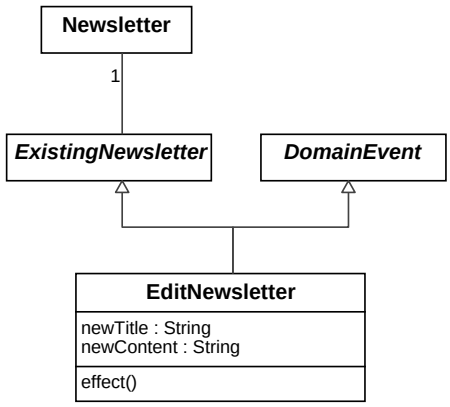
Effect

```
context  EditManufacturer::effect()
  post :
     self.manufacturer.name = self.name and
     self.manufacturer.imagePath = self.imagePath and
     Langauge.allInstances() ->
        forAll(l|
           self.hasURL->select(language=l).url=
           self.manufacturer.manufacturerInLanguage->
              select(language=l).url)
  post :
     self.manufacturer.lastModified = Now()
```

Event

EditNewsletter

Event diagram



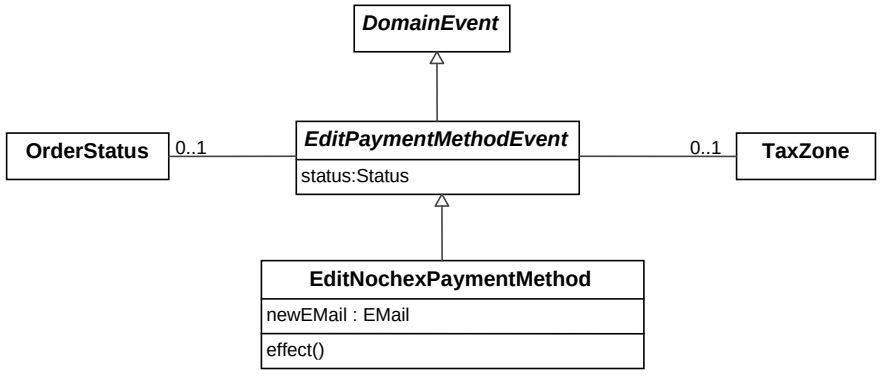
Effect

context EditNewsletter::effect()
post :
newsletter.title = self.newTitle **and**
newsletter.content = self.newContent

Event

EditNochexPaymentMethod

Event diagram



Initial Integrity Constraints

```
context EditNochexPaymentMethod::PaymentMethodIsInstalled():Boolean
  body : CheckMoney.allInstances() -> notEmpty()
```

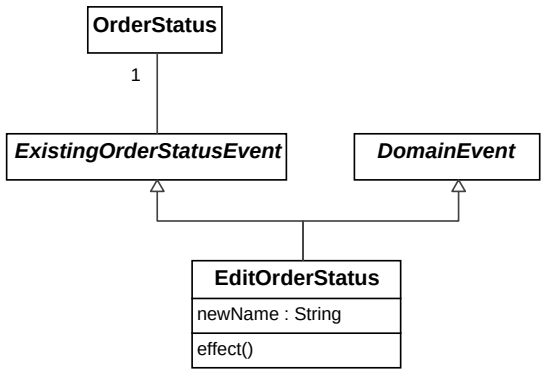
Effect

```
context EditNochexPaymentMethod::effect()
  post :
    let pm: Nochex = Nochex.allInstances() -> any(true) in
    pm.eMail=self.newEMail and
    pm.status=self.status and
    pm.orderStatus=self.orderStatus and
    pm.taxZone=self.taxZone
```

Event

EditOrderStatus

Event diagram



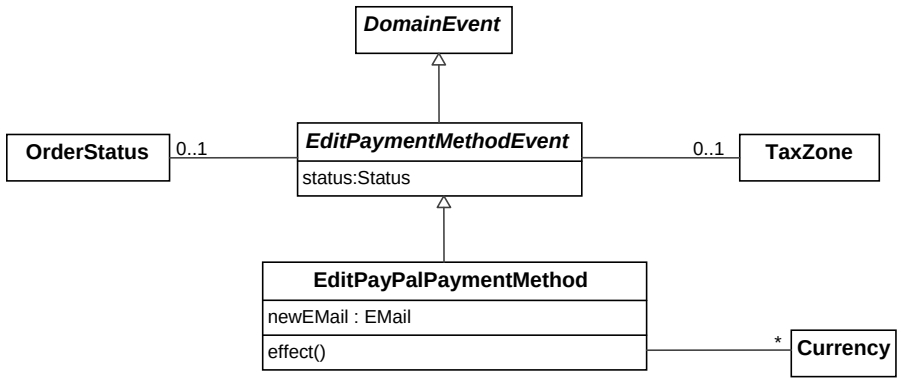
Effect

```
context EditOrderStatus::effect()
  post : self.orderStatus.name = self.newName
```

Event

EditPayPalPaymentMethod

Event diagram



Initial Integrity Constraints

```
context EditPayPalPaymentMethod::PaymentMethodIsInstalled():Boolean
  body : PayPal.allInstances() -> notEmpty()
```

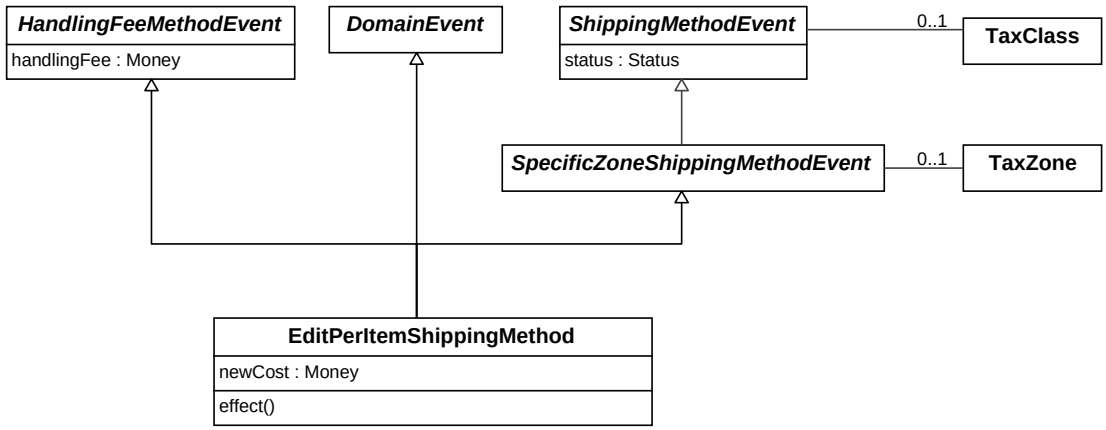
Effect

```
context EditPayPalPaymentMethod::effect()
  post :
    let pm: PayPal = PayPal.allInstances() -> any(true) in
    pm.eMail=self.newEMail and
    pm.status=self.status and
    pm.orderStatus=self.orderStatus and
    pm.taxZone=self.taxZone
```

Event

EditPerItemShippingMethod

Event diagram



Initial Integrity Constraints

```
context EditPerItemShippingMethod::PaymentMethodIsInstalled():Boolean
  body : PerItem.allInstances() -> notEmpty()
```

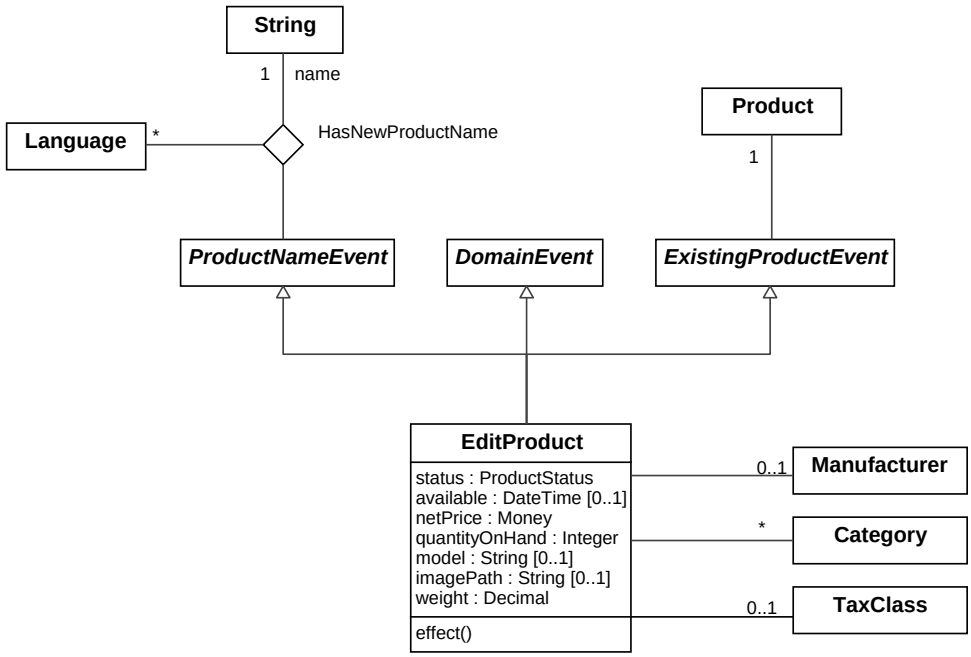
Effect

```
context EditPerItemShippingMethod::effect()
  post :
    let sm: PerItem= PerItem.allInstances() -> any(true) in
    sm.cost=self.newCost and
    sm.handlingFee=self.handlingFee and
    sm.taxZone=self.taxZone and
    sm.taxClass=self.taxClass and
    sm.status = self.status
```

Event

EditProduct

Event diagram

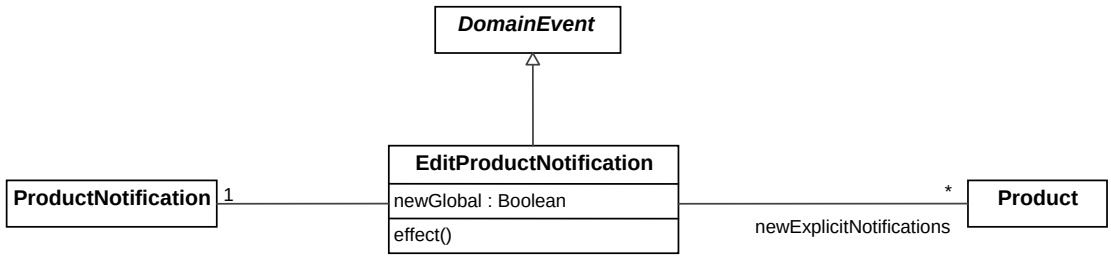


Effect

```
context  EditProduct::effect()
  post :
     self.product.status = self.status and
     self.product.available = self.available and
     self.product.netPrice = self.netPrice and
     self.product.quantityOnHand = self.quantityOnHand and
     self.product.model = self.model and
     self.product.imagePath = self.imagePath and
     self.product.weight = self.weight and
     self.product.manufacturer = self.manufacturer and
     self.product.category = self.category and
     self.product.taxClass = self.taxClass and
     Language.allInstances()
        -> forAll (l|
           self.hasNewProductName -> select(language=l).name =
           self.product.productInLanguage->select(language=l).name)
  post :
     self.product.lastModified = Now()
```

Event
EditProductNotification

Event diagram

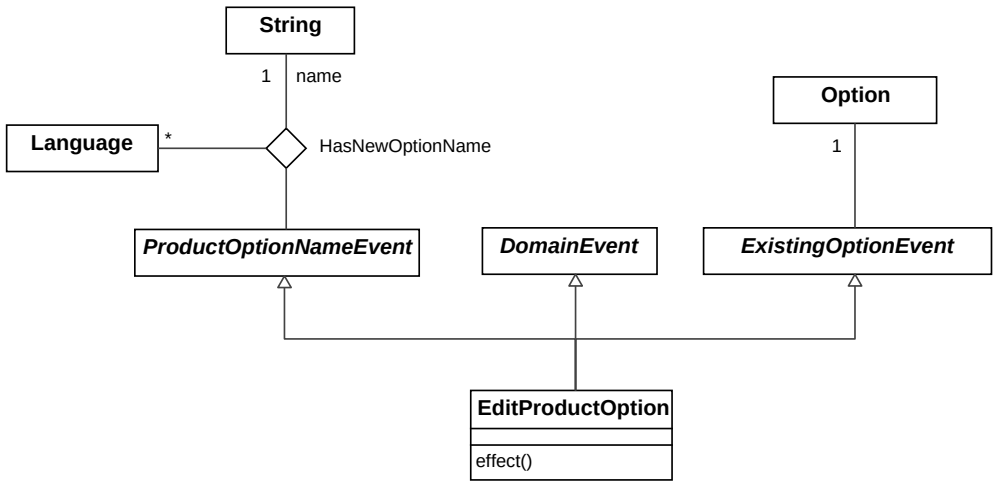


Effect

context EditProductNotification::effect()
post :
self.productNotification.global = self.newGlobal and
self.productNotification.explicitNotifications = self.newExplicitNotifications

Event
EditProductOption

Event diagram



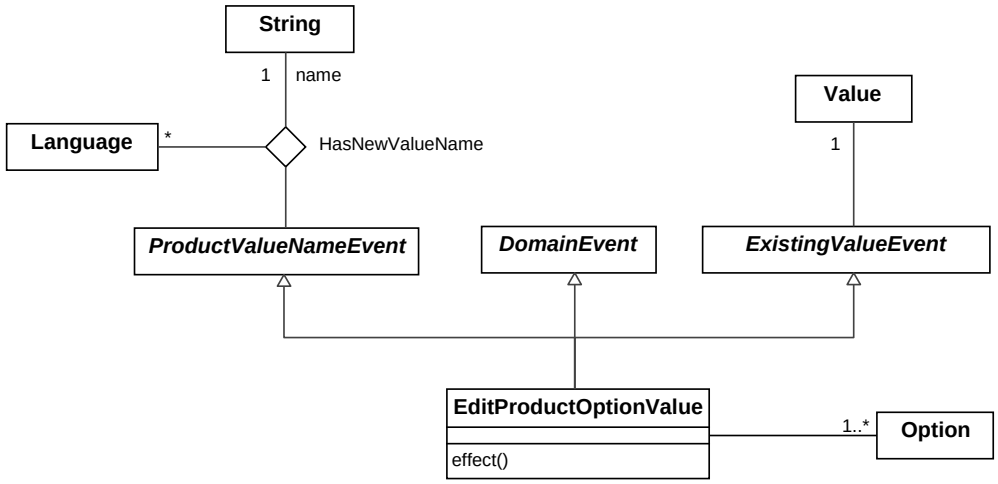
Effect

```
context EditProductOption::effect()
  post :
    Language.allInstances() ->
      forAll (l| self.hasNewOptionName -> select(language=l).name =
                 option.hasOptionName->select(language=l).optionName)
```

Event

EditProductOptionValue

Event diagram



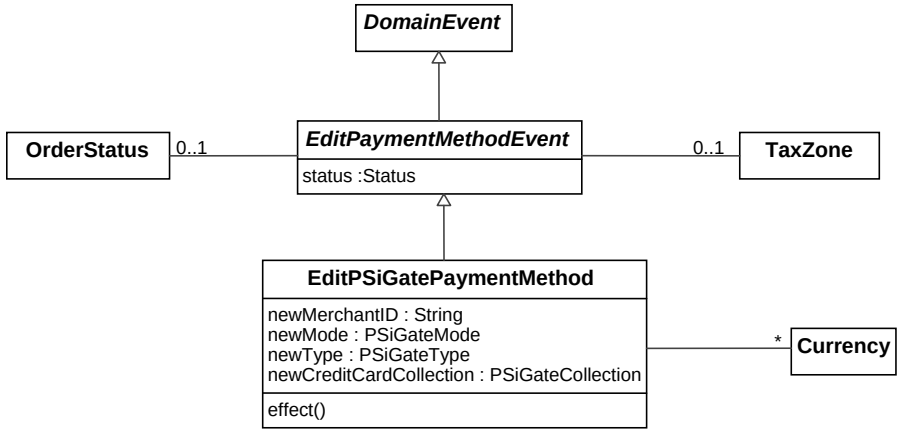
Effect

```
context EditProductOptionValue::effect()
  post :
    Language.allInstances() ->
      forAll (l| self.hasNewValueName -> select(language=l).name =
                 value.hasValueName->select(language=l).valueName) and
    self.value.option = self.option
```

Event

EditPSiGatePaymentMethod

Event diagram



Initial Integrity Constraints

context EditPSiGatePaymentMethod::PaymentMethodIsInstalled():Boolean
body : PSiGate.allInstances() -> notEmpty()

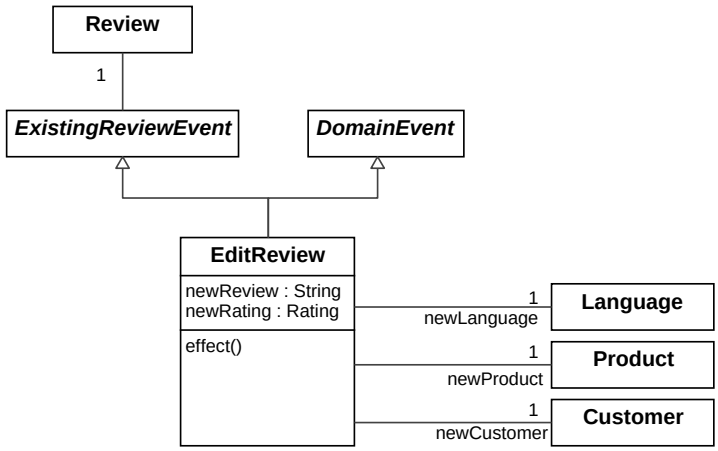
Effect

context EditPSiGatePaymentMethod::effect()
post :
let pm: PSiGate= PSiGate.allInstances() -> any(true) **in**
pm.merchantID=self.newMerchantID **and**
pm.mode=self.newMode **and**
pm.type=self.newType **and**
pm.creditCardCollection=self.newCreditCardCollection **and**
pm.status=self.enabled **and**
pm.orderStatus=self.orderStatus **and**
pm.taxZone=self.taxZone

Event

EditReview

Event diagram



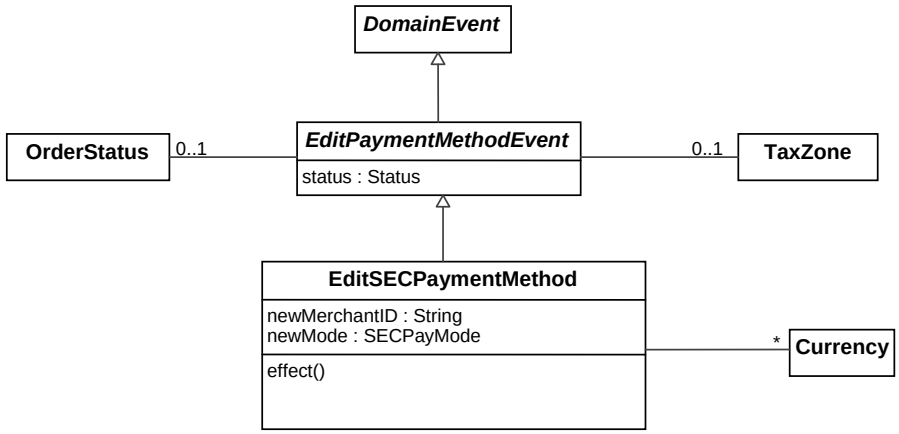
Effect

```
context EditReview::effect()
  post :
     self.review.review = self.newReview and
     self.review.rating = self.newRating and
     self.review.language = self.newLanguage and
     self.review.product = self.newProduct and
     self.review.customer = self.newCustomer
  post :
     self.review.lastModified = Now()
```

Event

EditSECPaymentMethod

Event diagram



Initial Integrity Constraints

context EditSECPaymentMethod::PaymentMethodIsInstalled():Boolean
body : SECPay.allInstances() -> notEmpty()

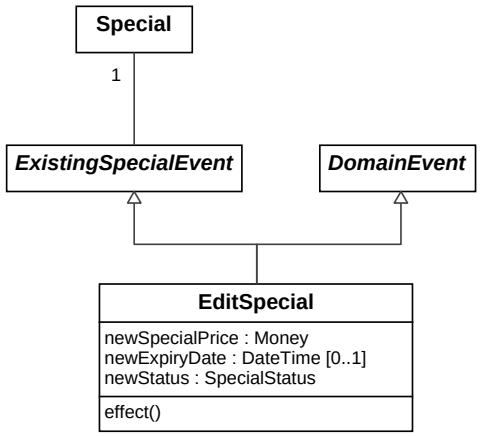
Effect

context EditSECPaymentMethod::effect()
post :
let pm: SECPay= SECPay.allInstances() -> any(true) **in**
pm.merchantID=self.newMerchantID **and**
pm.mode=self.newMode **and**
pm.status=self.status **and**
pm.orderStatus=self.orderStatus **and**
pm.taxZone=self.taxZone

