8 STRUCTURAL SCHEMA

8.1 INTRODUCTION

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

The main purpose of the osCommerce structural schema is to provide a description of the conceptualization of the osCommerce domain.

The structural schema is too large to be presented together. Therefore, this chapter begins with a UML general view diagram of the most important conceptual entity types and their relationship types.

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

Entities which appear in a structural schema fragment but are fully specified in other conceptual grouping diagrams are drawn without showing their attributes. These "external" entities are referenced to the page where their complete specification can be found.

Each fragment of the whole structural schema diagram is represented in UML and 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, etc.) explained in Conceptual Modeling of Information Systems [Oli07] written by Antoni Olivé.

In addition, and in order to improve comprehension, a detailed description of the schema fragment and example instantiations of it are provided. 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 taken from real life experiences.
8.2 OVERVIEW DIAGRAM

The overview 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.
8.3 MAIN DOMAIN CONCEPTS

The products in the store are produced 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 can work 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 increases or decreases depending on the chosen attributes. The price variation produced by an attribute is indicated, for each product, by the product attribute entity type.

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. Every session always has a current currency and a current language.

In the context of sessions, users can navigate through 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 customers confirm that they want to buy the contents of their 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.
8.4 STORE CONFIGURATION

Store Data

- Overview

*osCommerce* saves general data about the store and some other information which is used to customize the behavior 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 saves 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:

```plaintext
gaudisc : Store

name = "Gaudisc"
eMailAddress = "info@gaudisc.com"
eMailFrom = "gaudisc@gaudisc.com"
expectedSortField = productName
expectedSortOrder = ascending
sendExtraOrderFMail = "orders@gaudisc.com",
backup@gaudisc.com"
displayCartAfterAddingProduct = true
allowGuestToTellAFriend = true
defaultSearchOperator = or
storeAddressAndPhone = "C/Historiador Mayans, 7
08028 Barcelona
Tel 934 355 441"
taxDecimalPlaces = "2"
displayPricesWithTax = false
switchToDefaultLanguageCurrency = false
```

```plaintext
spanish : Language
default_language

guro : Currency
default_currency

cancelado : OrderStatus
cancelledStatus

pendiente : OrderStatus
defaultStatus
```

```plaintext
spain : Country

barcelona : Zone
```
Minimum and maximum values

- Overview

-osCommerce allows the minimum and maximum length for some String attributes to be defined.

- Structural Schema

```
<<utility>>
MinimumValues

<table>
<thead>
<tr>
<th>attribute</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstName</td>
<td>PositiveInteger</td>
</tr>
<tr>
<td>lastName</td>
<td>PositiveInteger</td>
</tr>
<tr>
<td>dateOfBirth</td>
<td>PositiveInteger</td>
</tr>
<tr>
<td>emailAddress</td>
<td>PositiveInteger</td>
</tr>
<tr>
<td>streetAddress</td>
<td>PositiveInteger</td>
</tr>
<tr>
<td>companyName</td>
<td>Natural</td>
</tr>
<tr>
<td>postcode</td>
<td>PositiveInteger</td>
</tr>
<tr>
<td>city</td>
<td>PositiveInteger</td>
</tr>
<tr>
<td>state</td>
<td>PositiveInteger</td>
</tr>
<tr>
<td>telephoneNumber</td>
<td>PositiveInteger</td>
</tr>
<tr>
<td>password</td>
<td>PositiveInteger</td>
</tr>
<tr>
<td>creditCardOwnerName</td>
<td>PositiveInteger</td>
</tr>
<tr>
<td>creditCardNumber</td>
<td>PositiveInteger</td>
</tr>
<tr>
<td>reviewText</td>
<td>Natural</td>
</tr>
</tbody>
</table>
```

```
<<utility>>
MaximumValues

<table>
<thead>
<tr>
<th>attribute</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>addressBookEntries</td>
<td>Natural</td>
</tr>
</tbody>
</table>
```

- 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

Minimum Values also specifies the minimum length of:

- Product review texts

Finally, Maximum Values specifies the maximum number of:

- Address book entries permitted for each customer.
Customer details configuration

- Overview

The system allows the store administrator to specify whether some customer attributes are shown and required when creating, editing or showing an account.

