Telco Service Application

Raffaele Cicellini 10626081

Francesca Forbicini 10628756

Index

- Specification
- Conceptual (ER) and logical data models
  - Explanation of the ER diagram
  - Explanation of the logical model
- Trigger design and code
- ORM relationship design with explanations
- Entities code
- Interface diagrams or functional analysis of the specifications (textual notation)
- List of components
  - Motivations of the components design
- UML sequence diagrams
  - Observations

Specification

TELCO SERVICE APPLICATIONS

A telco company offers pre-paid online services to web users. Two client applications using the same database need to be developed.

CONSUMER APPLICATION

The consumer application has a public Landing page with a form for login and a form for registration. Registration requires a username, a password and an email. Login leads to the Home page of the consumer application. Registration leads back to the landing page where the user can log in.

The user can log in before browsing the application or browse it without logging in. If the user has logged in, his/her username appears in the top right corner of all the application pages.

The Home page of the consumer application displays the service packages offered by the telco company.

A service package has an ID and a name (e.g., "Basic", "Family", "Business", "All Inclusive", etc). It comprises one or more services. Services are of four types: fixed phone, mobile phone, fixed internet, and mobile internet. The mobile phone service specifies the number of minutes and SMSs included in the package plus the fee for extra minutes and the fee for extra SMSs. The mobile and fixed internet services specify the number of Gigabytes included in the package and the fee for extra Gigabytes. A service package must be associated with one validity period. A validity period specifies the number of months (12, 24, or 36). Each validity period has a different monthly fee (e.g., 20€/month for 12 months, 18€/month for 24 months, and 15€ /month for 36 months). A package may be associated with one or more optional products (e.g., an SMS news feed, an internet TV channel, etc.). The validity period of an optional product is the same as the validity period that the user has chosen for the service package. An optional product has a name and a monthly fee independent of the validity period duration. The same optional product can be offered in different service packages.

Specification

From the Home page, the user can access a Buy Service page for purchasing a service package and thus creating a service subscription. The Buy Service page contains a form for purchasing a service package. The form allows the user to select one package from the list of available ones and choose the validity period duration and the optional products to buy together with the chosen service. The form also allows the user to select the start date of his/her subscription. After choosing the service packages, the validity period and (0 or more) optional products, the user can press a CONFIRM button. The application displays a CONFIRMATION page that summarizes the details of the chosen service package, the validity period, the optional products and the total price to be pre-paid: (monthly fee of service package * number of months) + (sum of monthly fees of options * number of months).

If the user has already logged in, the CONFIRMATION page displays a BUY button. If the user has not logged in, the CONFIRMATION page displays a link to the login page and a link to the REGISTRATION page. After either logging in or registering and immediately logging in, the CONFIRMATION page is redisplayed with all the confirmed details and the BUY button.

When the user presses the BUY button, an order is created. The order has an ID and a date and hour of creation. It is associated with the user and with the service package, its validity period and the chosen optional products. It also contains the total value (as in the CONFIRMATION page) and the start date of the subscription. After creating the order, the application bills the customer by calling an external service. If the external service accepts the billing, the order is marked as valid and a service activation schedule is created for the user. A service activation schedule is a record of the services and optional products to activate for the user with their date of activation and date of deactivation.

If the external service rejects the billing, the order is put in the rejected status and the user is flagged as insolvent. When an insolvent user logs in, the home page also contains the list of rejected orders. The user can select one of such orders, access the CONFIRMATION page, press the BUY button and attempt the payment again. When the same user causes three failed payments, an alert is created in a dedicated auditing table, with the user Id, username, email, and the amount, date and time of the last rejection.

Specification

EMPLOYEE APPLICATION

The employee application allows the authorized employees of the telco company to log in. In the Home page, a form allows the creation of service packages, with all the needed data and the possible optional products associated with them. The same page lets the employee create optional products as well.