Event

EditSpecial

Event diagram



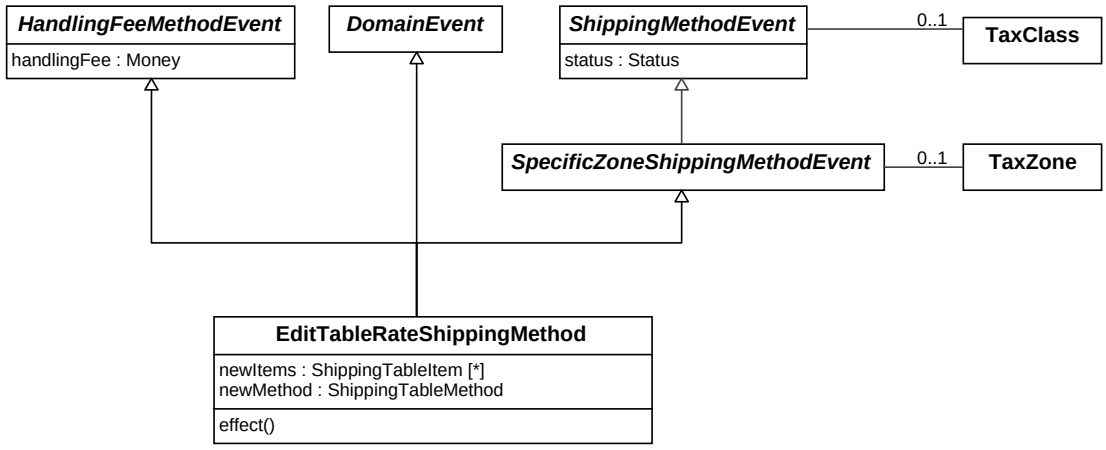
Effect

```
context EditSpecial::effect()
  post :
    self.special.specialPrice = self.newSpecialPrice and
    self.special.expiryDate = self.newExpiryDate and
    self.special.status = self.newStatus
  post :
    self.special.lastModified = Now()
  post :
    self.special@pre.status <> self.newStatus implies
    self.special.dateStatusChanged = Now()
```

Event

EditTableRateShippingMethod

Event diagram



Initial Integrity Constraints

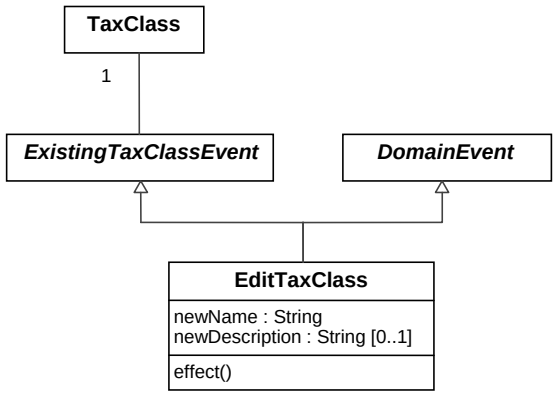
context EditTableRateShippingMethod::PaymentMethodIsInstalled():Boolean
body : TableRate.allInstances() -> notEmpty()

Effect

context EditTableRateShippingMethod::effect()
post :
let sm: TableRate= TableRate.allInstances() -> any(true) **in**
sm.items=self.newItems **and**
sm.method=self.newMethod **and**
sm.handlingFee=self.handlingFee **and**
sm.taxZone=self.taxZone **and**
sm.taxClass=self.taxClass **and**
sm.status = self.status

Event
EditTaxClass

Event diagram

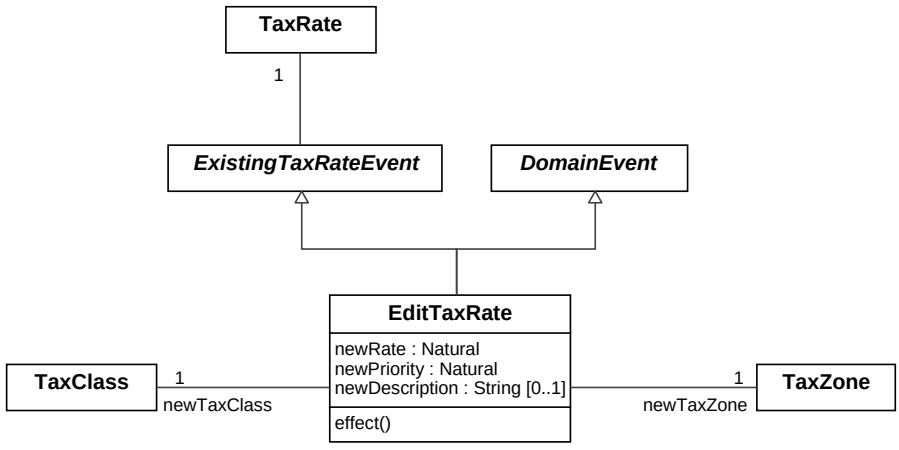


Effect

context EditTaxClass::effect()
post :
self.taxClass.name = self.newName and
self.taxClass.description = self.newDescription

Event
EditTaxRate

Event diagram



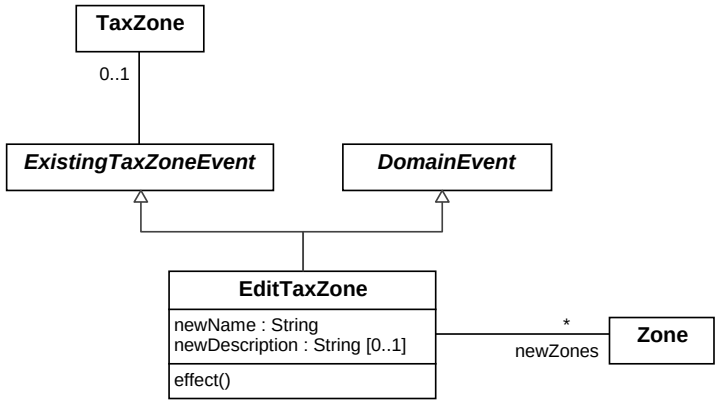
Effect

```
context  EditTaxRate::effect()
  post :
     self.taxRate.rate = self.newRate and
     self.taxRate.priority = self.newPriority and
     self.taxRate.description = self.newDescription and
     self.taxRate.taxClass = self.newTaxClass and
     self.taxRate.taxZone = self.newTaxZone
```

Event

EditTaxZone

Event diagram



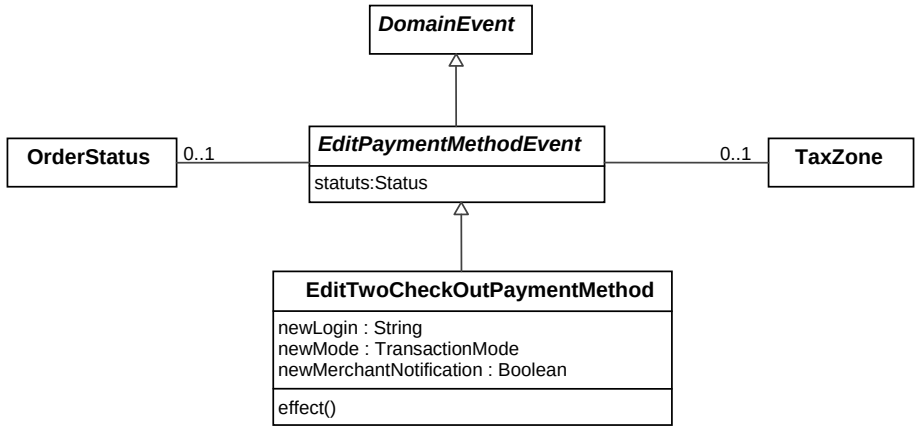
Effect

```
context  EditTaxZone::effect()
  post :
     self.taxZone.name = self.newName and
     self.taxZone.description = self.newDescription and
     self.taxZone.zone = self.newZones
```

Event

EditTwoCheckOutPaymentMethod

Event diagram

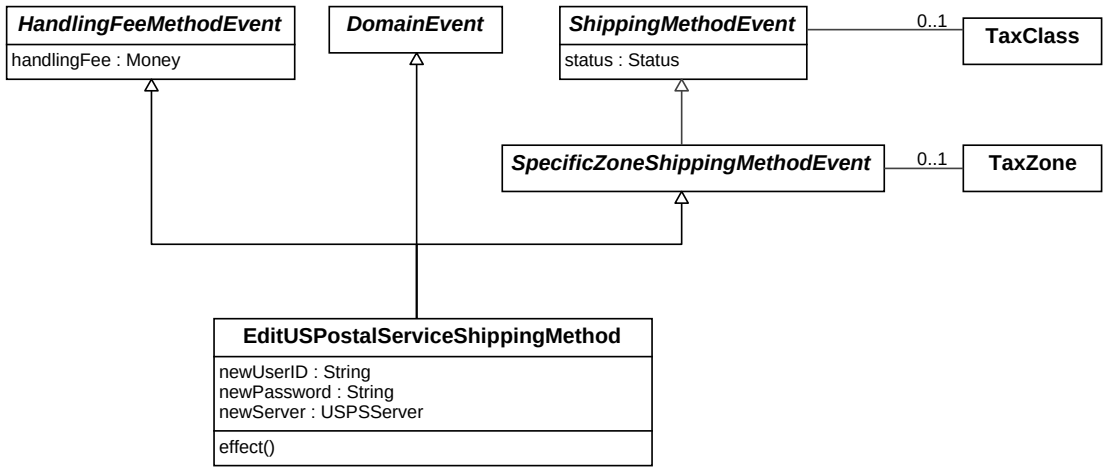



```
context EditTwoCheckOutPaymentMethod::PaymentMethodIsInstalled()
  body : TwoCheckOut.allInstances() -> notEmpty()

context EditTwoCheckOutPaymentMethod::effect()
  post :
    let pm: TwoCheckOut = TwoCheckOut.allInstances() -> any(true)
    in
      pm.login=self.newLogin and
      pm.model=self.newMode and
      pm.merchantNotification=self.newMerchantNotification and
      pm.status=self.status and
      pm.orderStatus=self.orderStatus and
      pm.taxZone=self.taxZone
```

Event

EditUSPostalServiceShippingMethod

Event diagram



Initial Integrity Constraints

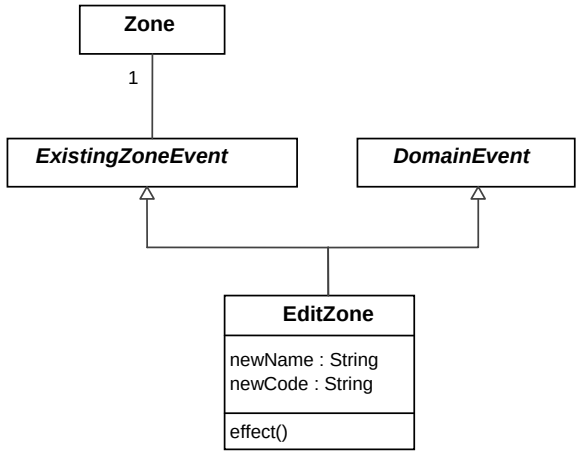
```
context EditUSPostalServiceShippingMethod::PaymentMethodIsInstalled():Boolean
  body : USPostalService.allInstances() -> notEmpty()
```

Effect

```
context EditUSPostalServiceShippingMethod::effect()
  post :
    let sm: USPostalService= USPostalService.allInstances() -> any(true) in
    sm.userID=self.newUserID and
    sm.password=self.newPassword and
    sm.server=self.newServer and
    sm.handlingFee=self.handlingFee and
    sm.taxZone=self.taxZone and
    sm.taxClass=self.taxClass and
    sm.status = self.status
```

Event
EditZone

Event diagram

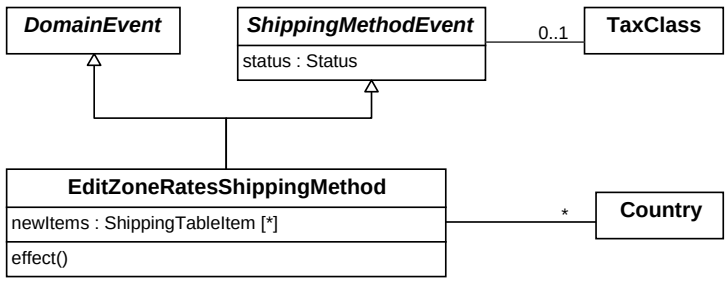


Effect

context EditZone::effect()
post :
self.zone.name = self.newName **and**
self.zone.code = self.newCode

Event
EditZoneRatesShippingMethod

Event diagram



Initial Integrity Constraints

```
context EditZoneRatesShippingMethod::PaymentMethodIsInstalled():Boolean
  body : ZoneRates.allInstances() -> notEmpty()
```

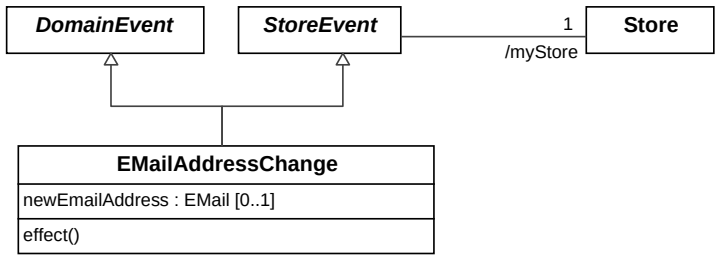
Effect

```
context EditZoneRatesShippingMethod::effect()
  post :
    let sm:ZoneRates= ZoneRates.allInstances() -> any(true) in
    sm.items=self.newItems and
    sm.country=self.country and
    sm.taxClass=self.taxClass and
    sm.status=self.status
```

Event

EMailAddressChange

Event diagram

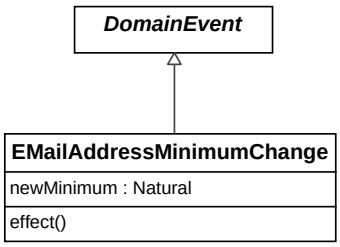


Effect

```
context EMailAddressChange::effect()
  post : myStore.eMailAddress = self.newEmailAddress
```

Event
EMailAddressMinimumChange

Event diagram

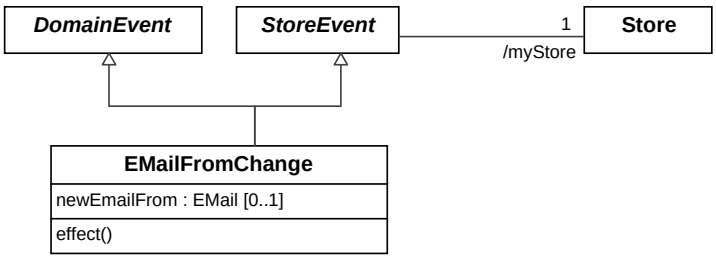


Effect

context EMailAddressMinimumChange::effect()
post : MinimumValues.eMailAddress = self.newMinimum

Event
EMailFromChange

Event diagram



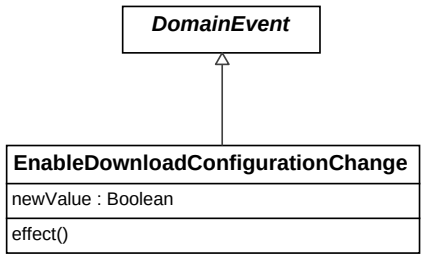
Effect

context EMailFromChange::effect()
post : myStore.eMailFrom = self.newEmailFrom

Event

EnableDownloadConfigurationChange

Event diagram



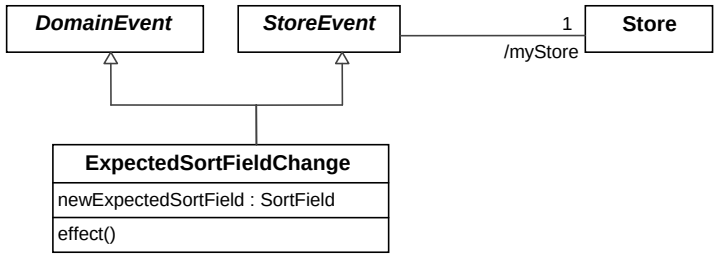
Effect

context EnableDownloadConfigurationChange::effect()
post : Download.enableDownload= self.newValue

Event

ExpectedSortFieldChange

Event diagram



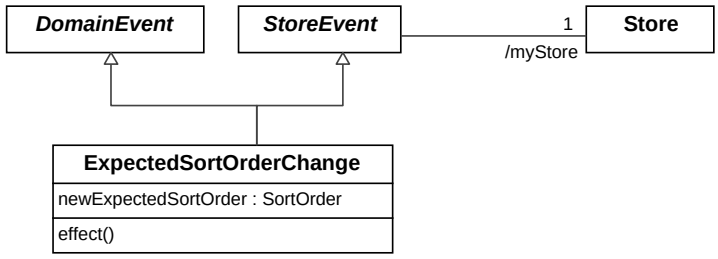
Effect

context ExpectedSortFieldChange::effect()
post : myStore.expectedSortField = self.newExpectedSortField

Event

ExpectedSortOrderChange

Event diagram



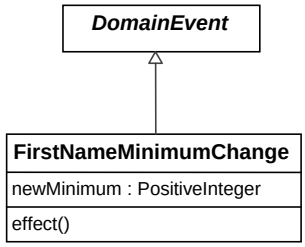
Effect

context ExpectedSortOrderChange::effect()
post : myStore.expectedSortOrder = self.newExpectedSortOrder

Event

FirstNameMinimumChange

Event diagram



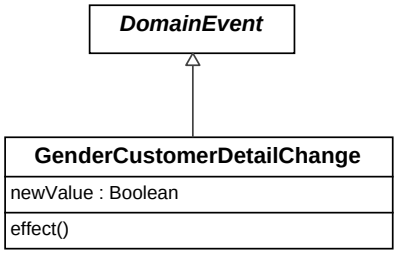
Effect

context FirstNameMinimumChange::effect()
post : MinimumValues.firstName = self.newMinimum

Event

GenderCustomerDetailChange

Event diagram



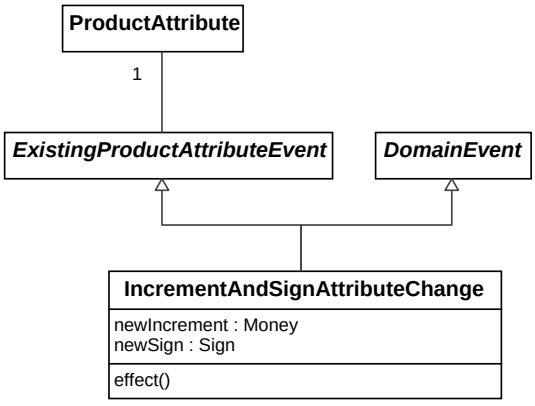
Effect

context GenderCustomerDetailChange::effect()
post : CustomerDetails.gender = self.newValue

Event

IncrementAndSignAttributeChange

Event diagram



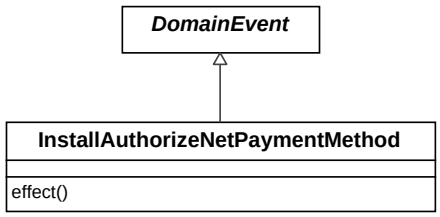
Effect

context IncrementAndSignAttributeChange::effect()
post : self.productAttribute.increment = self.newIncrement **and**
self.productAttribute.sign = self.newSign

Event

InstallAuthorizeNetPaymentMethod

Event diagram



Initial Integrity Constraints

context InstallAuthorizeNetPaymentMethod::PaymentMethodIsNotInstalled():Boolean
body : AuthorizeNet.allInstances() -> isEmpty()

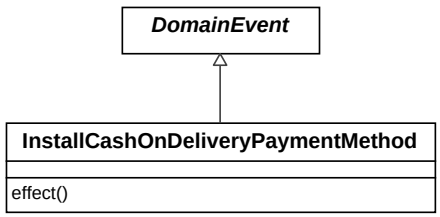
Effect

context InstallAuthorizeNetPaymentMethod::effect()
post : pm.oclIsNew() **and** pm.oclIsTypeOf(AuthorizeNet)