- Structural Schema

```
<<utility>>
CustomerDetails

<table>
<thead>
<tr>
<th>gender</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>dateOfBirth</td>
<td>Boolean</td>
</tr>
<tr>
<td>company</td>
<td>Boolean</td>
</tr>
<tr>
<td>suburb</td>
<td>Boolean</td>
</tr>
<tr>
<td>state</td>
<td>Boolean</td>
</tr>
</tbody>
</table>
```

- Description

`CustomersDetails` configures 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 the store administrator to set up some configuration values used in shipping costs calculation.

Structural Schema

```
<<utility>>
ShippingAndPackaging

postCode : PostalCode [0..1]
maximumPackageWeight : Decimal
typicalPackageTareWeight : Decimal
percentageIncreaseForLargerPackages : Decimal

Context: Location & Taxes 61
countryOfOrigin

<<dataType>>
PostalCode

postalCode : String
```

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 by 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 the store administrator to customize the most important general downloadable product properties.

■ Structural Schema

```
<<utility>>
Download
enableDownload : Boolean
daysExpiryDelay : Natural
maximumNumberOfDownloads : Natural
```

■ Description

There is a special type of product *Option* which allows customers to download the product.

The general properties of downloadable products can be customized by 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 they can be redefined later.
Stock configuration

- Overview

The system allows the store administrator to configure some options over the administration of stock.

- Structural Schema

```
<<utilly>>
Stock
checkStockLevel : Boolean
subtractStock : Boolean
allowCheckout : Boolean
stockReOrderLevel : Natural
```

- Description

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

It is also 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 to check 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 can operate 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 to pay through different payment methods, which can be enabled or disabled by the store administrator.

Some of the payment methods, like Authorize.net, iPaiment, 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.

All the payment methods have specific information about all the data needed 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 can operate 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

The osCommerce store administrator can customize the shipping methods that 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 weigh or other criteria:

- **Flat rate**: A single price is used on all orders, regardless of how many items are bought, how much everything weighs, 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 the **Zone Rates** method represent 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

```
<table>
<thead>
<tr>
<th>Language</th>
<th>.</th>
<th>0..1</th>
<th>Currencies 50</th>
</tr>
</thead>
<tbody>
<tr>
<td>name : String</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>code : String</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>image : File [0..1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>directory : String</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sortOrder : Natural</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>defaultCurrency</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- 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 the corresponding 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.

Languages can also have a **default currency**.

If the **Store** attribute **SwitchToDefaultLanguageCurrency** is activated, when a language becomes the current language, its default currency becomes the current currency.
Examples

1: Language
   name = "English"
   code = "EN"

dollar: Currency

defaultCurrency

2: Language
   name = "Deutsch"
   code = "DE"

euro: Currency

defaultCurrency

3: Language
   name = "Español"
   code = "ES"

euro: Currency

defaultCurrency

4: Language
   name = "Catalá"
   code = "CAT"
Currencies

- Overview

osCommerce is able to work with different currencies.

- Structural Schema

<table>
<thead>
<tr>
<th>Currency</th>
</tr>
</thead>
<tbody>
<tr>
<td>title : String</td>
</tr>
<tr>
<td>code : String</td>
</tr>
<tr>
<td>symbolLeft : String [0..1]</td>
</tr>
<tr>
<td>symbolRight : String [0..1]</td>
</tr>
<tr>
<td>decimalPlaces : Natural</td>
</tr>
<tr>
<td>value : Decimal</td>
</tr>
<tr>
<td>lastUpdate : DateTime [0..1]</td>
</tr>
<tr>
<td>status : Status</td>
</tr>
</tbody>
</table>

<<enumeration>>
- Status
  - enabled
  - disabled

- Constraints

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

class 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 allow 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, deals with two different currencies: Euros and US Dollars. The instantiation of these currencies is:

```plaintext
<table>
<thead>
<tr>
<th>c1 : Currency</th>
<th>c2 : Currency</th>
</tr>
</thead>
<tbody>
<tr>
<td>title = &quot;Euro&quot;</td>
<td>title = &quot;U.S. Dollar&quot;</td>
</tr>
<tr>
<td>code = &quot;EUR&quot;</td>
<td>code = &quot;USD&quot;</td>
</tr>
<tr>
<td>decimalPlaces = &quot;2&quot;</td>
<td>decimalPlaces = &quot;2&quot;</td>
</tr>
<tr>
<td>value = &quot;1.0000&quot;</td>
<td>value = &quot;1.3286&quot;</td>
</tr>
<tr>
<td>symbolRight = &quot;€&quot;</td>
<td>symbolLeft = &quot;$&quot;</td>
</tr>
</tbody>
</table>
```
Location & Taxes

Overview

In order to supply a flexible use of taxes, product prices are stored tax free. This allows the final price to be calculated 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. osCommerce allows the store administrator to set 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 to group **Zones** with the same tax regulation.

**Tax rates** specify the tax percentage that is used in a TaxZone 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 by adding each rate together when the priorities are the same, or compounding the rates together in the defining priority order.

### Examples

Value Added Tax (VAT) in the European Union is a general and indirect consumption tax assessed on the value added to goods and services. Currently, applied rates 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 shows how osCommerce deals with the three types of VAT for a Spanish online store:
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, customers should pay a 7% and 7.5% compounded tax. That is, customers should add 15.025% of taxes \((1.075 \times 0.07 + 0.075 = 0.15025)\) to the price of the product.

The following instantiation shows how osCommerce deals with Canadian Federal Tax and Quebec Local Tax:
8.5 STORE ADMINISTRATION

Products

- Overview

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

- Structural Schema

![Diagram of the product schema]

- Operations

```plaintext
context Product def:
addTaxes(z:Zone, basePrice:Money) : Money =
let appliedTaxRates:Set(TaxRate) =
z.taxZone.taxRate -> select (tr | tr.taxClass = self.taxClass)->any(true)
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 selfoclIsTypeOf(Special) then
if selfoclAsType(Special).specialStatus=Status::enabled and
selfoclAsType(Special).expiryDate < Now()
then selfoclAsType(Special).specialPrice
else set()
else set()
endif
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 from when 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 track of how many products have been sold.

- **Model:** An additional information field for products. It can be used, for example, to specify the product model number.

- **Image path:** Every product can be associated with 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. In the case of a product is not associated with a category, it is assumed that it belongs at the top of the 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 more information about the product can be found.

- **Times viewed:** This attribute 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 following is an example instantiation of a film, sold on DVD, which can be found as a product in some of those shops:

```plaintext
theDaVinciCode : Product
  status = inStock
  available = "25/10/2006 00:00"
  netPrice = "11.95"
  grossPrice = "13.88"
  quantityOnHand = 120
  quantityOrdered = "51"
  imagePath = "/images/dvd/davincicode.jpg"
  weight = "0.15"
  added = "20/10/2006 12:34"

sonyPictures : Manufacturer

dvd : Category

en : Language

theDaVinciCode : Language
  name = "The Da Vinci Code"
  url = "www.sonypictures.com/movies/thedavincicode"

sp : Language

theDaVinciCode : Language
  name = "El código Da Vinci"
  url = "www.elcodigodavinci-laspelicula.com"
```
Product attributes and options

Overview

*osCommerce* allows several attributes to be defined 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

![Diagram of product attributes and options]

Constraints

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

```plaintext
context Language::optionNamesUnique(): Boolean
body : self.hasOptionName -> isUnique(optionName)
```
In each language, each product value has a unique name.

```
context Language::valueNamesUnique(): 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/Neutral.

Product attributes can be enabled or disabled by 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 added or subtracted 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 for downloadable products. In this case, the system requires information about:

- **File Name**
- **Expiry days**: The number of days the download will be enabled for after the product was ordered.
- **Maximum downloads**: The number of times the customer can download the product.

There are configuration properties which specify a general value for these 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 retailers:
A T-Shirt which can be bought in two sizes: large or small.