A Sales Report page allows the employee to inspect the essential data about the sales and about the users over the entire lifespan of the application:

- Number of total purchases per package.
- Number of total purchases per package and validity period.
- Total value of sales per package with and without the optional products.
- Average number of optional products sold together with each service package.
- List of insolvent users, suspended orders and alerts.
- Best seller optional product, i.e. the optional product with the greatest value of sales across all the sold service packages.

Assumptions on the specification

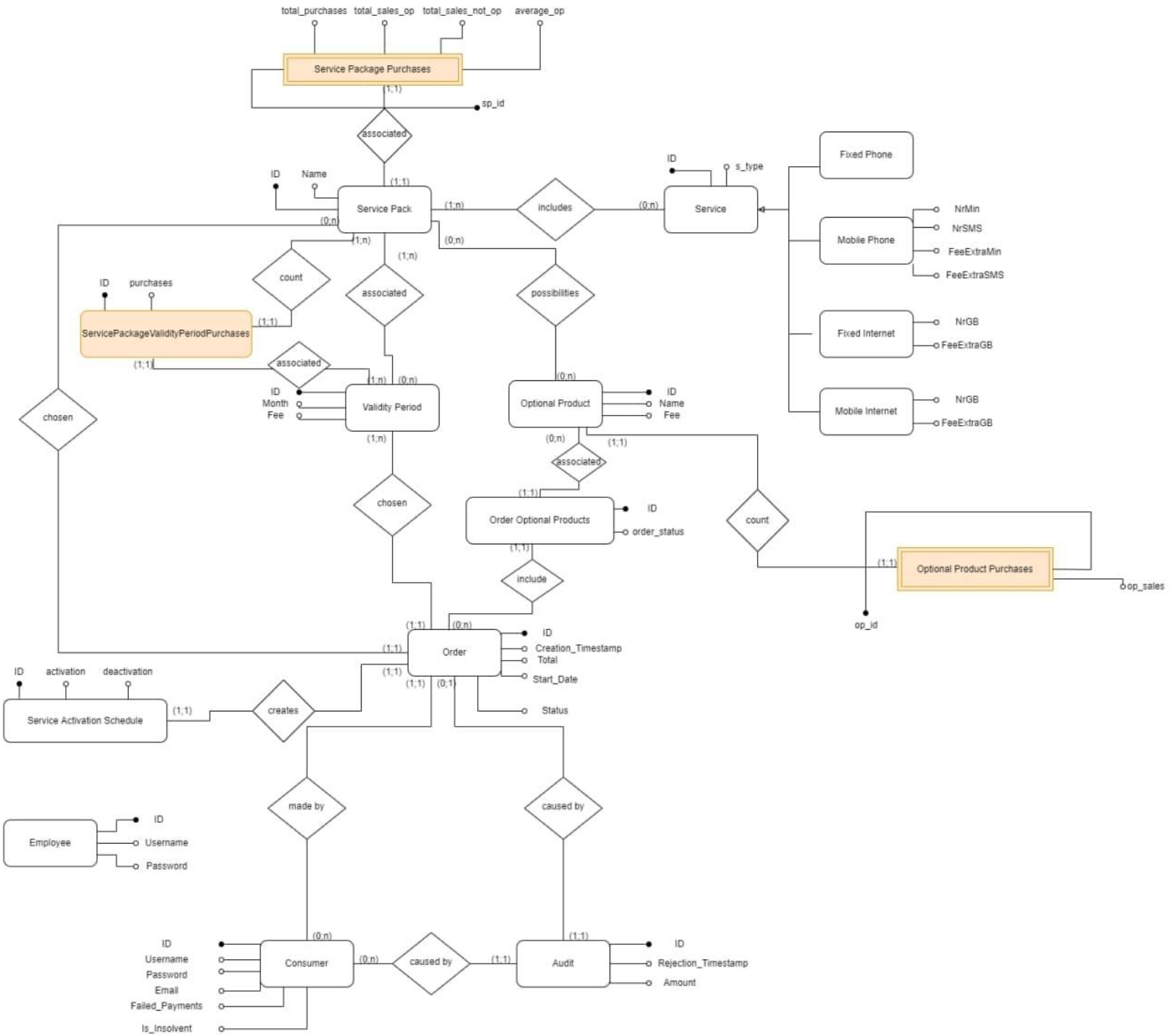
- We assumed that the subscription was not an entity of the db, so we simply used it to store consumer's selection of service package, optional products and validity period in the web session.
- We assumed that failed payments of a consumer can be used as a sort of warning for the company that the customer in the past made some invalid payments. So, we clean his «criminal» record (decrementing the total number of failed payments) every time he makes a successful payment related or not to a previous invalid order
- For the same reason of the previous point, we assumed that an audit for a consumer will be removed from the table only when a consumer has no more failed payments.