Event

InstallCashOnDeliveryPaymentMethod

Event diagram



Initial Integrity Constraints

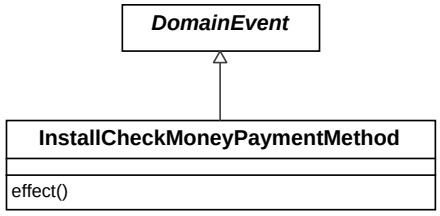
context InstallCashOnDeliveryPaymentMethod::PaymentMethodIsNotInstalled():Boolean
body : CashOnDelivery.allInstances() -> isEmpty()

Effect

context InstallCashOnDeliveryPaymentMethod::effect()
post : pm.oclIsNew() **and** pm.oclIsTypeOf(CashOnDelivery)

Event
InstallCheckMoneyPaymentMethod

Event diagram



Initial Integrity Constraints

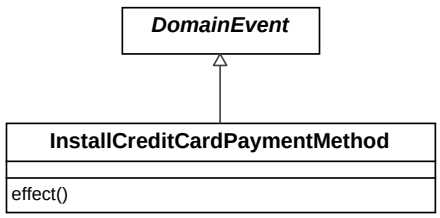
context InstallCheckMoneyPaymentMethod::PaymentMethodIsNotInstalled():Boolean
body : CheckMoney.allInstances() -> isEmpty()

Effect

context InstallCheckMoneyPaymentMethod::effect()
post : pm.oclIsNew() **and** pm.oclIsTypeOf(CheckMoney)

Event
InstallCreditCardPaymentMethod

Event diagram



Initial Integrity Constraints

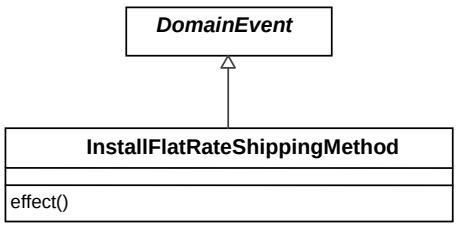
context InstallCreditCardPaymentMethod::PaymentMethodIsNotInstalled():Boolean
body : CreditCard.allInstances() -> isEmpty()

Effect

context InstallCreditCardPaymentMethod::effect()
post : pm.oclIsNew() **and** pm.oclIsTypeOf(CreditCard)

Event
InstallFlatRateShippingMethod

Event diagram



Initial Integrity Constraints

context InstallFlatRateShippingMethod::ShippingMethodIsNotInstalled():Boolean
body : FlatRate.allInstances() -> isEmpty()

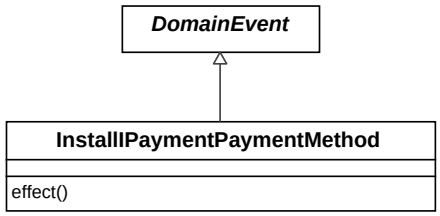
Effect

context InstallFlatRateShippingMethod::effect()
post : sm.oclIsNew() **and** sm.oclIsTypeOf(FlatRate)

Event

InstallIPaymentPaymentMethod

Event diagram



Initial Integrity Constraints

context InstallIPaymentPaymentMethod::PaymentMethodIsNotInstalled():Boolean
body : IPayment.allInstances() -> isEmpty()

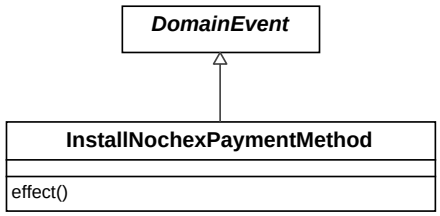
Effect

context InstallIPaymentPaymentMethod::PaymentMethodIsNotInstalled()
post : sm.oclIsNew() **and** sm.oclIsTypeOf(IPayment)

Event

InstallNochexPaymentMethod

Event diagram



Initial Integrity Constraints

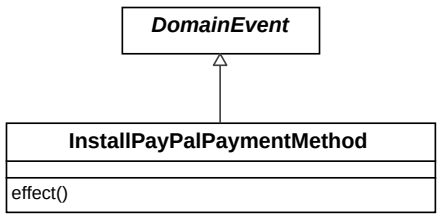
context InstallNochexPaymentMethod::PaymentMethodIsNotInstalled():Boolean
body : Nochex.allInstances() -> isEmpty()

Effect

context InstallNochexPaymentMethod::effect()
post : pm.oclIsNew() **and** pm.oclIsTypeOf(Nochex)

Event
InstallPayPalPaymentMethod

Event diagram



Initial Integrity Constraints

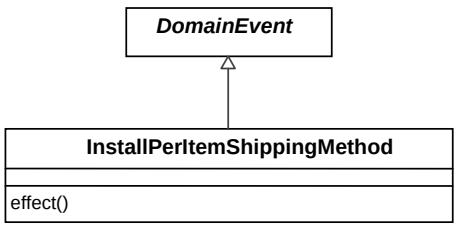
context InstallPayPalPaymentMethod::PaymentMethodIsNotInstalled():Boolean
body : PayPal.allInstances() -> isEmpty()

Effect

context InstallPayPalPaymentMethod::effect()
post : pm.oclIsNew() **and** pm.oclIsTypeOf(PayPal)

Event
InstallPerItemShippingMethod

Event diagram



Initial Integrity Constraints

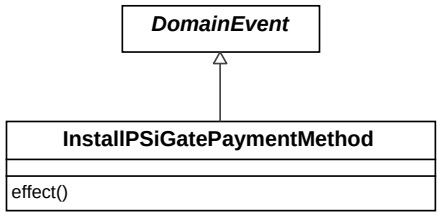
context InstallPerItemShippingMethod::ShippingMethodIsNotInstalled():Boolean
body : PerItem.allInstances() -> isEmpty()

Effect

context InstallPerItemShippingMethod::effect()
post : sm.oclIsNew() **and** sm.oclIsTypeOf(PerItem)

Event
InstallPSiGatePaymentMethod

Event diagram



Initial Integrity Constraints

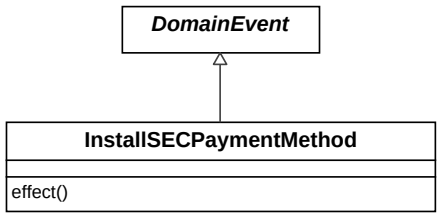
context InstallPSiGatePaymentMethod::PaymentMethodIsNotInstalled():Boolean
body : PSiGate.allInstances() -> isEmpty()

Effect

context InstallPSiGatePaymentMethod::effect()
post : pm.oclIsNew() **and** pm.oclIsTypeOf(PSiGate)

Event
InstallSECPaymentMethod

Event diagram



Initial Integrity Constraints

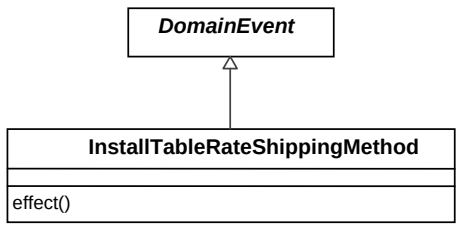
context InstallSECPaymentMethod::PaymentMethodIsNotInstalled():Boolean
body : SECPay.allInstances() -> isEmpty()

Effect

context InstallSECPaymentMethod::effect()
post : pm.oclIsNew() **and** pm.oclIsTypeOf(SECPay)

Event
InstallTableRateShippingMethod

Event diagram



Initial Integrity Constraints

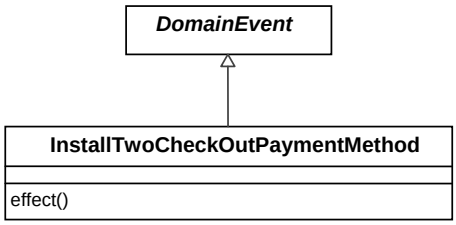
context InstallTableRateShippingMethod::ShippingMethodIsNotInstalled():Boolean
body : TableRate.allInstances() -> isEmpty()

Effect

context InstallTableRateShippingMethod::effect()
post : sm.oclIsNew() **and** sm.oclIsTypeOf(TableRate)

Event
InstallTwoCheckOutPaymentMethod

Event diagram



Initial Integrity Constraints

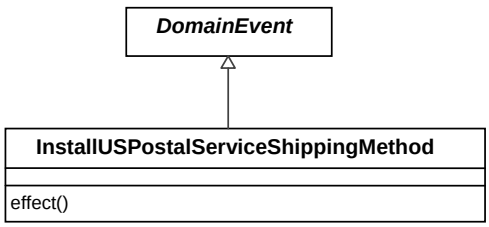
context InstallTwoCheckOutPaymentMethod::PaymentMethodIsNotInstalled():Boolean
body : TwoCheckOut.allInstances() -> isEmpty()

Effect

context InstallTwoCheckOutPaymentMethod::effect()
post : pm.oclIsNew() **and** pm.oclIsTypeOf(TwoCheckOut)

Event
InstallUSPostalServiceShippingMethod

Event diagram



Initial Integrity Constraints

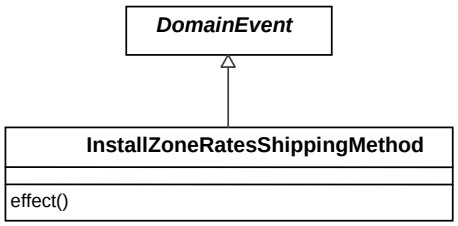
context InstallUSPostalServiceShippingMethod::ShippingMethodIsNotInstalled():Boolean
body : USPostalService.allInstances() -> isEmpty()

Effect

context InstallUSPostalServiceShippingMethod::effect()
post : sm.oclIsNew() **and** sm.oclIsTypeOf(USPostalService)

Event
InstallZoneRatesShippingMethod

Event diagram



Initial Integrity Constraints

context InstallZoneRatesShippingMethod::ShippingMethodIsNotInstalled():Boolean
body : ZoneRates.allInstances() -> isEmpty()

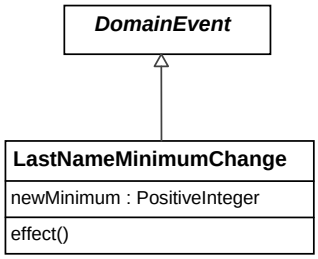
Effect

context InstallZoneRatesShippingMethod::effect()
post : sm.oclIsNew() **and** sm.oclIsTypeOf(ZoneRates)

Event

LastNameMinimumChange

Event diagram



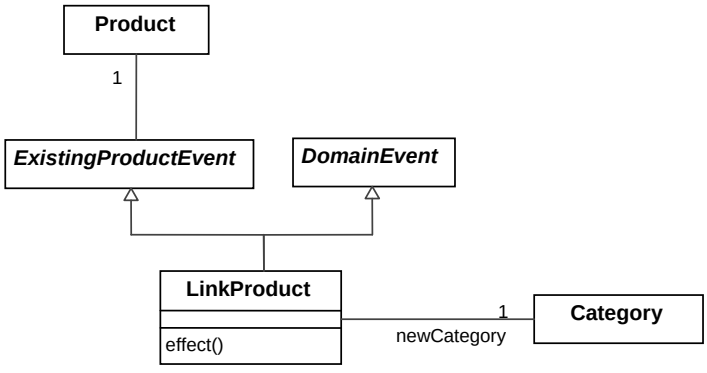
Effect

context LastNameMinimumChange::effect()
post : MinimumValues.lastName = self.newMinimum

Event

LinkProduct

Event diagram

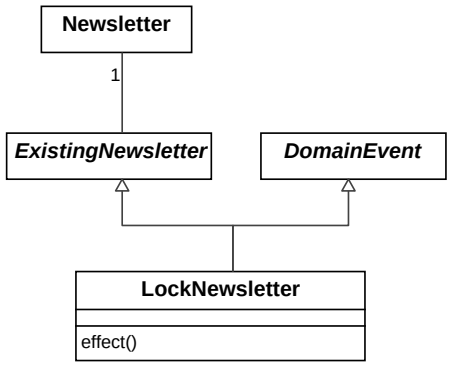


Effect

context LinkProduct::effect()
post: self.product.category -> includes(self.newCategory)

Event
LockNewsletter

Event diagram

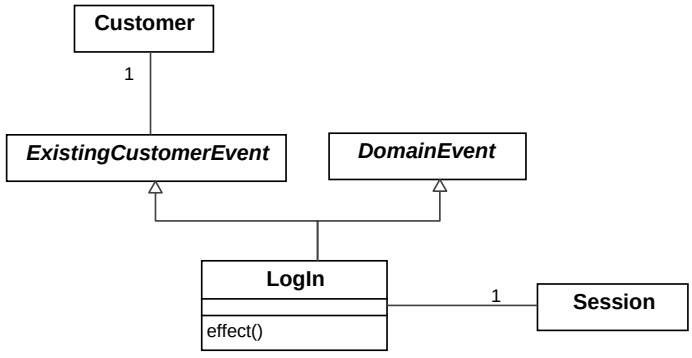


Effect

context LockNewsletter::effect()
post : self.newsletter.status = NewsletterStatus::locked

Event
LogIn

Event diagram



Initial Integrity Constraints

context LogIn::CustomerIsNotLoggedIn (): Boolean
body : self.customer.session -> isEmpty()

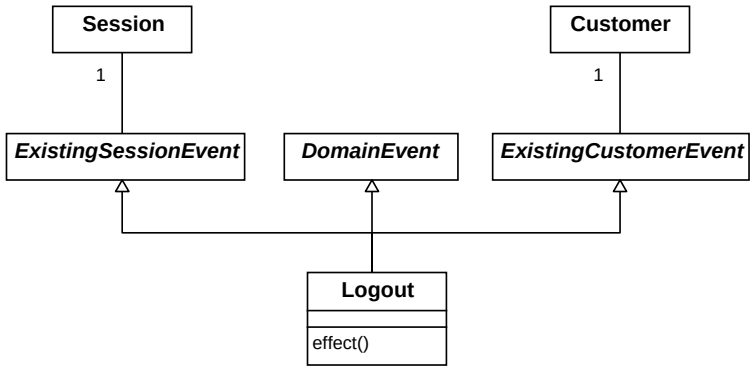
Effect

```
context  LogIn::effect()
  post :
    self.session.customer = self.customer
  post :
    self.customer.lastLogon = Now() and
    self.customer.numberOfLogons = self.customer.numberOfLogons@pre + 1
```

Event

LogOut

Event diagram



Initial Integrity Constraints

```
context LogIn::CustomerIstLoggedIn (): Boolean
  body : self.session.customer = self.customer
```

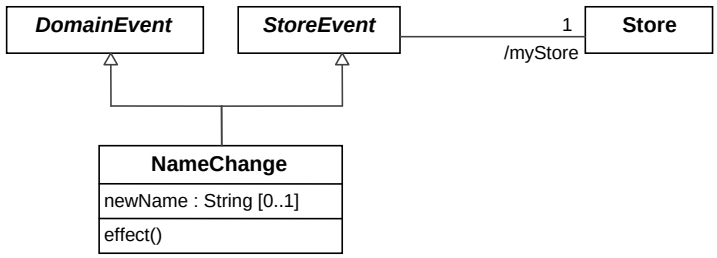
Effect

```
context  LogOut::effect()
  post :  self.session.customer -> isEmpty()
```

Event

NameChange

Event diagram



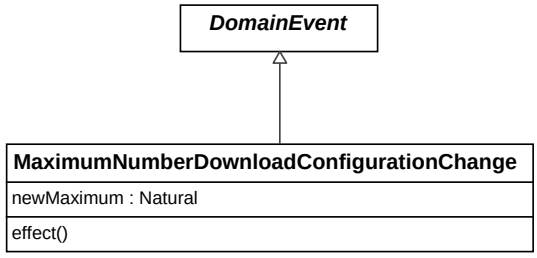
Effect

context NameChange::effect()
post : self.myStore.name = self.newName

Event

MaximumNumberDownloadConfigurationChange

Event diagram



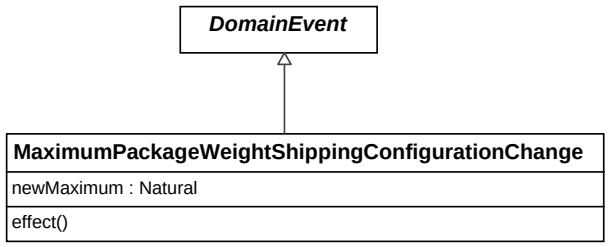
Effect

context MaximumNumberDownloadConfigurationChange::effect()
post : Download.maximumNumberOfDownloads= self.newMaximum

Event

MaximumPackageWeightShippingConfigurationChange

Event diagram



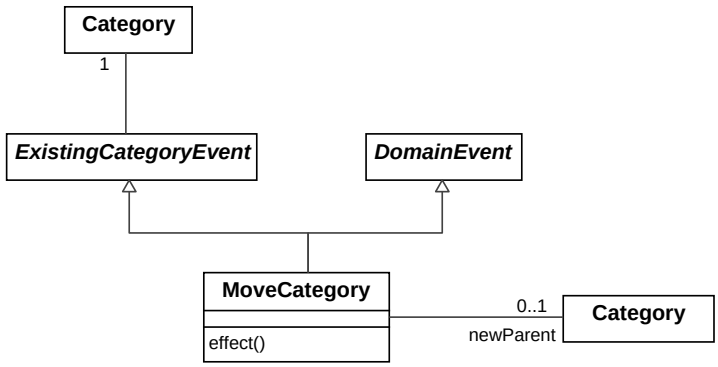
Effect

context MaximumPackageWeightShippingConfigurationChange::effect()
post : ShippingAndPackaging.maximumPackageWeight = self.newMaximum

Event

MoveCategory

Event diagram

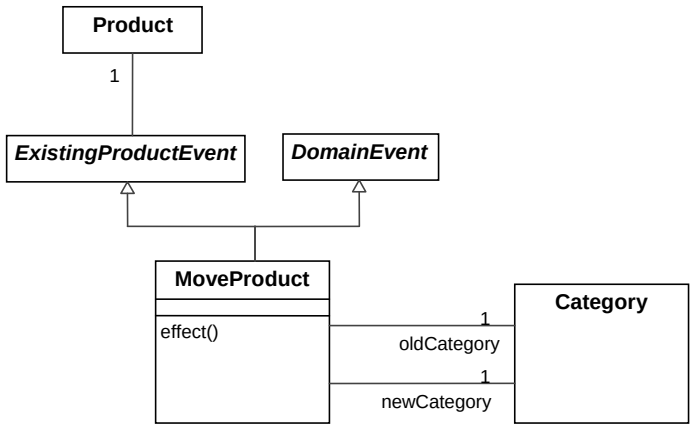


Effect

context MoveCategory::effect()
post : self.category.parent = self.newParent

Event
MoveProduct

Event diagram

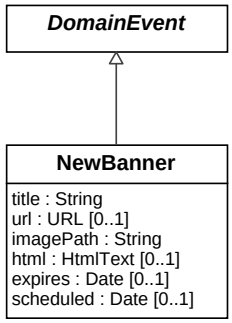


Effect

context MoveProduct::effect()
post: self.product.category -> includes(self.newCategory) **and**
self.product.category -> excludes(self.oldCategory)

Event
NewBanner

Event diagram



Initial Integrity Constraints

context NewBanner::bannerDoesNotExist(): Boolean
body : not Banner.allInstances() ->exists (b | b.title= self.title)

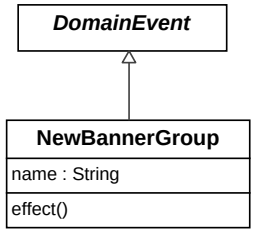
Effect

context NewBanner::effect()
post :
b.oclIsNew() and
b.oclIsTypeOf(Banner) and
b.title = self.title and
b.url = self.url and
b.imagePath = self.imagePath and
b.html = self.html and
b.expires = self.expires and
b.scheduled = self.scheduled and
b.status = BannerStatus::enabled

Event

NewBannerGroup

Event diagram



Initial Integrity Constraints

context NewBannerGroup::bannerGroupDoesNotExist(): Boolean
body : not BannerGroup.allInstances() ->exists (bg | bg.name= self.name)

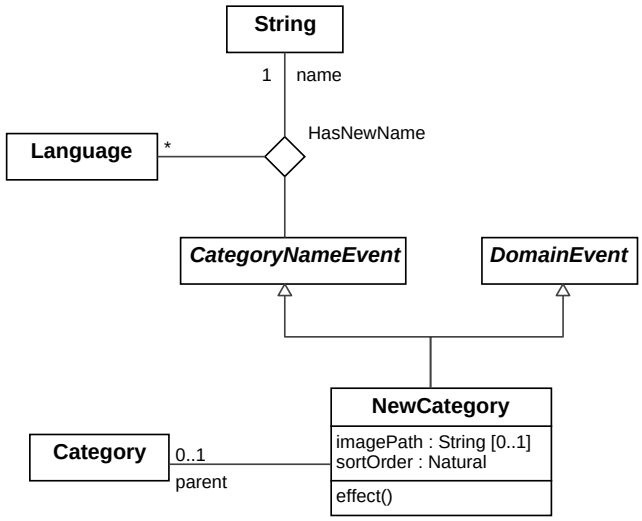
Effect

context NewBannerGroup::effect()
post :
bg.oclIsNew() and
bg.oclIsTypeOf(BannerGroup) and
bg.name = self.name