```
pa1 : ProductAttribute
increment = "5"
  sign = plus

largeSize : Attribute
  large : Value

size : Option

smallSize : Attribute
  small : Value

pa2 : ProductAttribute
increment = "2"
  sign = minus

FashionT-Shirt : Product
```


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::categoryNamesUnique(): 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 with 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 cases where the sort order is the same, they are displayed in alphabetical ordered.

- **Added**: The DateTime when the category was created. *Derived attribute.*

- **Last modified**: The last time 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 toy 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

The osCommerce store administrator can offer specials, that is, lower prices for a set of products can be offered during a specific time period.

- Structural Schema

```
Product

Special

- specialPrice : Money
- expiryDate : DateTime [0..1]
- specialAdded : DateTime (<constant>)
- specialLastModified : DateTime [0..1]
- specialStatus : Status
- dateStatusChanged : DateTime [0..1]

<<enumeration>>
- Status
  - enabled
  - disabled

<<dataType>>
- Money
```

- 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, at lower price over a period of time. The information kept by the system about specials is:

- **Special price**: The product’s price during the special offer.
• **Expiry date:** The date until which the special offer remains 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 instantiation corresponding to one of their special offers:

```plaintext
<table>
<thead>
<tr>
<th>goldWingedHorusNecklace : Special</th>
</tr>
</thead>
<tbody>
<tr>
<td>status = inStock</td>
</tr>
<tr>
<td>available = &quot;25/10/2006 00:00&quot;</td>
</tr>
<tr>
<td>netPrice = &quot;17.40&quot;</td>
</tr>
<tr>
<td>grossPrice = &quot;20.48&quot;</td>
</tr>
<tr>
<td>quantityOnHand = 2</td>
</tr>
<tr>
<td>quantityOrdered = &quot;1&quot;</td>
</tr>
<tr>
<td>imagePath ==&quot;/images/necklaces/goldWingedHorusNecklace.jpg&quot;</td>
</tr>
<tr>
<td>weight = &quot;0.16&quot;</td>
</tr>
<tr>
<td>added = &quot;20/10/2006 12:34&quot;</td>
</tr>
<tr>
<td>specialPrice = &quot;1400&quot;</td>
</tr>
<tr>
<td>specialStatus = enabled</td>
</tr>
<tr>
<td>specialAdded = &quot;17/02/2007 15:32&quot;</td>
</tr>
</tbody>
</table>
```
Manufacturers

Overview

In osCommerce, the products in the store are produced by manufacturers.

Structural Schema

Derivation Rules

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

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

Constraints

[1] A manufacturer is identified by its name

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

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

callback: 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 an online German wine shop. This osCommerce based online store sells wines of the Penedès region, 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 to be posted on the online store.

■ Structural Schema

![Diagram of BannerGroup and Banner entities]

■ Derivation Rules

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

custom 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 the store administrator to administrate the current banners and add 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 automatically updates the last time when the banner status was modified.
- **Status**: The banner can be enabled or disabled.

Banners grouped under 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 on every day.

**Example**

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

- Overview

*osCommerce* allows store owners to send emails and product notifications to customers.

- Structural Schema