Explanation of the ER diagram

Legenda:

- The orange entities are the entities whose tables are populated by triggers and the explanation of their schema is in the Trigger Section
- The entities with the double block are the **weak entities**.
- The entity 'OrderOptionalProducts' has been created to avoid problems with JPA mapping, because of the column «order_status». The column was needed in two triggers to check the update of the status of an order

the **Entity Diagram** is in the next slide →


total_purchases
total_sales_op
total_sales_not_op
average_op
Service Package Purchases
(1;1)
sp_id
associated
ID
Name
(1;1)
Service Pack
(1;n)
includes
(0;n)
Service
ID
s_type
Fixed Phone
Mobile Phone
NrMin
NrSMS
FeeExtraMin
FeeExtraSMS
Fixed Internet
NrGB
FeeExtraGB
Mobile Internet
NrGB
FeeExtraGB
(0;n)
(1;n)
count
associated
possibilities
ID
purchases
ServicePackageValidityPeriodPurchases
(1;1)
(1;1)
associated
(1;n)
(0;n)
ID
Month
Fee
Validity Period
(1;n)
Optional Product
ID
Name
Fee
(0;n)
chosen
(0;n)
(1;1)
associated
(1;1)
Order Optional Products
ID
order_status
(1;1)
include
chosen
count
(1;1)
Optional Product Purchases
op_sales
op_id
(1;1)
(0;n)
Order
ID
Creation_Timestamp
Total
Start_Date
Status
(1;1)
(1;1)
(0;1)
ID
activation
deactivation
Service Activation Schedule
(1;1)
creates
made by
caused by
Employee
ID
Username
Password
(0;n)
(1;1)
ID
Username
Password
Email
Failed_Payments
Is_Insolvent
Consumer
(0;n)
caused by
(1;1)
Audit
ID
Rejection_Timestamp
Amount

Relation Model

Audit (id, rejection_timestamp, order_id, consumer_id, amount)

Consumer (id, username, password, email, failed_payments, is_insolvent)

Optional Product (id, name, fee)

Order (id, creation_timestamp, total, start_date, status, consumer_id, sp_id, vp_id)

Order OptionalProduct(id, order_id, optprod_id, orders_status)

ValidityPeriod(id, months, fee)

Service (id, nr_min, nr_sms, fee_extramin, fee_extrasms, nr_gb, fee_extragb, s_type)

Service Pack (id, name)

ServicePack_OptionalProducts(sp_id, optprod_id)

ServicePack_Service(sp_id, service_id)

ServicePack_ValidityPeriod(sp_id, vp_id)

Service Activation Schedule (id, order_id,activation,deactivation)

Motivations of the logical design

As we specify in the E-R diagram, the relational model for the tables populated by triggers is in the Trigger Section

Trigger design & code

- For all triggers high level specification of
  - Event
  - Condition
  - Action
  - SQL code of the trigger
- Trigger design motivation
  - Termination and triggering cycle

Trigger design & code

- package_validityperiod_purchases
  - After_insert
- op_purchases
  - After_insert
  - After_update
- package_purchases
  - After_insert

package_validityperiod_purchases

When a service activation schedule is inserted, this trigger updates or inserts the tuple into its associated table and calculates the purchases of service package and validity period that have been selected by the schedule which caused the event.