Event

NewCategory

Event diagram



Initial Integrity Constraints

```
context NewCategory::categoryDoesNotExist(): Boolean
 body :
    Language.allInstances() -> forAll ( l |
            l.hasCategoryName.categoryName ->
            excludes(self.hasNewName->select(language=l).name))
```

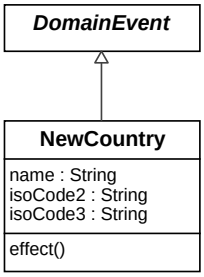
Effect

```
context  NewCategory::effect()
  post :
     c.oclIsNew()  and
     c.oclIsTypeOf(Category) and
     c.imagePath = self.imagePath and
     c.sortOrder = self.sortOrder and
     c.parent = self.parent and
     Language.allInstances() ->
        forAll (l| self.hasNewName -> select(language=l).name =
                   c.hasCategoryName->select(language=l).categoryName)
```

Event

NewCountry

Event diagram



Initial Integrity Constraints

```
context NewCountry::countryDoesNotExist(): Boolean
 body :
    not Country.allInstances() -> exists(c | c.name=self.name and
                                             c.isoCode2=self.isoCode2 and
                                             c.isoCode3=self.isoCode3)
```

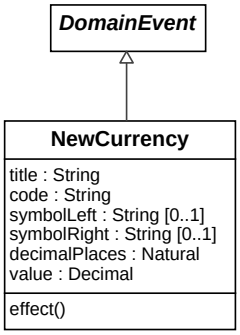
Effect

```
context  NewCountry::effect()
  post :
     c.oclIsNew()  and
     c.oclIsTypeOf(Country) and
     c.name = self.name and
     c.isoCode2 = self.isoCode2 and
     c.isoCode3 = self.isoCode3
```

Event

NewCurrency

Event diagram



Initial Integrity Constraints

```
context NewCurrency::currencyDoesNotExist(): Boolean
  body :
     not Currency.allInstances() -> exists(c | c.title=self.title and
                                              c.code=self.code)
```

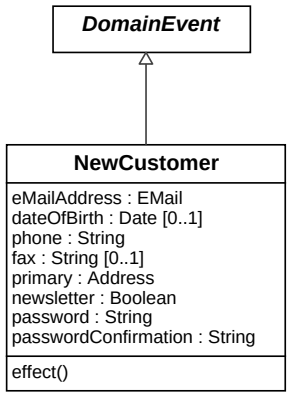
Effect

```
context  NewCurrency::effect()
   post :
      c.oclIsNew()  and
      c.oclIsTypeOf(Currency) and
      c.title = self.title and
      c.code = self.code and
      c.symbolLeft = self.symbolLeft and
      c.symbolRight = self.symbolRight and
      c.decimalPlaces = self.decimalPlaces and
      c.value = self.value
```

Event

NewCustomer

Event diagram



Initial Integrity Constraints

context NewCustomer::customerDoesNotExist(): Boolean
body : **not** Customer.allInstances() -> exists (c | c.eMailAddress = self.eMailAddress)

context NewCustomer::passwordCorrect(): Boolean
body : password = passwordConfirmation

context NewCustomer::firstNameRight(): Boolean
body : self.primary.firstName.size() >= MinimumValues.firstName

context NewCustomer::lastNameRight(): Boolean
body : self.primary.lastName.size() >= MinimumValues.lastName

context NewCustomer::dateOfBirthRight(): Boolean
body :
CustomerDetails.dateOfBirth **implies**
self.dateOfBirth -> notEmpty() **and**
self.dateOfBirth.size() >= MinimumValues.dateOfBirth

context NewCustomer::genderRight(): Boolean
body : CustomerDetails.gender **implies** self.gender->notEmpty()

context NewCustomer::suburbRight(): Boolean
body : CustomerDetails.suburb **implies** self.suburb->notEmpty()

context NewCustomer::eMailRight(): Boolean
body : self.eMailAddress.size() >= MinimumValues.eMailAddress

context NewCustomer::streetAddressRight(): Boolean
body : self.primary.street.size() >= MinimumValues.streetAddress

context NewCustomer::companyRight(): Boolean
body :

```
      CustomerDetails.company implies
      self.primary.company -> notEmpty() and
      self.primary.company.size() >= MinimumValues.companyName

context NewCustomer::postCodeRight(): Boolean
  body : self.primary.postCode.size() >= MinimumValues.postCode

context NewCustomer::cityRight(): Boolean
  body : self.primary.city.size() >= MinimumValues.city

context NewCustomer::stateRight(): Boolean
  body :
      CustomerDetails.state implies
      self.primary.state -> notEmpty() and
      self.primary.state.size() >= MinimumValues.state

context NewCustomer::telephoneRight(): Boolean
  body : self.telephone.size() >= MinimumValues.telephoneNumber

context NewCustomer::passwordRight(): Boolean
  body : self.password.size() >= MinimumValues.password
```

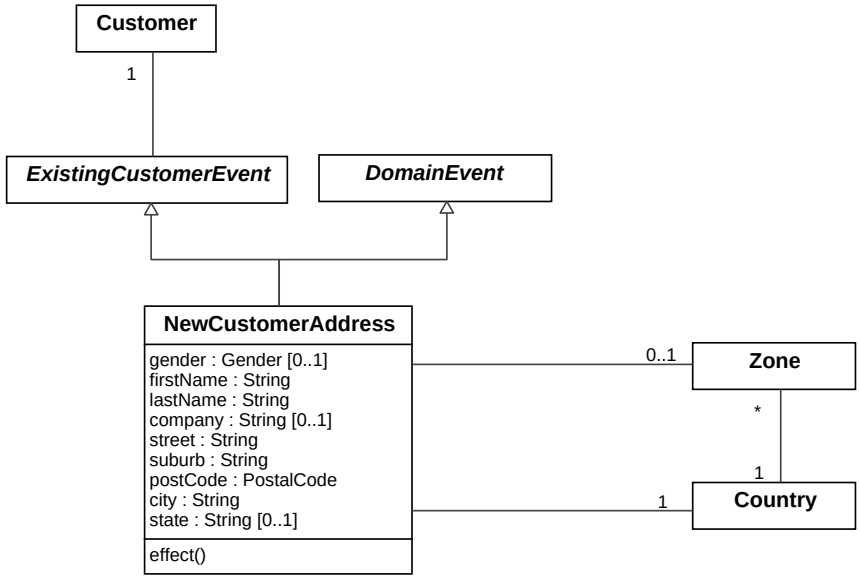
Effect

```
context  NewCustomer::effect()
   post :
      c.oclIsNew() and
      c.oclIsTypeOf(Customer) and
      c.gender = self.primary.gender and
      c.firstName = self.primary.firstName and
      c.lastName = self.primary.lastName and
      c.dateOfBirth = self.dateOfBirth and
      c.eMailAddress = self.eMailAddress and
      c.phone = self.phone and
      c.fax = self.fax and
      c.newsletter = self.newsletter and
      c.password = self.password and
      c.numberOfLogons = 0 and
      c.address = Set{primary}  and
      c.primary = primary
```

Event

NewCustomerAddress

Event diagram



Initial Integrity Constraints

context NewCustomerAddress::firstNameRight(): Boolean
body : self.primary.firstName.size() >= MinimumValues.firstName

context NewCustomerAddress::lastNameRight(): Boolean
body : self.primary.lastName.size() >= MinimumValues.lastName

context NewCustomerAddress::genderRight(): Boolean
body : CustomerDetails.gender **implies** self.gender->notEmpty()

context NewCustomerAddress::suburbRight(): Boolean
body : CustomerDetails.suburb **implies** self.suburb->notEmpty()

context NewCustomerAddress::streetAddressRight(): Boolean
body : self.primary.street.size() >= MinimumValues.streetAddress

context NewCustomerAddress::companyRight(): Boolean
body :
CustomerDetails.company **implies**
self.primary.company -> notEmpty() **and**
self.primary.company.size() >= MinimumValues.companyName

context NewCustomerAddress::postCodeRight(): Boolean

```
  body : self.primary.postCode.size() >= MinimumValues.postCode

context NewCustomerAddress::cityRight(): Boolean
  body : self.primary.city.size() >= MinimumValues.city

context NewCustomerAddress::stateRight(): Boolean
  body :
      CustomerDetails.state implies
      self.primary.state -> notEmpty() and
      self.primary.state.size() >= MinimumValues.state

context NewCustomerAddress::addressesHaveZoneIfNeeded(): Boolean
  body :
     self.zone -> notEmpty() implies
     self.state = self.zone.name and
     self.country = self.zone.country

context NewCustomerAddress::numberOfAddressesRight(): Boolean
  body : self.customer.address -> size() < MaximumValues.addressBookEntries
```

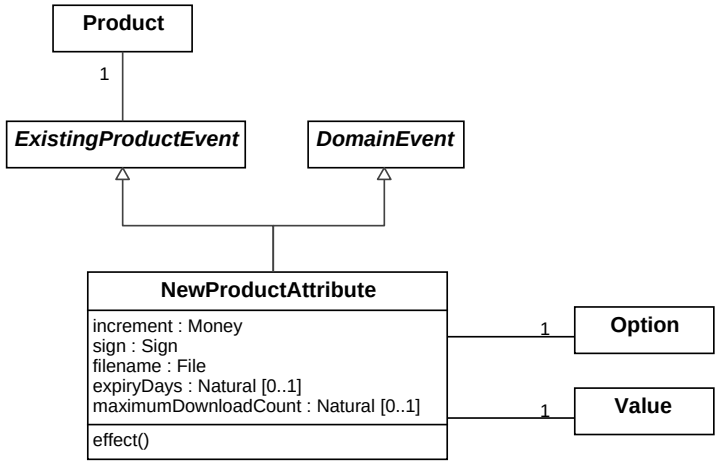
Effect

```
context  NewCustomerAddress::effect()
   post :
         Address.allInstances() ->exists (a |
         a.gender = self.gender and
         a.firstName = self.firstName and
         a.lastName = self.lastName and
         a.company = self.company and
         a.street = self.street and
         a.suburb = self.suburb and
         a.postCode = self.postCode and
         a.city = self.city and
         a.state = self.state and
         a.zone = self.zone and
         a.country = self.country and
         self.customer.address -> includes(a))
```

Event

NewDownloadableProductAttribute

Event diagram



Initial Integrity Constraints

```
context NewDownloadableProductAttribute::productAttributeDoesNotExist(): Boolean
 body :
    not ProductAttribute.allInstances() -> exists (pa | pa.attribute.option = self.option and
                                                         pa.attribute.value = self.value and
                                                         pa.product = self.product)
```

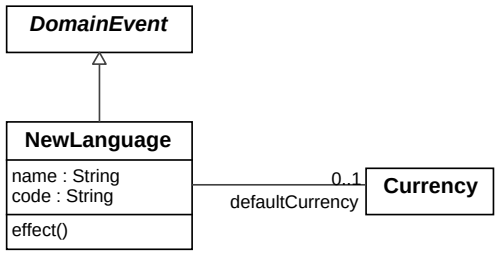
Effect

```
context  NewDownloadableProductAttribute::effect()
  post :
    dpa.oclIsNew()  and
    dpa.oclIsTypeOf(Downloadable) and
    dpa.increment = self.increment and
    dpa.sign = self.sign and
    dpa.filename = self.filename and
    dpa.product = self.product and
    dpa.attribute.option=self.option and
    dpa.attribute.value=self.value and
    if self.expiryDays.notEmpty() then dpa.expiryDays = self.expiryDays
    else self.expiryDays = Download.daysExpiryDelay
    endif
    and
    if self.maximumDownloadCount .notEmpty() then
       dpa.maximumDownloadCount = self.maximumDownloadCount
    else self.maximumDownloadCount = Download.maximumNumberOfDownload
    endif
```

Event

NewLanguage

Event diagram



Initial Integrity Constraints

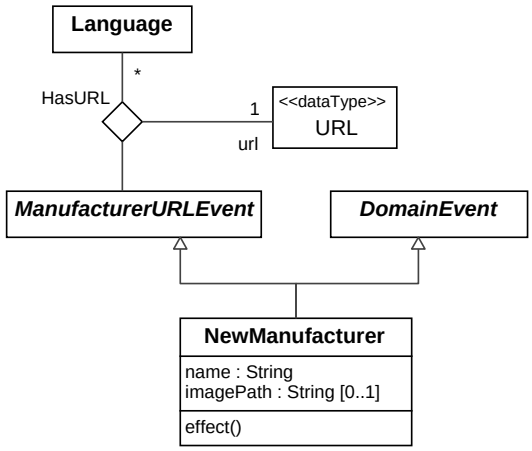
```
context NewLanguage::languageDoesNotExist(): Boolean
 body :
    not Language.allInstances() -> exists (l | l.name=self.name and
                                              l.code = self.code)
```

Effect

```
context  NewLanguage::effect()
  post :
     l.oclIsNew()  and
     l.oclIsTypeOf(Language) and
     l.name = self.name and
     l.code = self.code and
     l.defaultCurrency = self.defaultCurrency
```

NewManufacturer

Event diagram



Initial Integrity Constraints

```
context NewManufacturer::manufacturerDoesNotExist(): Boolean
  body :
     not Manufacturer.allInstances() -> exists (m | m.name=self.name)
```

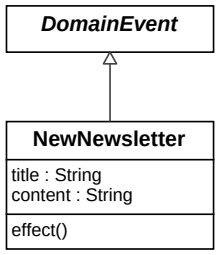
Effect

```
context  NewManufacturer::effect()
   post :
      m.oclIsNew()  and
      m.oclIsTypeOf(Manufacturer) and
      m.name = self.name and
      m.imagePath = self.imagePath and
      Language.allInstances() ->
         forAll (l|
           self.hasURL -> select(language=l).url =
           m.manufacturerInLanguage->select(language=l).url)
```

Event

NewNewsletter

Event diagram



Initial Integrity Constraints

```
context NewNewsletter::newsletterDoesNotExist(): Boolean
 body :  not Newsletter.allInstances() -> exists (n | n.title=self.title)
```

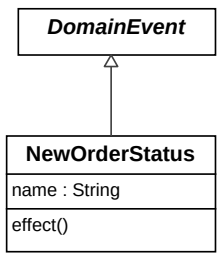
Effect

```
context  NewNewsletter::effect()
  post :
     n.oclIsNew()  and
     n.oclIsTypeOf(Newsletter) and
     n.title = self.title and
     n.content = self.content and
     n.status = NewsletterStatus::unlocked
```

Event

NewOrderStatus

Event diagram



Initial Integrity Constraints

```
context NewOrderStatus::orderStatusDoesNotExist(): Boolean
 body :
    not OrderStatus.allInstances() -> exists (os | os.name=self.name)
```

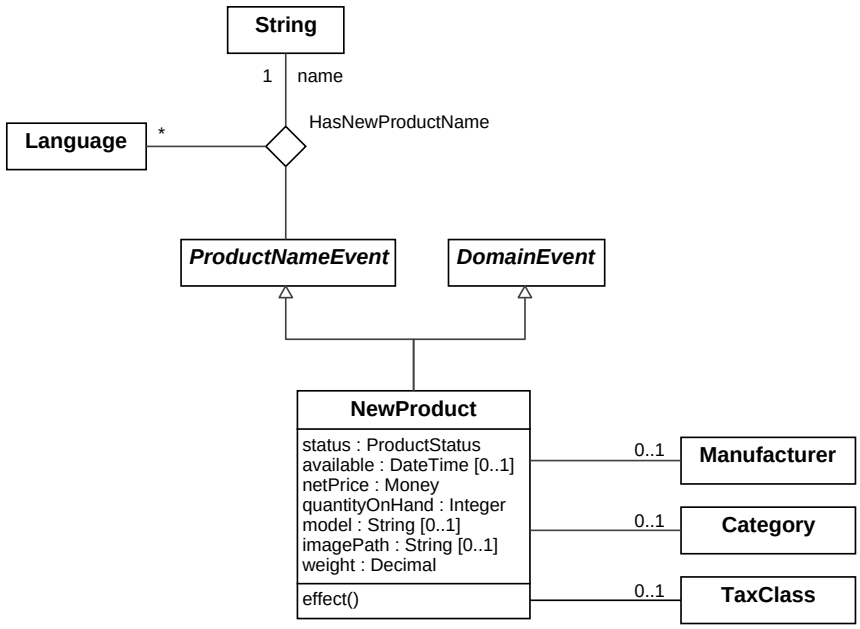
Effect

```
context  NewOrderStatus::effect()
  post :
     os.oclIsNew()  and
     os.oclIsTypeOf(OrderStatus) and
     os.name = self.name
```

Event

NewProduct

Event diagram



Initial Integrity Constraints

```
context NewProduct::productDoesNotExist(): Boolean
 body :
    Language.allInstances() -> forAll ( l |
          l.productInLanguage.name
          -> excludes( self.hasNewProductName -> select(language=l).name))
```

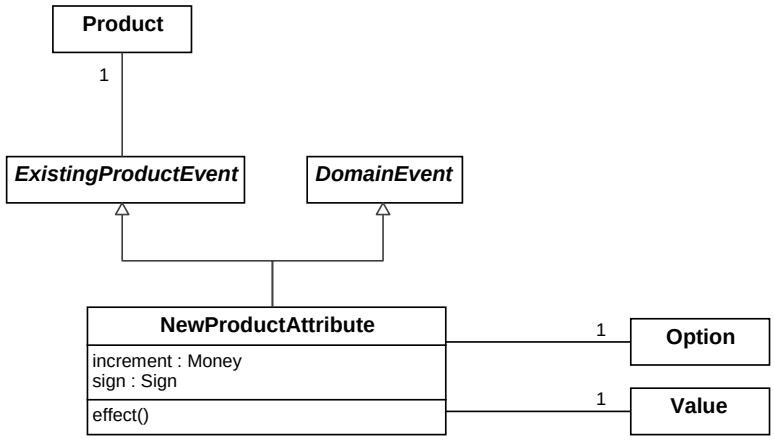
Effect

```
context  NewProduct::effect()
  post :
     p.oclIsNew()  and
     p.oclIsTypeOf(Product) and
     p.status = self.status and
     p.available = self.available and
     p.netPrice = self.netPrice and
     p.quantityOnHand = self.quantityOnHand and
     p.model = self.model and
     p.imagePath = self.imagePath and
     p.weight = self.weight and
     p.category = Set{self.category} and
     p.manufacturer = self.manufacturer and
     p.taxClass = self.taxClass and
     Language.allInstances() ->
        forAll (l |
          self.hasNewProductName -> select(language=l).name =
          p.productInLanguage->select(language=l).name)
```

Event

NewProductAttribute

Event diagram



Initial Integrity Constraints

```
context NewProductAttribute::productAttributeDoesNotExist(): Boolean
  body :
     not self.product.productAttribute ->
              exists(attribute.value=self.value and
                 attribute.option = self.option)
```

```
context NewProductAttribute::optionValueIsValid(): Boolean
  body :  self.option.value -> includes(self.value)
```

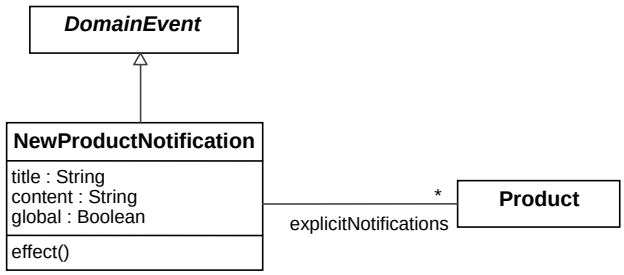
Effect

```
context  NewProductAttribute::effect()
   post :
      pa.oclIsNew()  and
      pa.oclIsTypeOf(ProductAttribute) and
      pa.increment = self.increment and
      pa.sign = self.sign and
      pa.product = self.product and
      pa.attribute.option = self.option and
      pa.attribute.value = self.value
```

Event

NewProductNotification

Event diagram



Initial Integrity Constraints

```
context NewProductNotification::ProductNotificationDoesNotExist(): Boolean
  body : not Newsletter.allInstances() -> exists (n | n.title = self.title)
```