```
<table>
<thead>
<tr>
<th>Newsletter</th>
</tr>
</thead>
<tbody>
<tr>
<td>title : String</td>
</tr>
<tr>
<td>content : String</td>
</tr>
<tr>
<td>added : DateTime (&lt;&lt;constant&gt;&gt;)</td>
</tr>
<tr>
<td>sent : DateTime [0..1]</td>
</tr>
<tr>
<td>status : NewsletterStatus</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>ProductNotification</th>
</tr>
</thead>
<tbody>
<tr>
<td>global : Boolean</td>
</tr>
<tr>
<td>explicitRelatedProduct</td>
</tr>
<tr>
<td>explicitNotifications</td>
</tr>
<tr>
<td>/ relatedProduct</td>
</tr>
<tr>
<td>/ notifications</td>
</tr>
</tbody>
</table>
```

```
<<enumeration>>
NewsletterStatus
locked
unlocked
```

- 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 their 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 automatically saves 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. A newsletter can only be sent if it is locked.
- **Status**: The banner can be enabled or disabled.

Newsletters are sent to all the customers who selected the option 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:

```
pt1 : ProductNotification
   title = "Frankfurt 2007"
   content = "Catalan culture will be the guest of honour at the 2007 Frankfurt Book Fair. Visit our online catalan books section."
   global = false
   added = "23/04/2007 12:00"
   status = unlocked

aucaSenyorEsteve : Product
   explicitNotifications
   notifications

tirantLoBlanc : Product
   explicitNotifications
   notifications
```
8.6 CUSTOMERS

Customers

- Overview

*osCommerce* saves 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::eMailsUnique(): 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 subscribe 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, saves the active subscriptions to product notifications associated with 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 delivery address 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:
8.7 ONLINE CATALOG

Reviews

- Overview

Customers can write reviews so that other users may read evaluations of a product.

- Structural Schema

| Review | 1 | Language
|        |   | Languages 57
|        | 1 | Product
|        |   | Products 65
|        |   | Customer
|        | 1 | Customers 85

<enumeration>
Rating
oneStar
twoStars
threeStars
fourStars
fiveStars

- Derivation Rules

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

countext Review::added():DateTime
tbody: 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 automatically updates 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 by some travel and tourism online shops. In these online shops, travel packages are sold as products.

Usually, in shops of this type, users write reviews giving their impressions about the hotels where they stayed while traveling.

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

```
pw1 : Review
review = "Very easy to find the hotel near Notting Hill gate. Generally very polite and helpful people in the area."
        rating = fourStars
timesRead = 2
        added = "08/10/2006"

pw2 : Review
rating = twoStars
review = "Muy bien localizado en Londres, al lado del mercado de Porto Bello. Es un hotel con una distribución extraña al ocupar varios edificios lo que hace que el laberinto de pasillos sea de lo más divertido. El personal es distante."
        added = "25/01/2007"
```
Shopping carts

■ Overview

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

■ Structural Schema

![Structural Schema Diagram]

■ Derivation Rules

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

```plaintext
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 contain the products chosen by customers from the online catalog.

A shopping cart is anonymous until the 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 the 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 modeled 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, rather, it will be restored when the customer initiates a new session.
Orders

- Overview

Orders are the confirmation that customers want to buy the contents of their 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) -> any(true)
    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

class ShippingMethod def:

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

class FlatRate def:

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

class PerItem def:

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

class 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

class USPostalService def:

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

class 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.

class 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.

class Order::primary():Address
body: self.customer.primary
[3] **Order::eMailAddress** of an order is that of its customer.

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

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

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

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

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

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

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

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

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

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

```plaintext
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.oclsIsTypeOf(HandlingFeeMethod)
      then self.shippingMethod.cclAsType(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 -> any(true)

[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 -> any(true)
[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 -> any(true)
```

[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 -> any(true)
```

[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 -> any(true)
```

### 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::IDsIsUnique: Boolean
body: Order.allInstances() -> isUnique(id)

[6] Order status are identified by its name

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

### Description

When customers confirms that they want to buy the contents of a 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 number. *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 the *check out* process, and can be in different *OrderStatus* at different times.