SQL DDL:

```
CREATE TABLE `package_validityperiod_purchases` (
 `id` int NOT NULL AUTO_INCREMENT,
 `sp_id` int NOT NULL,
 `vp_id` int NOT NULL,
 `purchases` int NOT NULL,
 PRIMARY KEY (`id`),
 KEY `spid_fk_idx` (`sp_id`),
 KEY `vpid_fk_idx` (`vp_id`),
 CONSTRAINT `spid_fk` FOREIGN KEY (`sp_id`) REFERENCES `service_package` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,
 CONSTRAINT `vpid_fk` FOREIGN KEY (`vp_id`) REFERENCES `validity_period` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
```

package_validityperiod_purchases_AFTER_INSERT

Event «Insertion of a service activation schedule» → trigger checks: if the tuple with key < service package, validity period> is already present, updates the purchase, otherwhise inserts the tuple with its purchase(1).

```
CREATE TRIGGER `package_validityperiod_purchases_AFTER_INSERT`
AFTER INSERT ON `service_activation_schedule`
FOR EACH ROW
BEGIN
    DECLARE spid INT;
    DECLARE vpid INT;
    SELECT o.sp_id INTO spid FROM `order` AS o WHERE o.id = new.order_id;
    SELECT o.vp_id INTO vpid FROM `order` AS o WHERE o.id = new.order_id;
    IF NOT exists (SELECT* FROM `package_validityperiod_purchases`
        WHERE `sp_id` = spid AND `vp_id` = vpid) THEN
          INSERT INTO `package_validityperiod_purchases` (sp_id, vp_id, purchases) VALUES (spid,vpid,1)
    ELSE
        UPDATE `package_validityperiod_purchases` SET `purchases` = `purchases`+ 1
                WHERE `sp_id` = spid AND `vp_id`=vpid;
    END IF;
END$$
```

op_purchases

When an order, with status 1, is associated with one or more optional products, this trigger inserts or updates the sales of each optional product in the associated table.

SQL DDL:

```
CREATE TABLE `op_purchases` (
  `op_id` int NOT NULL,
  `op_sales` int NOT NULL,
  PRIMARY KEY (`op_id`),
  CONSTRAINT `op_id` FOREIGN KEY (`op_id`) REFERENCES `optional_product` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

op_purchases_AFTER_INSERT

Event «an order with optional products has been inserted» → the trigger checks: if the tuple associated with the optional product is already present, updates its sales, otherwise inserts it with its sales.

```
CREATE TRIGGER `op_purchases_AFTER_INSERT`
AFTER INSERT ON `order_optprod`
FOR EACH ROW
BEGIN
    DECLARE op_fee INT;
    DECLARE vp_month INT;
    IF (new.order_status = 1) THEN
        SELECT op.fee INTO op_fee FROM `optional_product` AS op
            WHERE new.optprod_id=op.id;
        SELECT vp.months INTO vp_month FROM `validity_period` AS vp, `order` AS o
            WHERE o.vp_id=vp.id AND o.id=new.order_id;
        IF NOT EXISTS (SELECT * FROM `op_purchases` AS opp WHERE opp.op_id=new.optprod_id) THEN
            INSERT INTO `op_purchases` (op_id, op_sales) VALUES (new.optprod_id, op_fee*vp_month);
        ELSE
            UPDATE `op_purchases`
            SET `op_sales`= `op_sales` + (op_fee*vp_month)
            WHERE `op_id`=new.optprod_id;
        END IF;
    END IF;
END$$
```

op_purchases_AFTER_UPDATE

Event «an order with optional products changes status» → the trigger checks: if the tuple associated with the optional product is already present, updates its sales, otherwise inserts it with its sales.