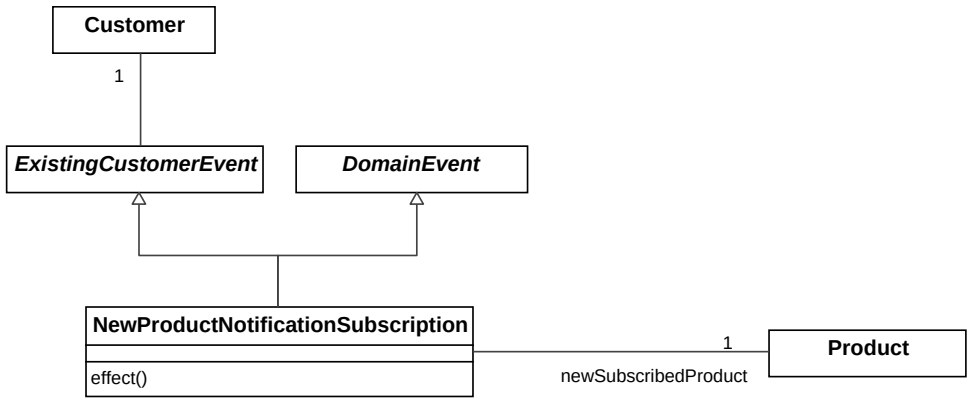
Effect

```
context  NewProductNotification::effect()
   post :
      n.oclIsNew()  and
      n.oclIsTypeOf(ProductNotification) and
      n.title = self.title and
      n.content = self.content and
      n.global = self.global and
      n.explicitNotifications = self.explicitNotifications and
      n.status = self.NewsletterStatus::unlocked
```

Event

NewProductNotificationSubscription

Event diagram



Initial Integrity Constraints

context NewProductNotificationSubscription::ProductIsUnsubscribed(): Boolean
body :
not self.customer.globalNotifications **and**
self.customer.explicitNotifications -> excludes(self.newSubscribedProduct)

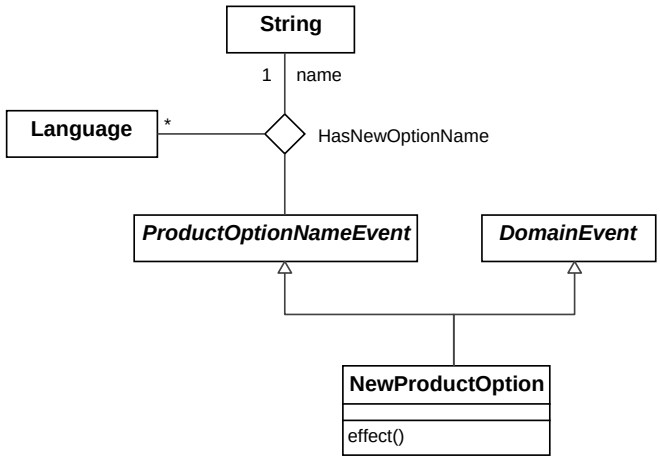
Effect

context NewProductNotificationSubscription::effect()
post : self.customer.explicitNotifications -> includes(self.newSubscribedProduct)

Event

NewProductOption

Event diagram



Initial Integrity Constraints

```
context NewProductOption::productOptionDoesNotExist(): Boolean
 body :
    Language.allInstances() -> forAll ( l |
          l.hasOptionName.optionName
          -> excludes(self.hasNewOptionName -> select(language=l).name))
```

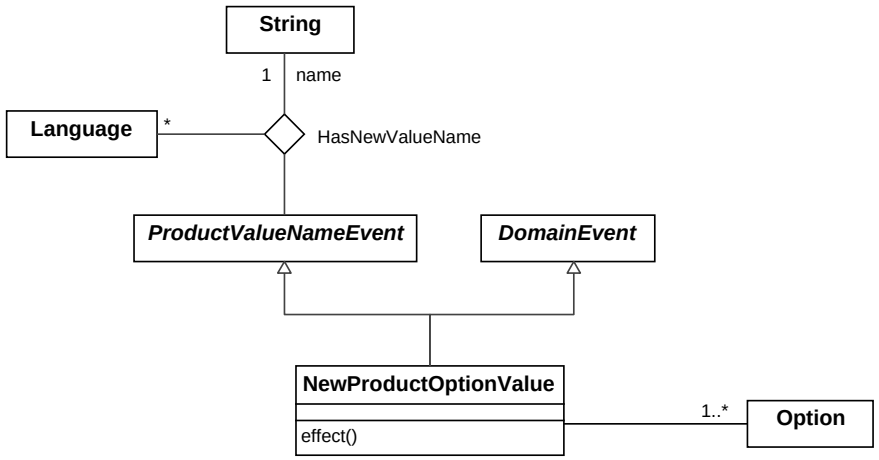
Effect

```
context  NewProductOption::effect()
  post :
     po.oclIsNew()  and
     po.oclIsTypeOf(Option) and
     Language.allInstances() ->
        forAll (l| self.hasNewOptionName -> select(language=l).name =
                   po.hasOptionName->select(optionLanguage=l).optionName)
```

Event

NewProductOptionValue

Event diagram



Initial Integrity Constraints

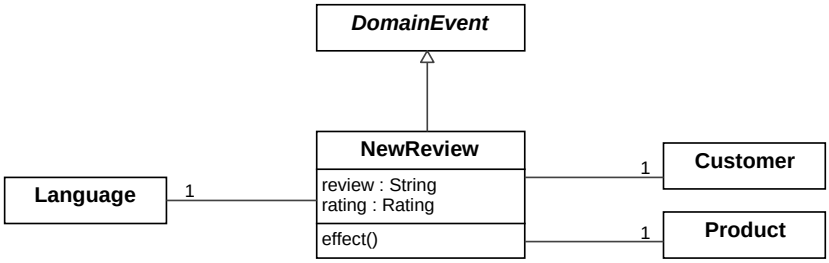
```
context NewProductOptionValue::optionValueDoesNotExist(): Boolean
 body :
    Language.allInstances() -> forAll ( l |
          l.hasValueName.valueName
          -> excludes(self.hasNewValueName -> select(Language=l).name))
```

Effect

```
context  NewProductOptionValue::effect()
  post :
     pov.oclIsNew()  and
     pov.oclIsTypeOf(Value) and
     Language.allInstances() ->
        forAll (l| self.hasNewValueName -> select(language=l).name =
                   pov.hasValueName->select(valueLanguage=l).valueName) and
     pov.option = self.option
```

Event
NewReview

Event diagram



Initial Integrity Constraints

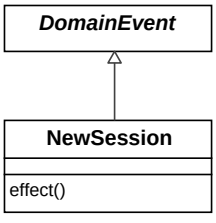
context NewReview::reviewRight(): Boolean
body : self.review.size() >= MinimumValues.reviewText

Effect

context NewReview::effect()
post :
r.oclIsNew() **and**
r.oclIsTypeOf(Review) **and**
r.review = self.review **and**
r.rating = self.rating **and**
r.customer = self.customer **and**
r.product = self.product **and**
r.language = self.language

Event
NewSession

Event diagram



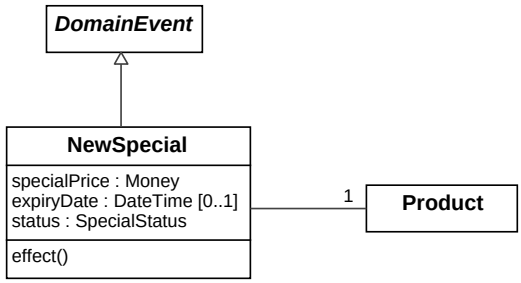
Effect

```
context  NewSession::effect()
  post :
     s.oclIsNew() and
     s.oclIsTypeOf(Session)
```

Event

NewSpecial

Event diagram



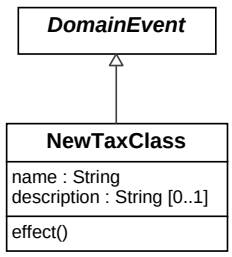
Effect

```
context NewSpecial::effect()
  post :
     self.product.oclIsTypeOf(Special) and
     self.product.oclAsTypeOf(Special).specialPrice=self.specialPrice and
     self.product.oclAsTypeOf(Special).expiryDate=self.expiryDate and
     self.product.oclAsTypeOf(Special).status=self.status
```

Event

NewTaxClass

Event diagram



Initial Integrity Constraints

```
context NewTaxClass::TaxClassDoesNotExist(): Boolean
 body :  not TaxClass.allInstances() -> exists (tc | tc.name = self.name)
```

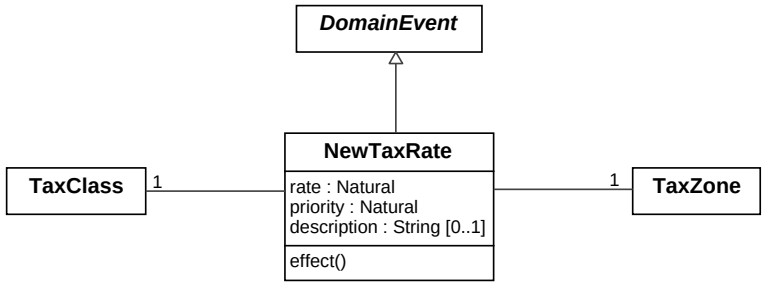
Effect

```
context  NewTaxClass::effect()
  post :
     tc.oclIsNew()  and
     tc.oclIsTypeOf(TaxClass) and
     tc.name = self.name and
     tc.description = self.description
```

Event

NewTaxRate

Event diagram



Initial Integrity Constraints

```
context NewTaxRate::TaxRateDoesNotExist(): Boolean
 body :
    not TaxRate.allInstances() -> exists (tr | tr.taxClass = self.taxClass and
                                               tr.taxZone = self.taxZone)
```

Effect

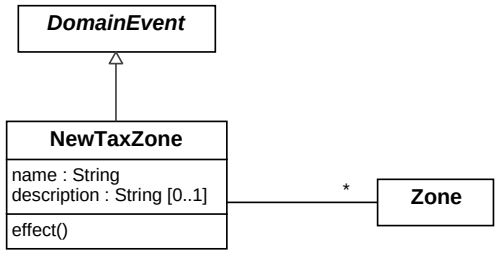
```
context  NewTaxRate::effect()
  post :
     tr.oclIsNew()  and
     tr.oclIsTypeOf(TaxRate) and
     tr.rate = self.rate and
     tr.priority = self.priority and
```

tr.description = self.description **and**
tr.taxClass = self.taxClass **and**
tr.taxZone = self.taxZone

Event

NewTaxZone

Event diagram



Initial Integrity Constraints

context NewTaxZone::TaxZoneDoesNotExist(): Boolean
body : **not** TaxZone.allInstances() -> exists (tz | tz.name = self.name)

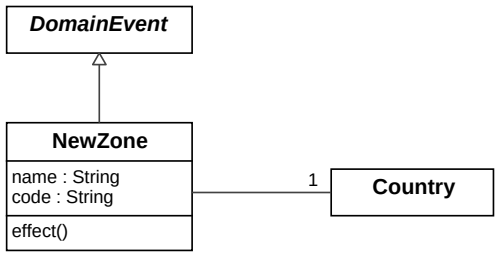
Effect

context NewTaxZone::effect()
post :
tz.oclIsNew() **and**
tz.oclIsTypeOf(TaxZone) **and**
tz.name = self.name **and**
tz.description = self.description **and**
tz.zone = self.zone

Event

NewZone

Event diagram



Initial Integrity Constraints

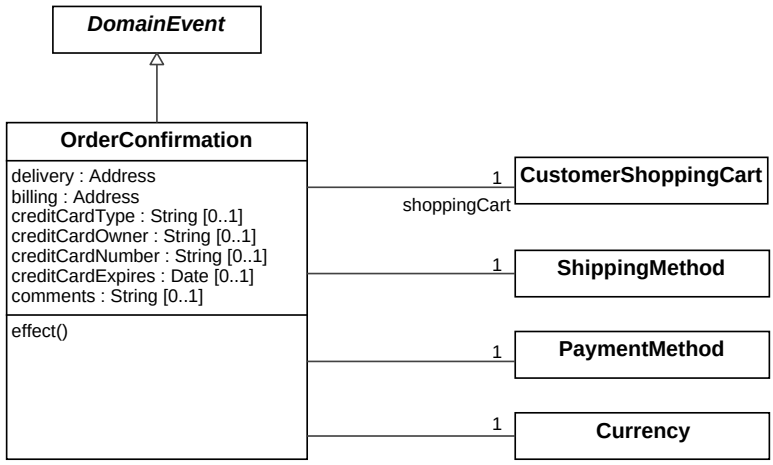
```
context NewZone::ZoneDoesNotExist(): Boolean
 body :
    not Zone.allInstances() -> exists (z | z.name = self.name and z.country = self.country or
                                          z.code = self.code and z.country = self.country)
```

Effect

```
context  NewZone::effect()
  post :
     z.oclIsNew()  and
     z.oclIsTypeOf(Zone) and
     z.name = self.name and
     z.code = self.code and
     z.country = self.country
```

Event
OrderConfirmation

Event diagram



Initial Integrity Constraints

context OrderConfirmation::ShippingMethodIsEnabled(): Boolean
body : self.shippingMethod.status= Status::enabled

context OrderConfirmation::PaymentMethodIsEnabled(): Boolean
body : self.paymentMethod.status= Status::enabled

context OrderConfirmation::CurrencyIsEnabled(): Boolean
body : self.currency.status = Status::enabled

context OrderConfirmation::CreditCardDetailsNeeded(): Boolean
body :
self.paymentMethod.oclIsTypeOf(AuthorizeNet) **or**
self.paymentMethod.oclIsTypeOf(CreditCard) **or**
self.paymentMethod.oclIsTypeOf(IPayment) **or**
self.paymentMethod.oclIsTypeOf(TwoCheckOut) **or**
self.paymentMethod.oclIsTypeOf(PSiGate)
implies
creditCardType.notEmpty() **and**
creditCardOwner.notEmpty() **and**
creditCardNumber.notEmpty() **and**
creditCardExpires.notEmpty()

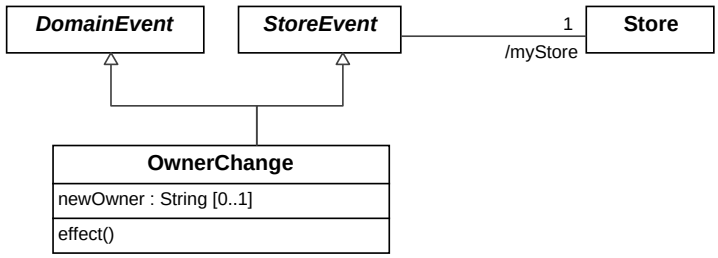
context OrderConfirmation::StockAllowsOrder(): Boolean
body :
Stock.allowCheckout **or**
not Stock.checkStockLevel **or**
self.shoppingCart.shoppingCartItem.product -> forAll (p | p.quantityOnHand > 0)

Effect

```
context  OrderConfirmation::effect()
  post theOrderIsCreated:
     o.oclIsNew() and
     o.oclIsTypeOf(Order) and
     o.customer = self.shoppingCart@pre.customer@pre and
     o.billing = self.billing and
     o.delivery = self.delivery and
     o.shippingMethod = self.shippingMethod and
     o.paymentMethod = self.paymentMethod and
     o.currency = self.currency and
     --The initial status of the order is pending
     osc.oclIsNew() and
     osc.oclIsTypeOf(OrderStatusChange) and
     osc.comments = self.comments and
     osc.orderStatus = Store.allInstances() -> any(true).defaultStatus and
     osc.order = o and
     --There is an order line for each shopping cart item
    shoppingCart@pre.shoppingCartItem@pre->forAll
      (i|OrderLine.allInstances() -> one
        (ol|ol.order = o  and
            ol.product = i.product@pre  and
            ol.quantity = i.quantity@pre  and
            i.attribute@pre->forAll
              (iAtt|OrderLineAttribute.allInstances() -> one
              (olAtt|olAtt.orderLine = ol and
                      olAtt.attribute = iAtt))))
  post theShoppingCartIsRemoved:
     not self.shoppingCart@pre.oclIsKindOf(OclAny)
  post updateProductQuantities:
     let productsBought:Set(Product) =
          self.shoppingCart@pre.shoppingCartItem@pre.product@pre->asSet()
     in  productsBought -> forAll (p|
           let quantityBought:PositiveInteger =
                self.shoppingCart@pre.shoppingCartItem@pre->select
                  (sc | sc.product = p).quantity -> sum()
           in
                p.quantityOrdered = p.quantityOrdered@pre + quantityBought  and
                Stock.substractStock implies
                p.quantityOnHand = p.quantityOnHand@pre – quantityBought)
```

Event
OwnerChange

Event diagram

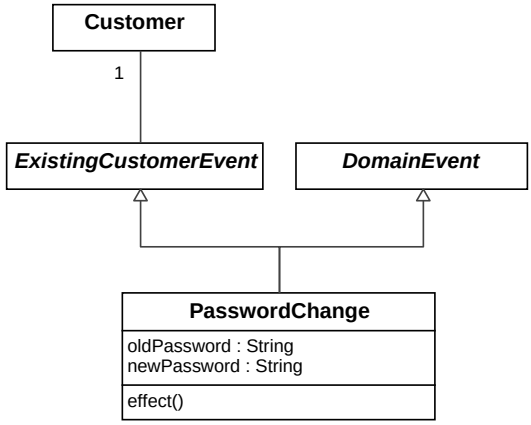


Effect

context OwnerChange::effect()
post : myStore.owner = self.newOwner

Event
PasswordChange

Event diagram



Initial Integrity Constraints

context ChangePassword::passwordRight(): Boolean
body : self.password.size() >= MinimumValues.password

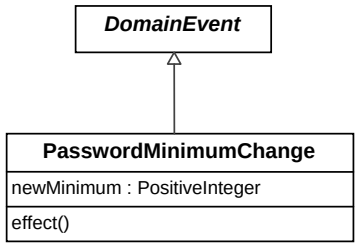
context ChangePassword::OldPasswordIsCorrect(): Boolean
body : customer@pre.password = self.oldPassword

Effect

context ChangePassword::effect()
post : self.customer.password = self.newPassword

Event
PasswordMinimumChange

Event diagram

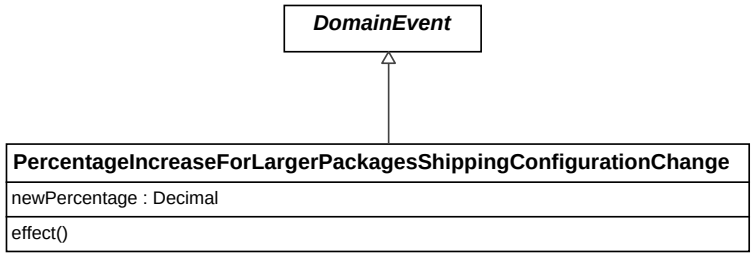


Effect

context PasswordMinimumChange::effect()
post : MinimumValues.password = self.newMinimum

Event
PercentageIncreaseForLargerPackagesShippingConfigurationChange

Event diagram

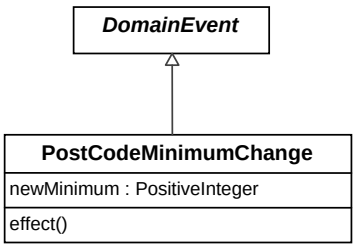


Effect

context PercentageIncreaseForLargerPackagesShippingConfigurationChange::effect()
post : ShippingAndPackaging.percentageIncreaseForLargerPackages= self.newPercentage

Event
PostCodeMinimumChange

Event diagram

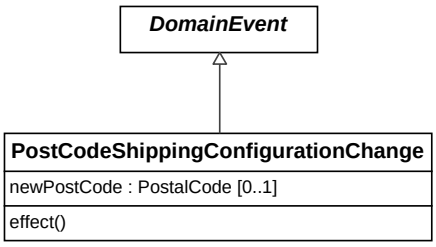


Effect

context PostCodeMinimumChange::effect()
post : MinimumValues.postCode = self.newMinimum