```
CREATE TRIGGER `op_purchases_AFTER_UPDATE`
AFTER UPDATE ON `order_optprod`
FOR EACH ROW
BEGIN
    DECLARE op_fee INT;
    DECLARE vp_month INT;
    IF (new.order_status = 1) THEN
        SELECT op.fee INTO op_fee FROM `optional_product` AS op
            WHERE new.optprod_id=op.id;
        SELECT vp.months INTO vp_month FROM `validity_period` AS vp, `order` AS o
            WHERE o.vp_id=vp.id AND o.id=new.order_id;
        IF NOT EXISTS (SELECT * FROM `op_purchases` AS opp WHERE opp.op_id=new.optprod_id) THEN
            INSERT INTO `op_purchases` (op_id, op_sales) VALUES (new.optprod_id, op_fee*vp_month);
        ELSE
            UPDATE `op_purchases`
            SET `op_sales`= `op_sales` + (op_fee*vp_month)
            WHERE `op_id`=new.optprod_id;
        END IF;
    END IF;
END$$
```

package_purchases

When a service activation schedule is inserted, this trigger updates or inserts a service package with the total purchases, total sales (with and without optional products) and the average.

SQL DDL:

```
CREATE TABLE `package_purchases` (
  `sp_id` int NOT NULL,
  `total_purchases` int NOT NULL,
  `total_sales_op` int DEFAULT NULL,
  `total_sales_not_op` int DEFAULT NULL,
  `average_op` float DEFAULT NULL,
  PRIMARY KEY (`sp_id`),
  CONSTRAINT `sp_id` FOREIGN KEY (`sp_id`) REFERENCES `service_package` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

package_purchases_AFTER_INSERT

Event «a service activation schedule has been inserted» → the trigger checks: if the tuple associated with the service package (of the inserted schedule) is already present, updates the total purchases, the total sales of the service package (with and without optional products) and the average , otherwise inserts it with all the attributes.