Event
PostCodeShippingConfigurationChange

Event diagram



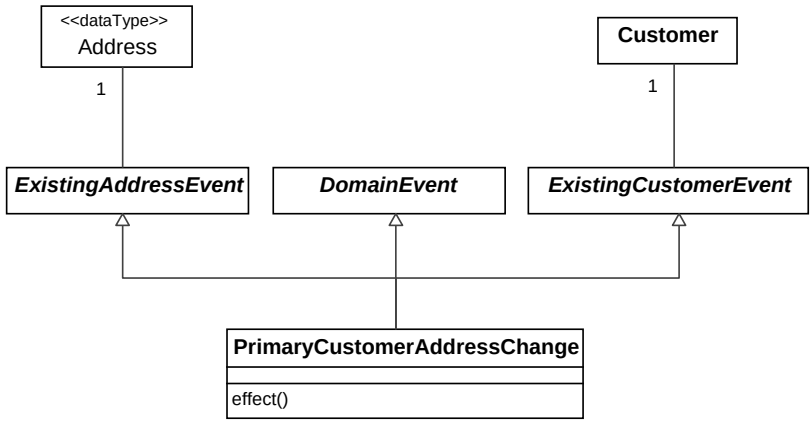
Effect

context PostCodeShippingConfigurationChange::effect()
post : ShippingAndPackaging.postCode = self.newPostCode

Event

PrimaryCustomerAddressChange

Event diagram



Initial Integrity Constraints

context PrimaryCustomerAddressChange::AddressOfCustomer(): Boolean
body : self.customer.address -> includes(self.address)

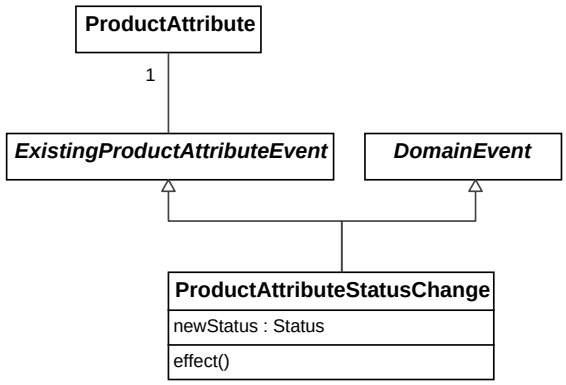
Effect

context PrimaryCustomerAddressChange::effect()
post : self.customer.primary = self.address

Event

ProductAttributeStatusChange

Event diagram



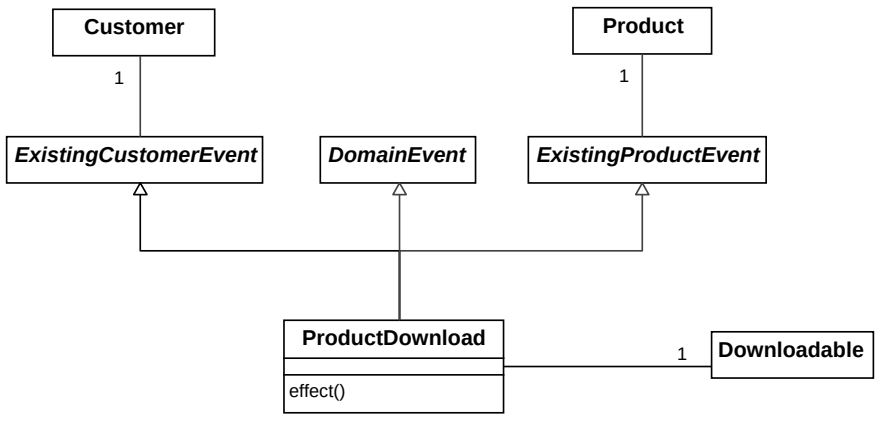
Effect

context ProductAttributeStatusChange::effect()
post : self.productAttribute.status = self.newStatus

Event

ProductDownload

Event diagram



Initial Integrity Constraints

```
context ProductDownload::DownloadEnabled(): Boolean
 body :  Download.enableDonwload

context ProductDownload::ProductWasPurchasedByCustomer(): Boolean
 body :  self.customer.order.orderLine.product -> includes (self.product)

context ProductDownload::DownloadableIsFromProduct(): Boolean
 body :  self.product.productAttribute -> select(pa | pa.oclIsTypeOf(Downloadable))
          -> includes (self.downloadable)

context ProductDownload::DownloadIsNotExpired(): Boolean
 body :
    let datePurchased:DateTime =
        self.customer.order
           -> select (o | o.orderLine.product -> includes(self.product))
              ->sortedBy(purchased)->last().purchased
    in
       Now() <= datePurchased + self.downloadable.expiryDays

context ProductDownload::DownloadsCountNotExceeded(): Boolean
 body :
    let DownloadCountFromProduct:Natural =
        self.customer.order.orderLine.orderLineAttribute
           -> select (ola | ola.oclIsTypeOf(OrderDownload) and ola.orderLine.product = self.product)
          -> sortedBy(orderLine.order.purchased)->last().oclAsType(OrderDownload).downloadCount
    in
       DownloadCountFromProduct < self.downloadable.maximumDownloadCount
```

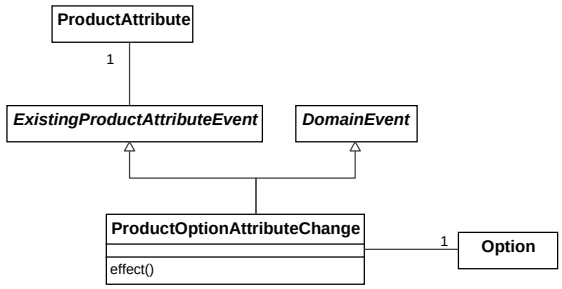
Effect

```
context  ProductDownload::effect()
 post :
    let OrderDownloadFromProduct:OrderDownload=
        self.customer.order.orderLine.orderLineAttribute
           -> select (ola | ola.oclIsTypeOf(OrderDownload) and ola.orderLine.product = self.product)
           -> sortedBy(orderLine.order.purchased) -> last()
          .oclAsType(OrderDownload)

    in
    let OldOrderDownloadCount:Integer =
        self.customer.order.orderLine.orderLineAttribute@pre
           -> select (ola | ola.oclIsTypeOf(OrderDownload) and ola.orderLine.product = self.product)
           -> sortedBy(orderLine.order.purchased) -> last()
          .oclAsType(OrderDownload).downloadCount
    in
      OrderDownloadFromProduct.downloadCount = OldOrderDownloadCount +1
```

Event

ProductOptionAttributeChange

Event diagram



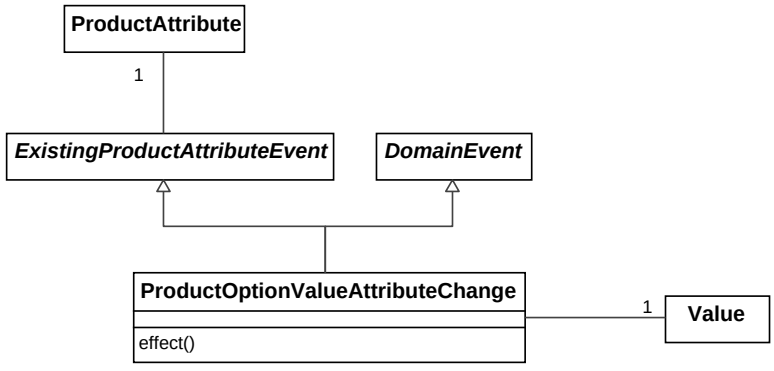
Effect

context ProductOptionAttributeChange::effect()
post : productAttribute.attribute.option = self.option

Event

ProductOptionValueAttributeChange

Event diagram

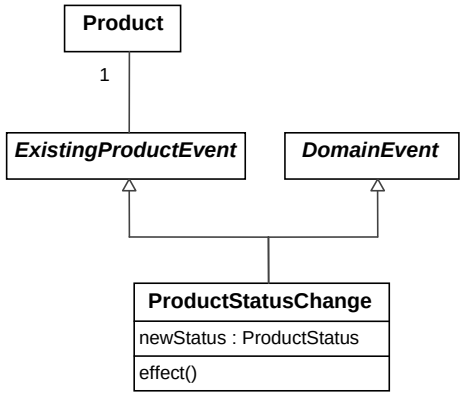


Effect

context ProductOptionValueAttributeChange::effect()
post : productAttribute.attribute.value = self.value

Event
ProductStatusChange

Event diagram

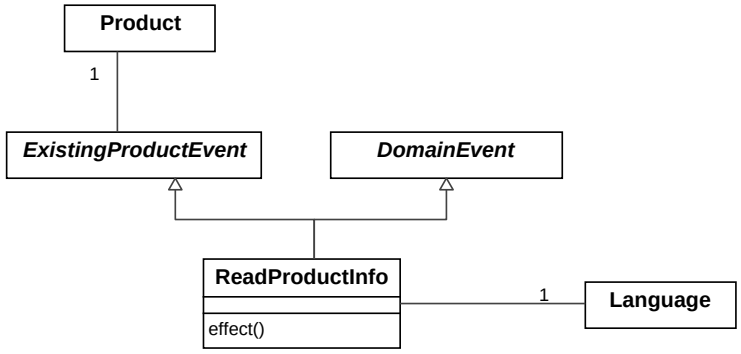


Effect

context ProductStatusChange::effect()
post : self.product.status = self.newStatus

Event
ReadProductInfo

Event diagram

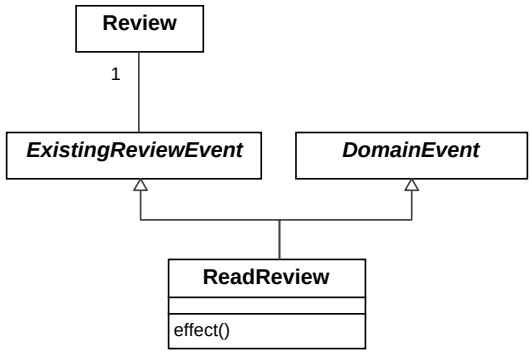


Effect

context ReadProductInfo::effect()
post : self.product.productInLanguage->select(pil | pil.language=self.language).viewed =
self.product@pre.productInLanguage@pre->select(pil | pil.language=self.language).viewed + 1

Event
ReadReview

Event diagram

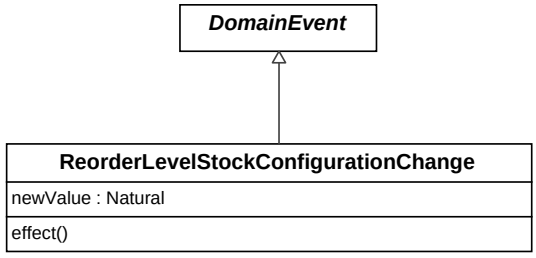


Effect

context ReadReview::effect()
post : self.review.timesRead = self.review@pre.timesRead + 1

Event
ReorderLevelStockConfigurationChange

Event diagram

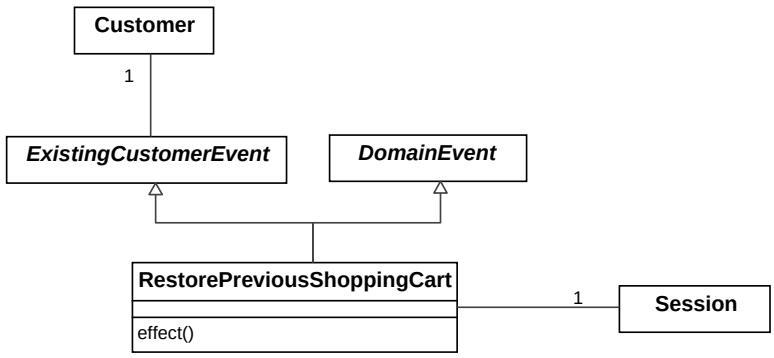


Effect

context ReorderLevelStockConfigurationChange::effect()
post : Stock.stockReorderLevel = self.newValue

Event
RestorePreviousShoppingCart

Event diagram



Initial Integrity Constraints

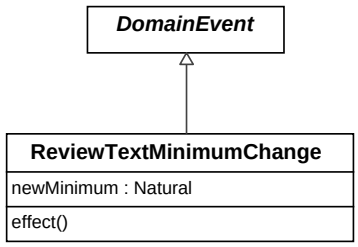
context RestorePreviousShoppingCart::CustomerHasAPreviousShoppingCart(): Boolean
body : self.customer.customerShoppingCart->notEmpty()

Effect

context RestorePreviousShoppingCart::effect()
post : self.session.shoppingCart = self.customer.customerShoppingCart

Event
ReviewTextMinimumChange

Event diagram

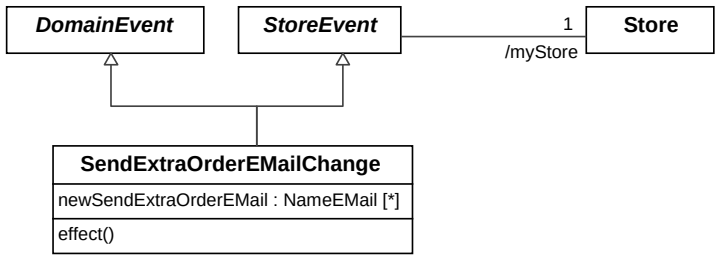


Effect

context ReviewTextMinimumChange::effect()
post : MinimumValues.reviewText = self.newMinimum

Event
SendExtraOrderEmailChange

Event diagram

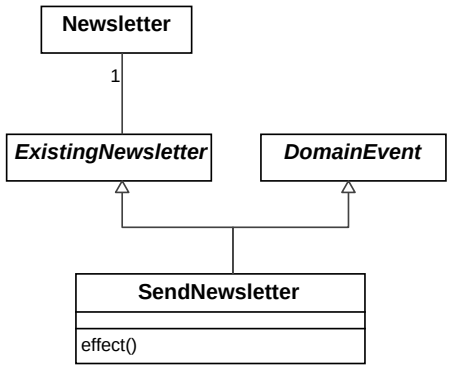


Effect

context SendExtraOrderEmailChange::effect()
post : myStore. sendExtraOrderEmail = self.newSendExtraOrderEmail

Event
SendNewsletter

Event diagram

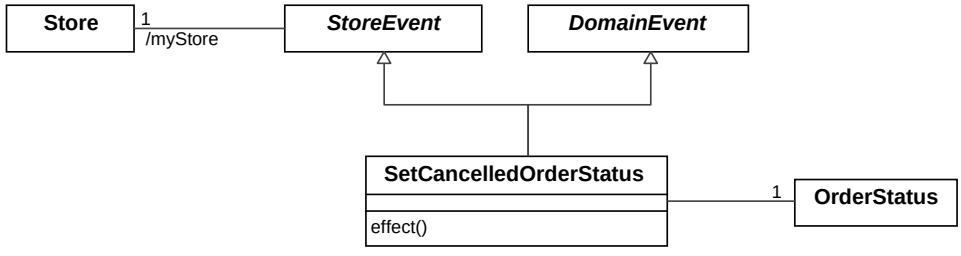


Effect

context SendNewsletter::effect()
post : self.newsletter.sent = Now()

Event
SetCancelledOrderStatus

Event diagram



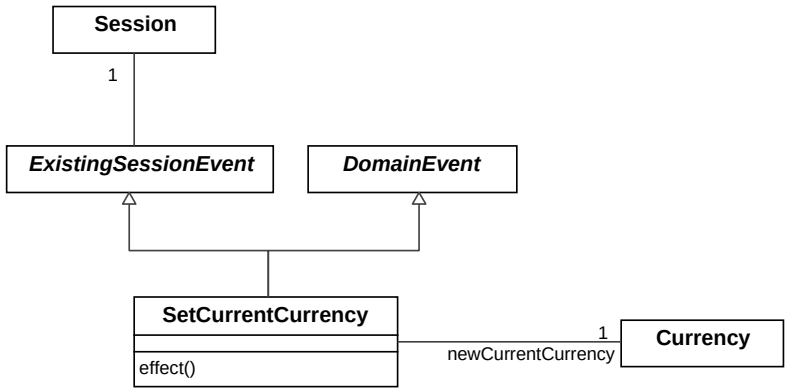
Effect

context SetCancelledOrderStatus::effect()
post : self.myStore.cancelledStatus = self.orderStatus

Event

SetCurrentCurrency

Event diagram



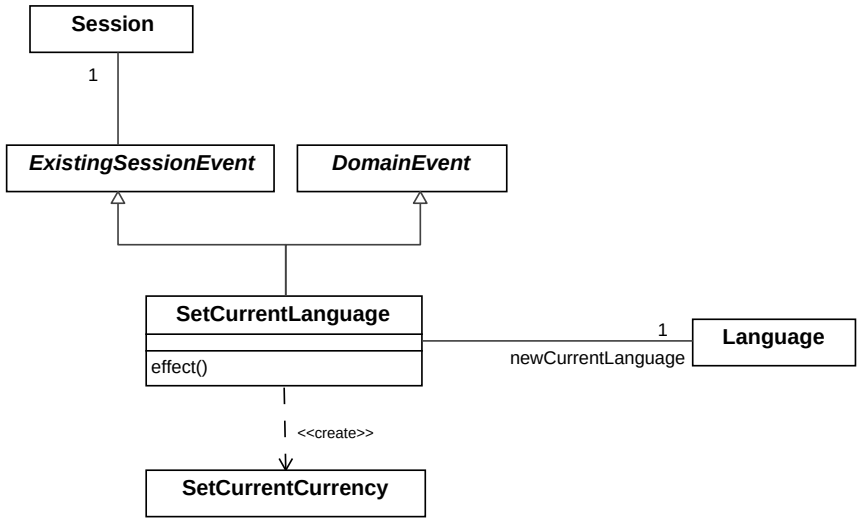
Effect

context SetCurrentCurrency::effect()
post : self.session.currentCurrency = self.newCurrentCurrency

Event

SetCurrentLanguage

Event diagram



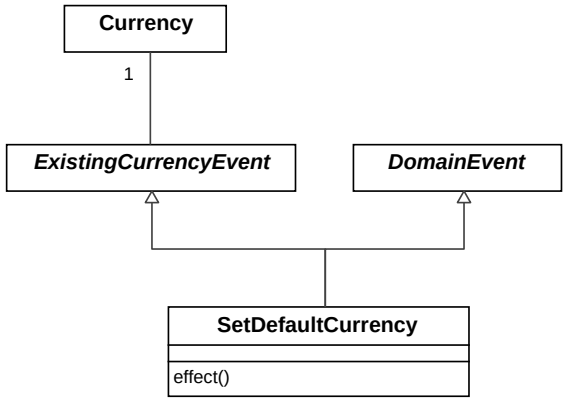
Effect

```
context ChangeCurrentLanguage::effect()
  post :
    session.currentLanguage = self.newCurrentLanguage
  post :
    Store.allInstances() -> any(true).switchToDefaultLanguageCurrency and
    self.language.defaultCurrency -> notEmpty()
    implies
    ccc.oclIsNew() and
    ccc.oclIsTypeOf(ChangeCurrentCurrency) and
    ccc.session = self.session and
    ccc.newCurrentCurrency = self.language.defaultCurrency
```

Event

SetDefaultCurrency

Event diagram

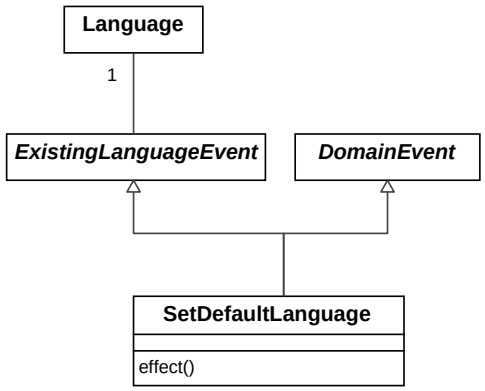


Effect

```
context SetDefaultCurrency::effect()
  post : Store.allInstances() -> any(true).defaultCurrency = self.currency
```

Event
SetDefaultLanguage

Event diagram

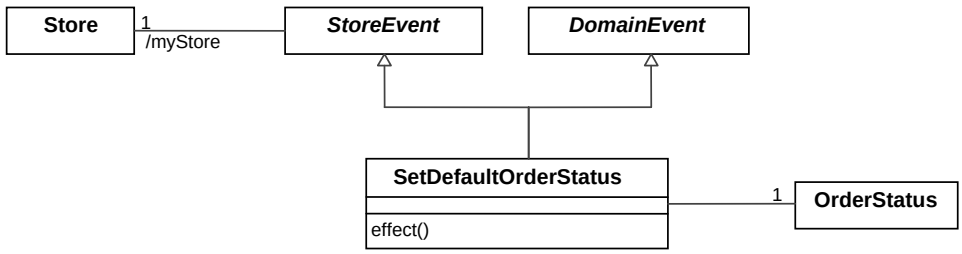


Effect

context SetDefaultLanguage::effect()
post : Store.allInstances() -> any(true).defaultLanguage = self.language

Event
SetDefaultOrderStatus

Event diagram

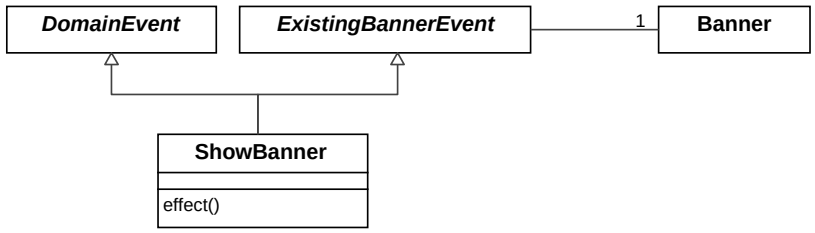


Effect

context SetPendingOrderStatus::effect()
post : self.myStore.defaultStatus = self.orderStatus

Event
ShowBanner

Event diagram

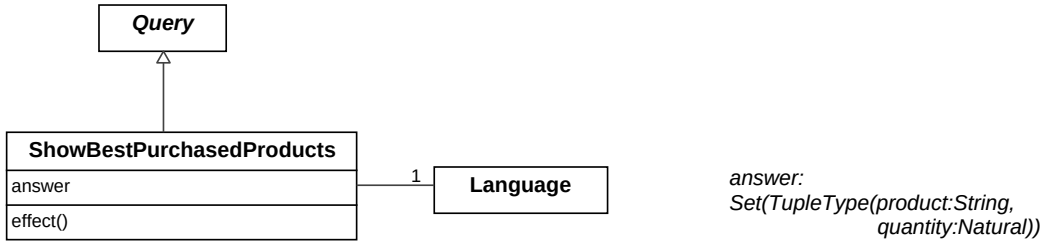


Effect

```
context ShowBanner::effect()
  post :
     BannerHistory.allInstances() -> one
       (bh | bh.banner = self.banner and
             bh.date = today() and
             bh.shown = bh@pre.shown + 1)
```

Event
ShowBestPurchasedProducts

Event diagram



Effect

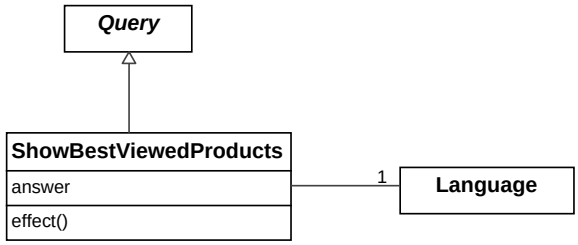
```
context ShowBestPurchasedProducts::effect()
  post :
     answer =
     Product.allInstances()
```

```
-> sortedBy(quantityOrdered)
-> collect (p | Tuple  {product = ProductInLanguage.allInstances() ->select
                                    (pil | pil.product = p  and
                                           pil.language=language).name,
                        quantity = p.quantityOrdered})->asSet()
```

Event

ShowBestViewedProducts

Event diagram



answer:
Set(TupleType(product:String,
quantity:Natural))

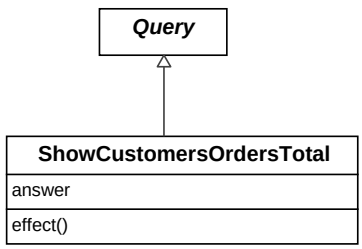
Effect

```
context  ShowBestViewedProducts::effect()
  post :
     answer =
     Product.allInstances()
     -> sortedBy(timesViewed)
     -> collect (p | Tuple  {product = ProductInLanguage.allInstances() ->select
                                    (pil | pil.product = p  and
                                           pil.language=language).name,
                             timesViewed = p.timesViewed})->asSet
```

Event

ShowCustomersOrdersTotal

Event diagram



answer:
Set(TupleType(name:String,
total:Money))

Effect

```
context ShowCustomersOrdersTotal::effect()
  post :
     answer =
     Customer.allInstances()
     -> collect (c | Tuple  {name = c.firstName.concat(c.lastName),
                             total = c.order.total -> sum()}) -> asSet()
```

Event

ShowExpectedProducts

Event diagram

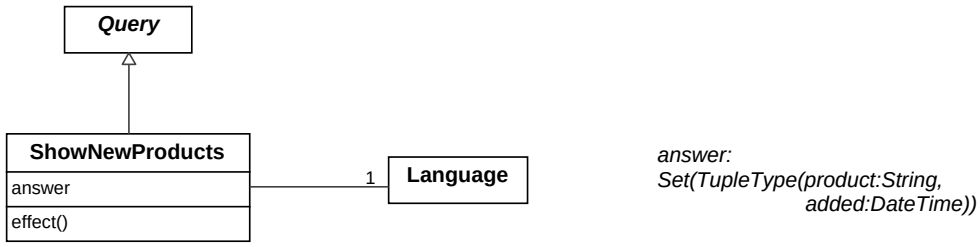
Effect

```
context ShowExpectedProducts::effect()
  post :
     answer =
     Product.allInstances() -> select(p|p.available.notEmpty() and p.available > Now())
     -> sortedBy(available)
     -> collect (p | Tuple  {product = ProductInLanguage.allInstances() ->select
                                       (pil | pil.product = p  and
                                              pil.language=language).name,
                             dateAvailable = p.available}) ->asSet()
```

Event

ShowNewProducts

Event diagram



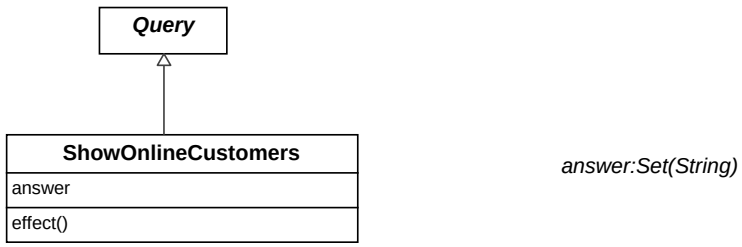
Effect

```
context  ShowNewProducts::effect()
  post :
     answer =
     Product.allInstances()
     -> SortedBy(added)
     -> collect (p | Tuple  {product = ProductInLanguage.allInstances() ->select
                                          (pil | pil.product = p  and
                                                 pil.language=language).name,
                                added = p.added})->asSet()
```

Event

ShowOnlineCustomers

Event diagram



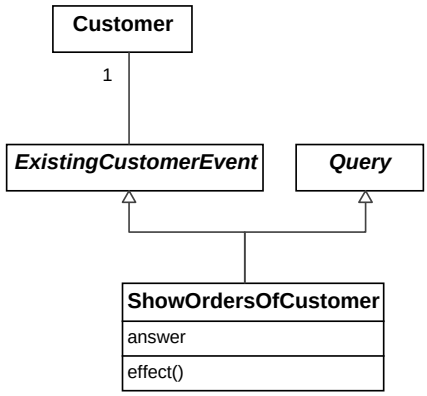
Effect

```
context ShowOnlineCustomers::effect()
  post :
     answer =
     Session.allInstances().customer
     -> collect (c | c.firstName.concat(c.lastName))->asSet()
```

Event

ShowOrdersOfCustomer

Event diagram



answer:
Set(TupleType(id:Natural,
total:Money,
status:OrderStatus))

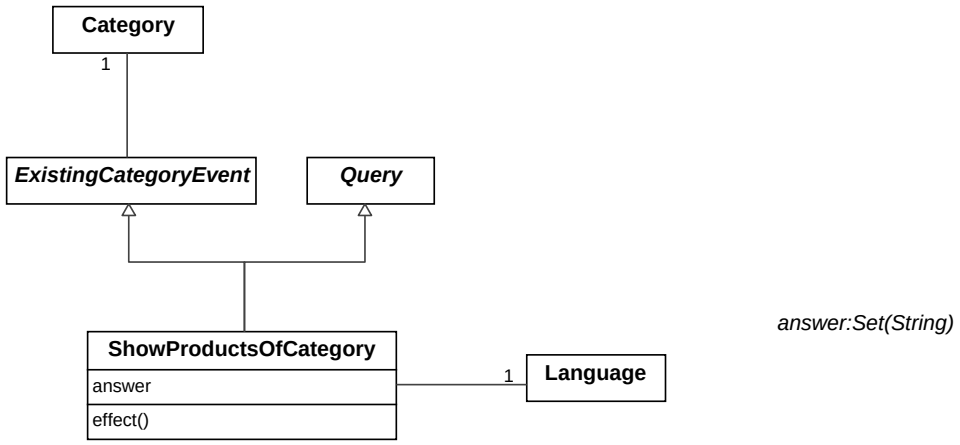
Effect

```
context ShowOrdersOfCustomer::effect()
  post :
     answer =
     self.customer.order
     -> collect (o | Tuple  {id = o.id,
                             total = o.total
                             status = o.orderStatusChange -> sortedBy(added) -> last()})->asSet()
```

Event

ShowProductsOfCategory

Event diagram



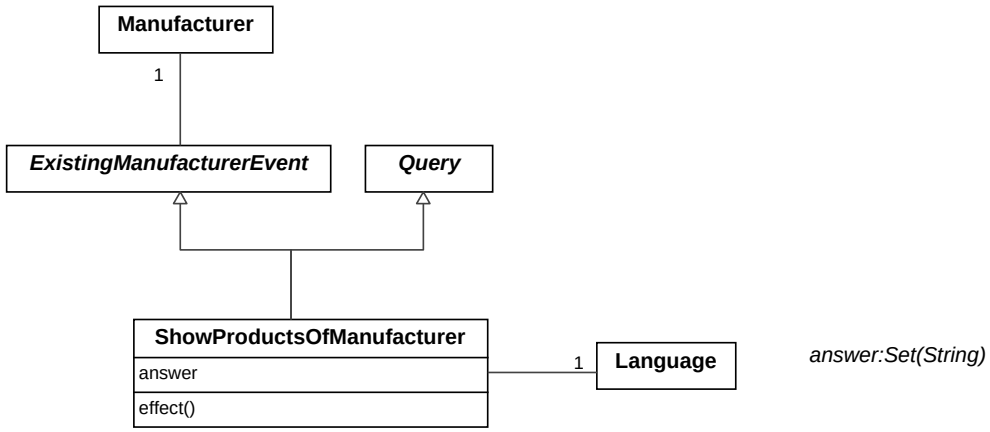
Effect

```
context  ShowProductsOfCategory::effect()
  post :
     answer =
     Product.allInstances() -> select(p | p.category -> includes(self.category))
     -> collect (p | ProductInLanguage.allInstances() ->select
                                        (pil | pil.product = p  and
                                               pil.language=language).name)->asSet()
```

Event

ShowProductsOfManufacturer

Event diagram

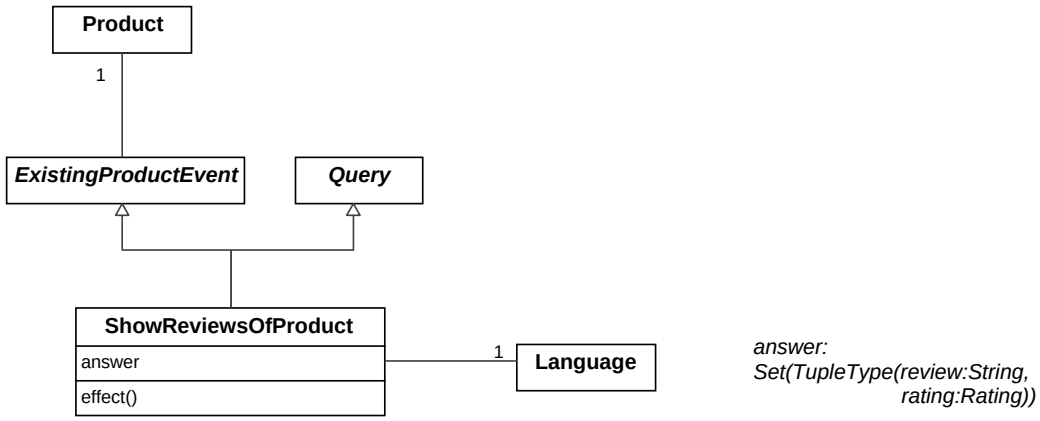


Effect

```
context  ShowProductsOfManufacturer::effect()
  post :
     answer =
     Product.allInstances() -> select(p | p.manufacturer=self.manufacturer)
     -> collect (p | ProductInLanguage.allInstances() ->select
                                        (pil | pil.product = p  and
                                               pil.language=language).name)->asSet()
```

Event
ShowReviewsOfProduct

Event diagram



Effect

```
context  ShowReviewsOfProduct::effect()
  post :
     answer =
     self.product.review -> select (r | r.language = self.language)
     -> collect (r | Tuple  {review = r.review
                             rating= r.rating})->asSet()
```

Event
ShowSpecials

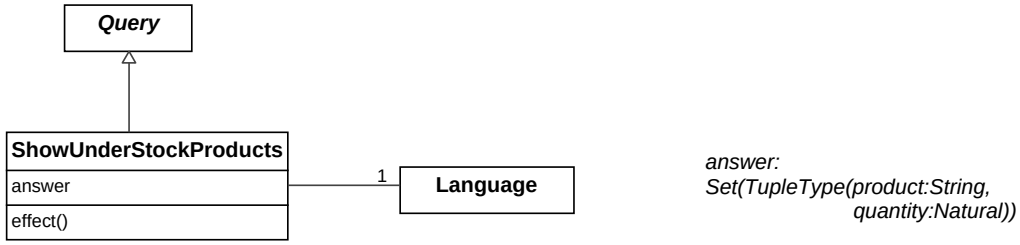
Event diagram

Effect

```
context ShowSpecials::effect()
  post :
     answer =
     Special.allInstances()
     -> collect (s | Tuple  {product = ProductInLanguage.allInstances() ->select
                                        (pil | pil.product = s  and
                                               pil.language=language).name,
                             oldPrice = s.netPrice
                             newPrice = s.specialPrice})->asSet()
```

Event
ShowUnderStockProducts

Event diagram

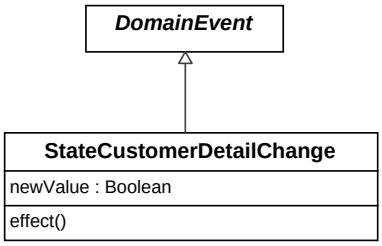


Effect

```
context ShowUnderStockProducts::effect()
  post :
     answer =
     Product.allInstances() -> select(p | p.quantityOnHand < Stock.stockReorderLevel)
     -> SortedBy(Store.allInstances()->any(true).sortedBy(expectedSortField)
     -> collect (p | Tuple  {product = ProductInLanguage.allInstances() ->select
                                        (pil | pil.product = p  and
                                               pil.language=language).name,
                             quantity = p.quantityOnHand}) -> asSet()
```

Event
StateCustomerDetailChange

Event diagram

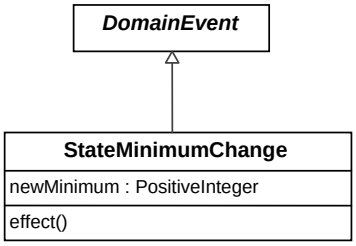


Effect

context StateCustomerDetailChange::effect()
post : CustomerDetails.state = self.newValue

Event
StateMinimumChange

Event diagram



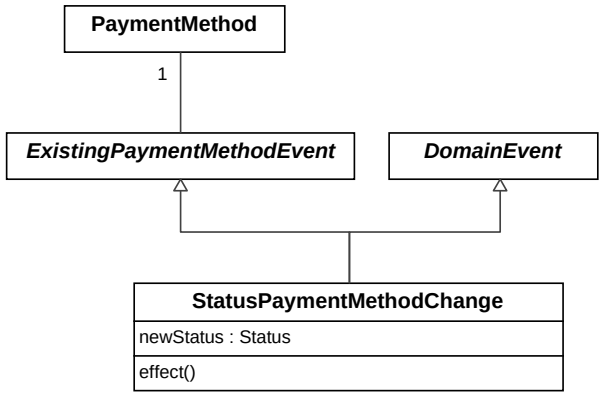
Effect

context StateMinimumChange::effect()
post : MinimumValues.state = self.newMinimum

Event

StatusPaymentMethodChange

Event diagram



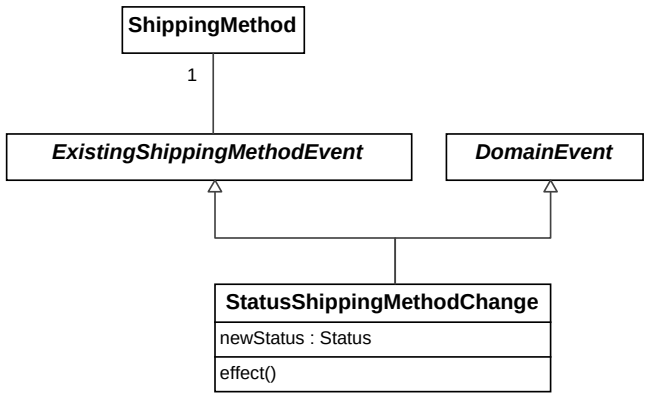
Effect

context StatusPaymentMethodChange::effect()
post : self.paymentMethod.status = self.newStatus

Event

StatusShippingMethodChange

Event diagram

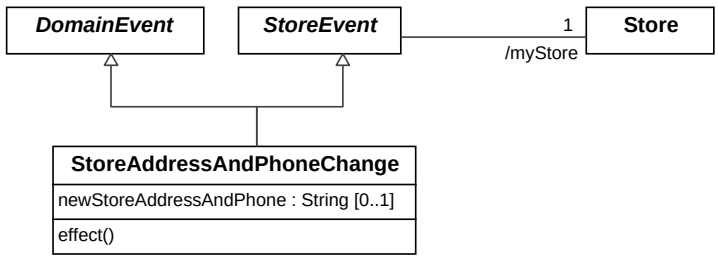


Effect

context StatusShippingMethodChange::effect()
post : self.shippingMethod.status = self.newStatus

Event
StoreAddressAndPhoneChange

Event diagram

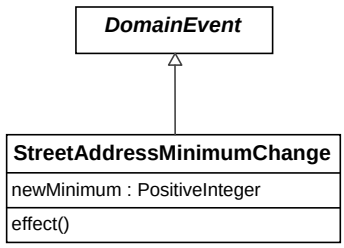


Effect

context StoreAddressAndPhoneChange::effect()
post : myStore.storeAddressAndPhone = self.newStoreAddressAndPhone

Event
StreetAddressMinimumChange

Event diagram

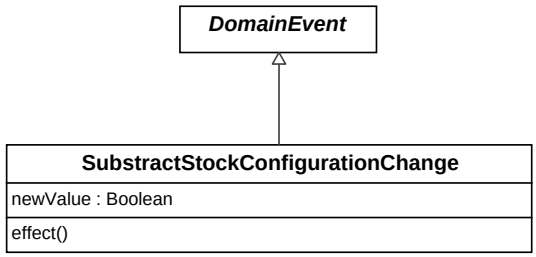


Effect

context StreetAddressMinimumChange::effect()
post : MinimumValues.streetAddress = self.newMinimum

Event
SubstractStockConfigurationChange

Event diagram

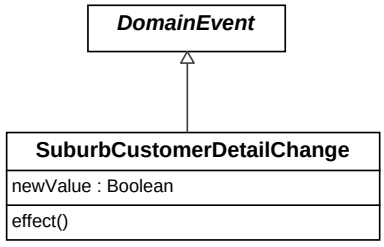


Effect

context SubstractStockConfigurationChange::effect()
post : Stock.substrackStock= self.newValue

Event
SuburbCustomerDetailChange

Event diagram



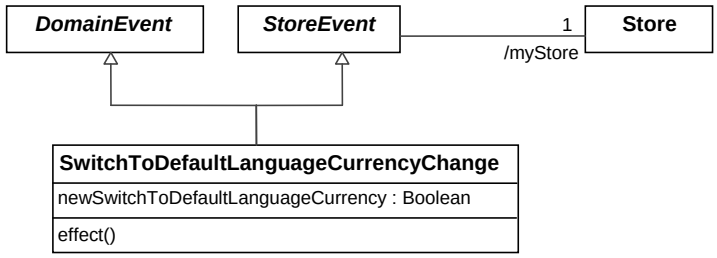
Effect

context SuburbCustomerDetailChange::effect()
post : CustomerDetails.suburb = self.newValue