The **implementation** is in the next slide →

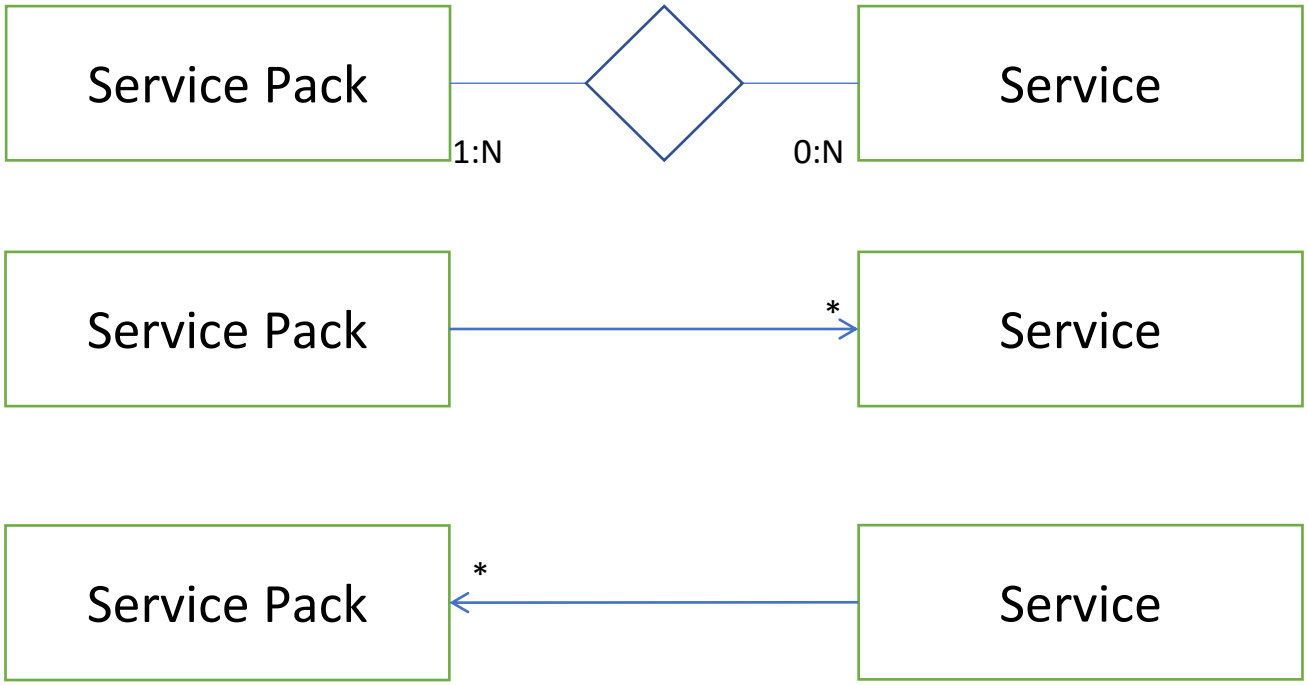
```
CREATE TRIGGER `package_purchases_AFTER_INSERT`
AFTER INSERT ON `service_activation_schedule`
FOR EACH ROW
BEGIN
    DECLARE op_fees INT;
    DECLARE vp_month INT;
    DECLARE count_op INT;
    DECLARE old_purchases INT;
    DECLARE spid INT;
    DECLARE total INT;
    SELECT o.sp_id INTO spid FROM `order` AS o WHERE new.order_id = o.id;
    SELECT o.total INTO total FROM `order` AS o WHERE new.order_id = o.id;
    SELECT coalesce(sum(op.fee), 0) INTO op_fees FROM `optional_product` AS op, `order_optprod` AS oop
        WHERE new.order_id=oop.order_id AND oop.optprod_id=op.id;
    SELECT vp.months INTO vp_month FROM `validity_period` AS vp, `order` AS o
        WHERE o.vp_id=vp.id AND new.order_id=o.id;
    SELECT coalesce(count(*), 0) INTO count_op FROM `order_optprod` AS oop, `order`  AS o
        WHERE o.sp_id = spid AND o.id = oop.order_id;
    SELECT pp.total_purchases INTO old_purchases FROM `package_purchases` AS pp, `order` AS o
        WHERE o.sp_id=pp.sp_id AND new.order_id=o.id;
    IF NOT EXISTS (SELECT * FROM `package_purchases` AS pp WHERE pp.sp_id=spid) THEN
        INSERT INTO `package_purchases` (sp_id, total_purchases, total_sales_op, total_sales_not_op, average_op)
            VALUES (spid, 1, total, total-(op_fees*vp_month), count_op);
    ELSE
        UPDATE `package_purchases`
        SET `total_purchases`=old_purchases + 1, `total_sales_op`= `total_sales_op` + total,
            `total_sales_not_op`= `total_sales_not_op` + (total - (op_fees*vp_month)),
            `average_op`=count_op/(old_purchases + 1)
        WHERE `sp_id`=spid;
    END IF;

END$$
```