Event

SwitchToDefaultLanguageCurrencyChange

Event diagram



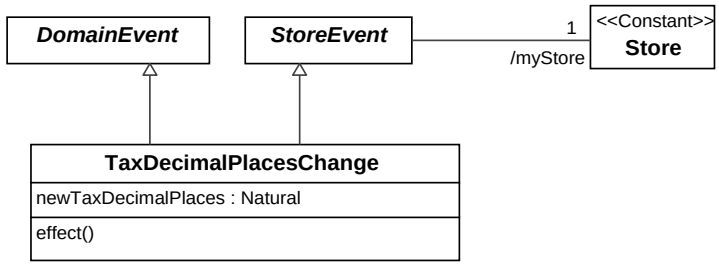
Effect

context SwitchToDefaultLanguageCurrencyChange::effect()
post : myStore.switchToDefaultLanguageCurrency = self.newSwitchToDefaultLanguageCurrency

Event

TaxDecimalPlacesChange

Event diagram



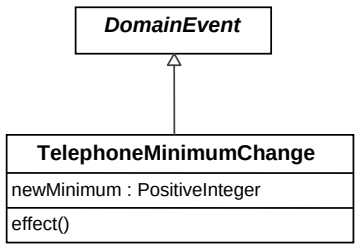
Effect

context TaxDecimalPlacesChange::effect()
post : myStore.taxDecimalPlaces = self.newTaxDecimalPlaces

Event

TelephoneMinimumChange

Event diagram



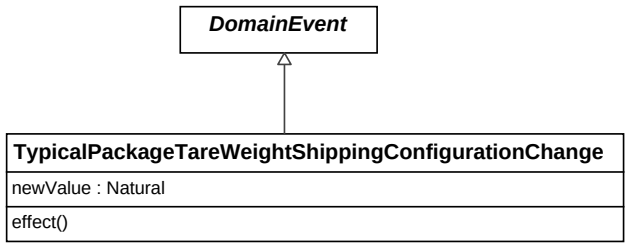
Effect

context TelephoneMinimumChange::effect()
post : MinimumValues.telephoneNumber = self.newMinimum

Event

TypicalPackageTareWeightShippingConfigurationChange

Event diagram



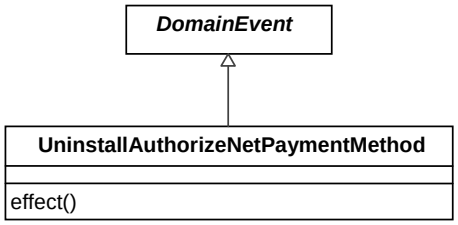
Effect

context TypicalPackageTareWeightShippingConfigurationChange::effect()
post : ShippingAndPackaging.typicalPackageTareWeight = self.newValue

Event

UnInstallAuthorizeNetPaymentMethod

Event diagram



Initial Integrity Constraints

context UninstallAuthorizeNetPaymentMethod::PaymentMethodIsInstalled():Boolean
body : AuthorizeNet.allInstances() -> notEmpty()

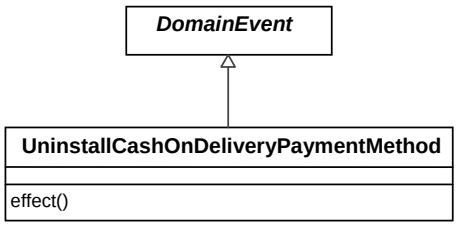
Effect

context UninstallAuthorizeNetPaymentMethod::effect()
post : AuthorizeNet.allInstances() -> any(true)@pre.oclIsKindOf(OclAny)

Event

UninstallCashOnDeliveryPaymentMethod

Event diagram



Initial Integrity Constraints

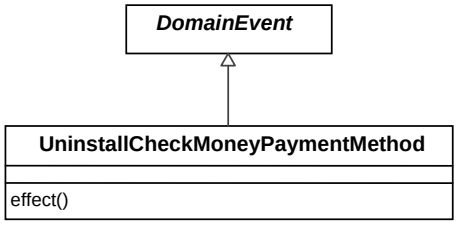
context UninstallCashOnDeliveryPaymentMethod::PaymentMethodIsInstalled():Boolean
body : CashOnDelivery.allInstances() -> notEmpty()

Effect

context UninstallCashOnDeliveryPaymentMethod::effect()
post : CashOnDelivery.allInstances() -> any(true)@pre.oclIsKindOf(OclAny)

Event
UninstallCheckMoneyPaymentMethod

Event diagram



Initial Integrity Constraints

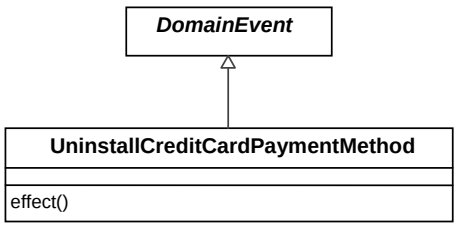
context UninstallCheckMoneyPaymentMethod::PaymentMethodIsInstalled():Boolean
body : CheckMoney.allInstances() -> notEmpty()

Effect

context UninstallCheckMoneyPaymentMethod::effect()
post : CheckMoney.allInstances() -> any(true)@pre.oclIsKindOf(OclAny)

Event
UninstallCreditCartPaymentMethod

Event diagram



Initial Integrity Constraints

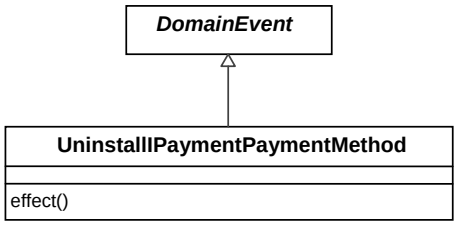
context UninstallCreditCardPaymentMethod::PaymentMethodIsInstalled():Boolean
body : CreditCard.allInstances() -> notEmpty()

Effect

context UninstallCreditCardPaymentMethod::effect()
post : CreditCard.allInstances() -> any(true)@pre.oclIsKindOf(OclAny)

Event
UninstallIPaymentPaymentMethod

Event diagram



Initial Integrity Constraints

context UninstallIPaymentPaymentMethod::PaymentMethodIsInstalled():Boolean
body : IPayment.allInstances() -> notEmpty()

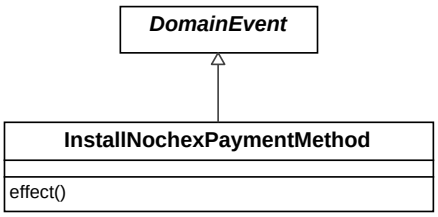
Effect

context UninstallIPaymentPaymentMethod::effect()
post : IPayment.allInstances() -> any(true)@pre.oclIsKindOf(OclAny)

Event

UninstallNochexPaymentMethod

Event diagram



Initial Integrity Constraints

context UninstallNochexPaymentMethod::PaymentMethodIsInstalled():Boolean
body : Nochex.allInstances() -> notEmpty()

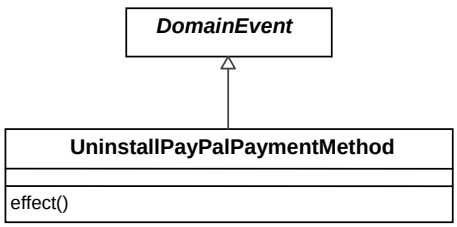
Effect

context UninstallNochexPaymentMethod::effect()
post : Nochex.allInstances() -> any(true)@pre.oclIsKindOf(OclAny)

Event

UninstallPayPalPaymentMethod

Event diagram



Initial Integrity Constraints

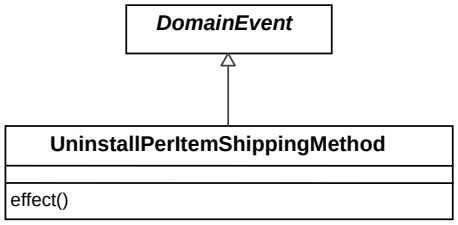
context UninstallPayPalPaymentMethod::PaymentMethodIsInstalled():Boolean
body : PayPal.allInstances() -> isEmpty()

Effect

context UninstallPayPalPaymentMethod::effect()
post : PayPal.allInstances() -> any(true)@pre.oclIsKindOf(OclAny)

Event
UninstallPerItemPaymentMethod

Event diagram



Initial Integrity Constraints

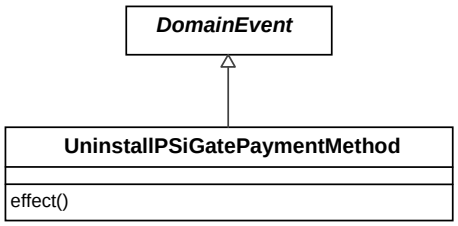
context UninstallPerItemShippingMethod::ShippingMethodIsInstalled():Boolean
body : PerItem.allInstances() -> notEmpty()

Effect

context UninstallPerItemShippingMethod::effect()
post : PerItem.allInstances() -> any(true)@pre.oclIsKindOf(OclAny)

Event
UninstallPSiGatePaymentMethod

Event diagram



Initial Integrity Constraints

context UninstallPSiGatePaymentMethod::PaymentMethodIsInstalled():Boolean
body : PSiGate.allInstances() -> notEmpty()

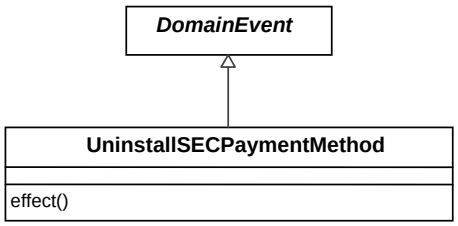
Effect

context UninstallPSiGatePaymentMethod::effect()
post : PSiGate.allInstances() -> any(true)@pre.oclIsKindOf(OclAny)

Event

UninstallSECPaymentMethod

Event diagram



Initial Integrity Constraints

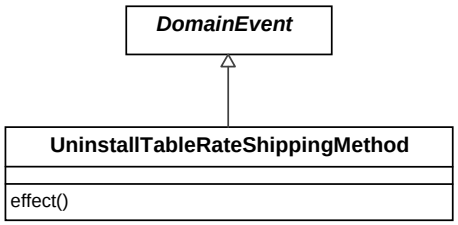
context UninstallSECPaymentMethod::PaymentMethodIsInstalled():Boolean
body : SECPay.allInstances() -> notEmpty()

Effect

context UninstallSECPaymentMethod::effect()
post : SECPay.allInstances() -> any(true)@pre.oclIsKindOf(OclAny)

Event
UninstallTableRatePaymentMethod

Event diagram



Initial Integrity Constraints

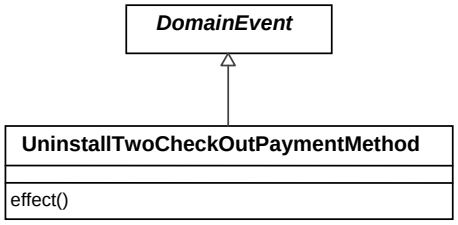
context UninstallTableRateShippingMethod::ShippingMethodIsInstalled():Boolean
body : TableRate.allInstances() -> notEmpty()

Effect

context UninstallTableRateShippingMethod::effect()
post : TableRate.allInstances() -> any(true)@pre.oclIsKindOf(OclAny)

Event
UninstallTwoCheckOutPaymentMethod

Event diagram



Initial Integrity Constraints

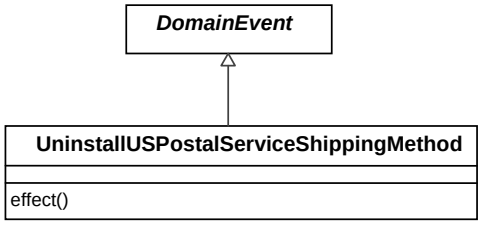
context UninstallTwoCheckOutPaymentMethod::PaymentMethodIsInstalled():Boolean
body : TwoCheckOut.allInstances() -> notEmpty()

Effect

context UninstallTwoCheckOutPaymentMethod::effect()
post : TwoCheckOut.allInstances() -> any(true)@pre.oclIsKindOf(OclAny)

Event
UninstallUSPostalServicePaymentMethod

Event diagram



Initial Integrity Constraints

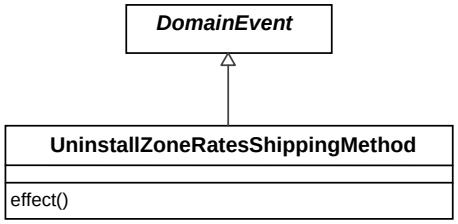
context UninstallUSPostalServiceShippingMethod::ShippingMethodIsInstalled():Boolean
body : USPostalService.allInstances() -> notEmpty()

Effect

context UninstallUSPostalServiceShippingMethod::effect()
post : USPostalService.allInstances() -> any(true)@pre.oclIsKindOf(OclAny)

Event
UninstallZoneRatesShippingMethod

Event diagram



Initial Integrity Constraints

context UninstallZoneRatesShippingMethod::ShippingMethodIsInstalled():Boolean
body : ZoneRates.allInstances() -> notEmpty()

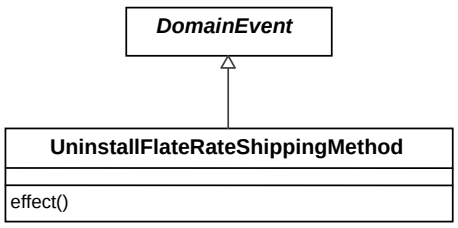
Effect

context UninstallZoneRatesShippingMethod::effect()
post : ZoneRates.allInstances() -> any(true)@pre.oclIsKindOf(OclAny)

Event

UninstallFlatRateShippingMethod

Event diagram



Initial Integrity Constraints

context UninstallFlatRateShippingMethod::ShippingMethodIsInstalled():Boolean
body : FlatRate.allInstances() -> notEmpty()

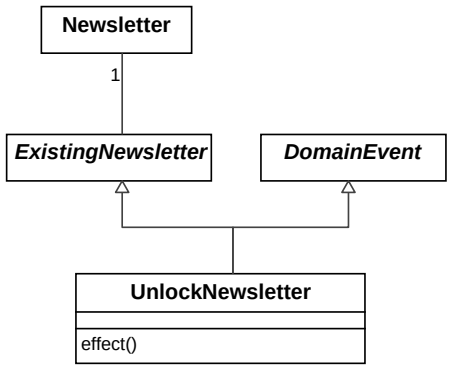
Effect

context UninstallFlatRateShippingMethod::effect()
post : FlatRate.allInstances() -> any(true)@pre.oclIsKindOf(OclAny)

Event

UnlockNewsletter

- **Event diagram**



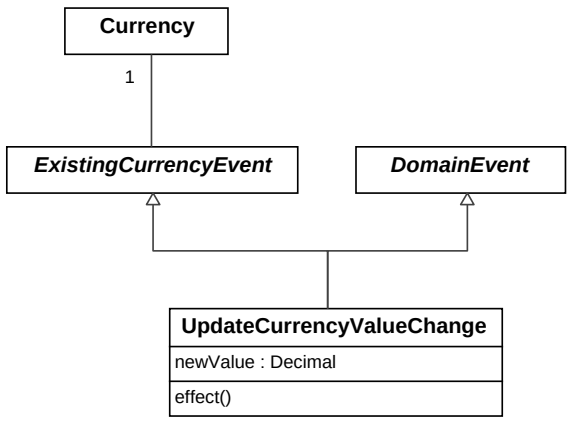
- **Effect**

context UnlockNewsletter::effect()
post : self.newsletter.status = NewsletterStatus::unlocked

Event

UpdateCurrencyValueChange

- **Event diagram**



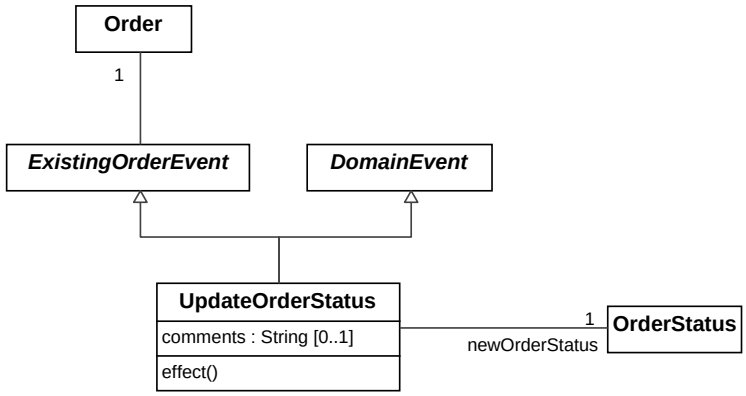
Effect

```
context UpdateCurrencyValueChange::effect()
  post : self.currency.value = self.newValue
  post : self.currency.lastUpdated = Now()
```

Event

UpdateOrderStatus

Event diagram



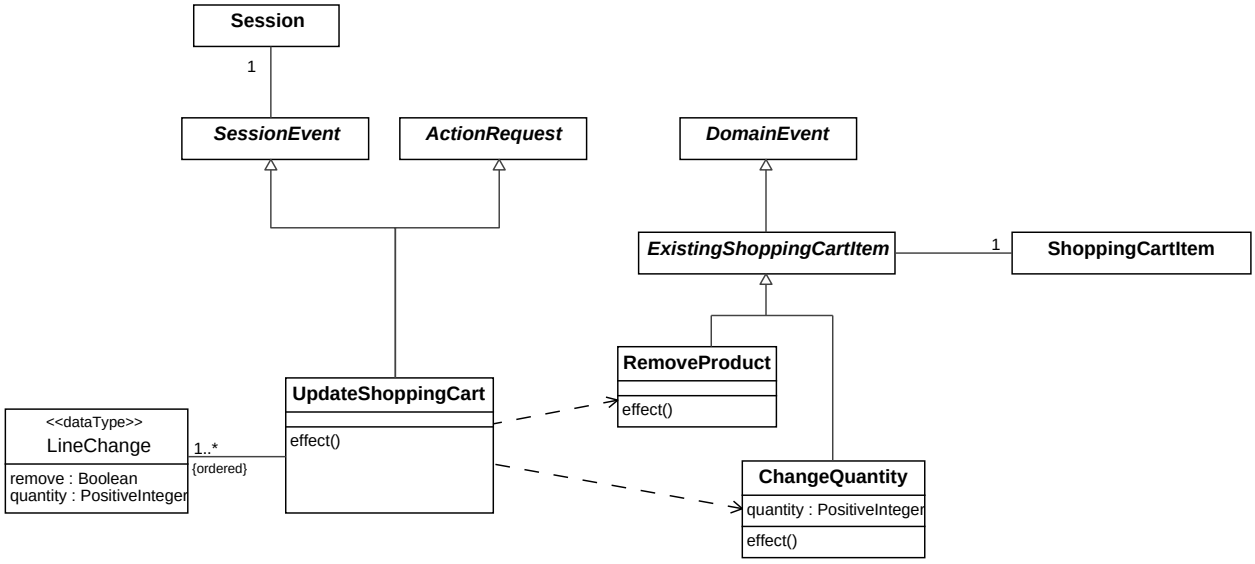
Effect

```
context ChangeOrderStatus::effect()
  post :
     osc.oclIsNew() and
     osc.oclIsTypeOf(OrderStatusChange) and
     osc.comments = self.comments and
     osc.order = self.order and
     osc.orderStatus = self.newOrderStatus
```

Event

UpdateShoppingCart

Event diagram



Initial Integrity Constraints

context UpdateShoppingCart::complete(): Boolean
body : self.lineChange->size() = self.session.shoppingCart.shoppingCartItem->size()

Effect

context RemoveProduct::effect()
post : **not** self.shoppingCartItem@pre.oclIsKindOf(OclAny)

context ChangeQuantity::effect()
post : self.shoppingCartItem.quantity = self.quantity

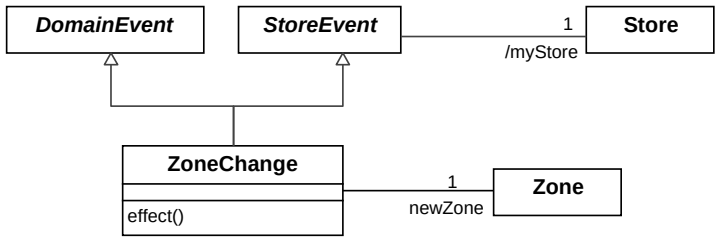
context UpdateShoppingCart::effect()
post :
self.lineChange ->forAll
(lc | **let** cartItem:ShoppingCartItem =
self.shoppingCart.shoppingCartItem->
at(lineChange->indexOf(lc))
in
(lc.remove or lc.quantity <> cartItem.quantity)
implies

```
if lc.remove then
   rp.oclIsNew and
   rp.oclIsTypeOf(RemoveProduct) and
   rp.shoppingCartItem = cartItem
else
   cq.oclIsNew() and
   cq.oclIsTypeOf(ChangeQuantity) and
   cq.shoppingCartItem = cartItem and
   cq.quantity = quantity
endif )
```

Event

ZoneChange

Event diagram



Effect

```
context  ZoneChange::effect()
 post :  myStore.zone = self.newZone
```