Termination and triggering cycle

Each trigger modifies a specific table on which no other triggers are defined. So there are no cycles in the triggering graph and all the triggers are guaranteed to terminate.

ORM design

Relationship «includes»



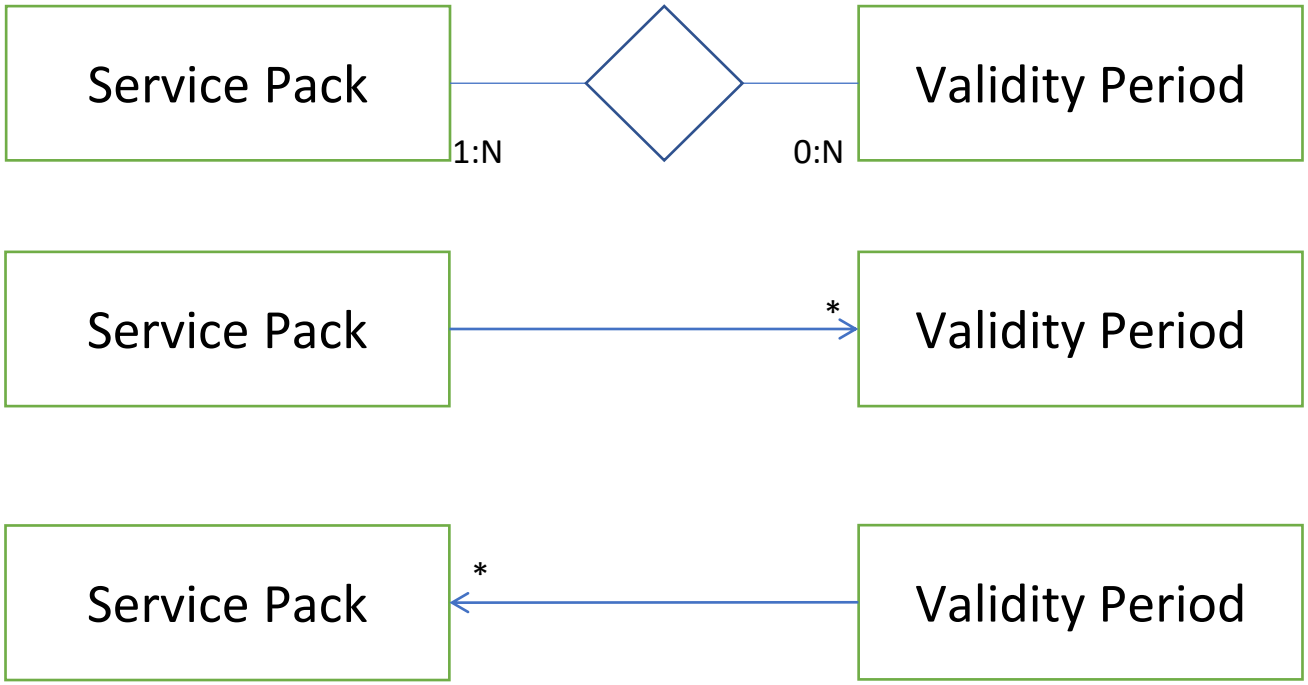
- ServicePack →Service

@ManyToMany is necessary to associate service packages to services

- Service →ServicePack

@ManyToMany not mapped

Relationship «associated»



- ServicePack →ValidityPeriod

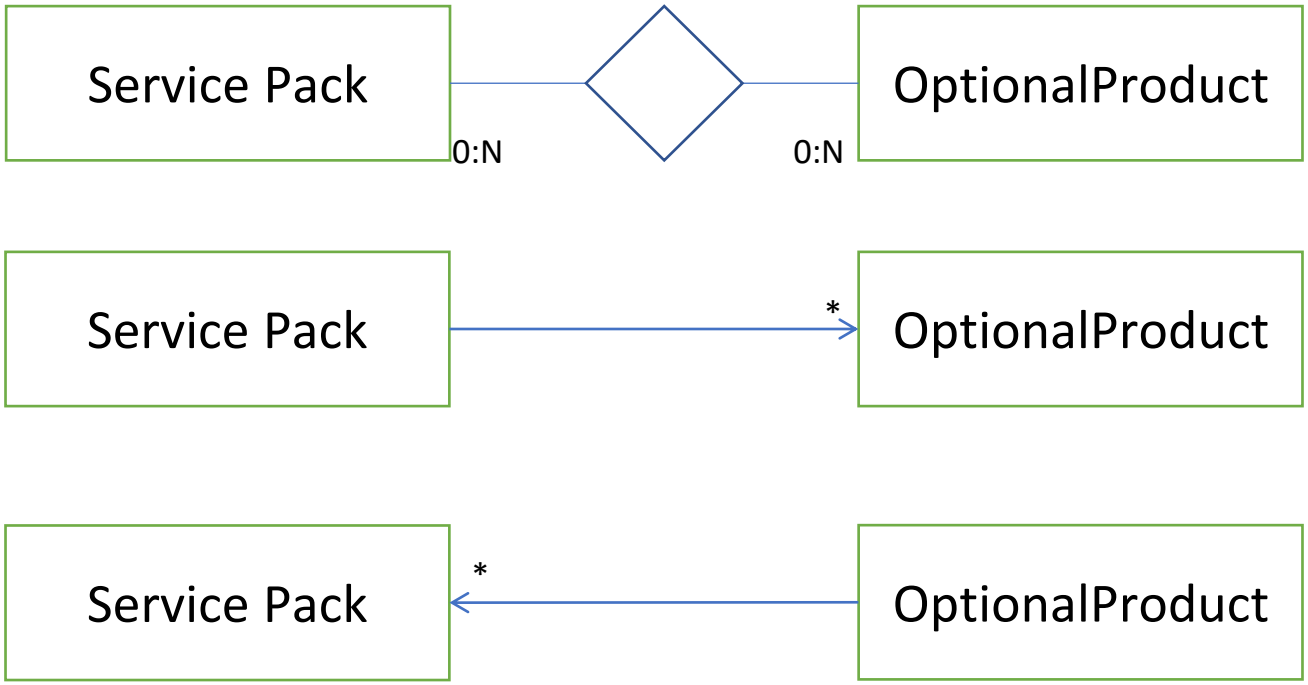
@ManyToMany is necessary to associate service packages to possible validity periods

- ValidityPeriod → ServicePack

@ManyToMany

not mapped

Relationship «possibilities»



- ServicePack→OptionalProduct

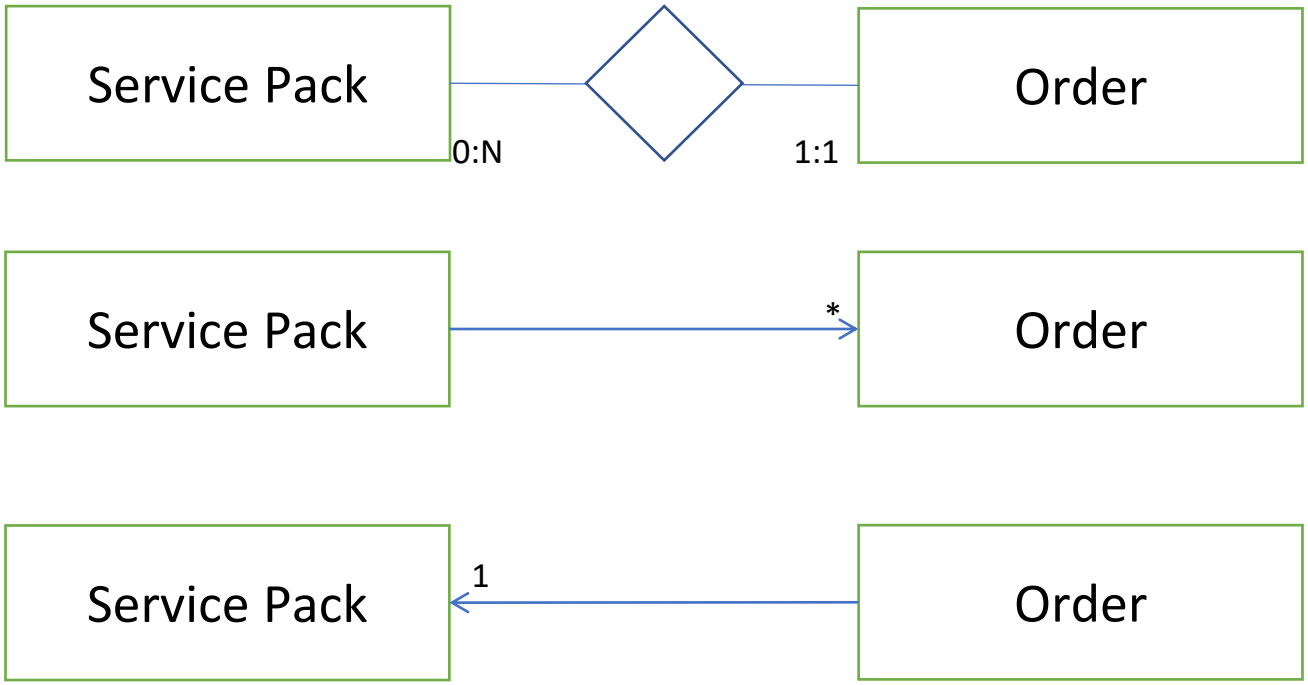
@ManyToMany is necessary to associate service packages to possible optional products

- OptionalProduct → ServicePack

@ManyToMany

not mapped

Relationship «chosen»



- ServicePack → Order

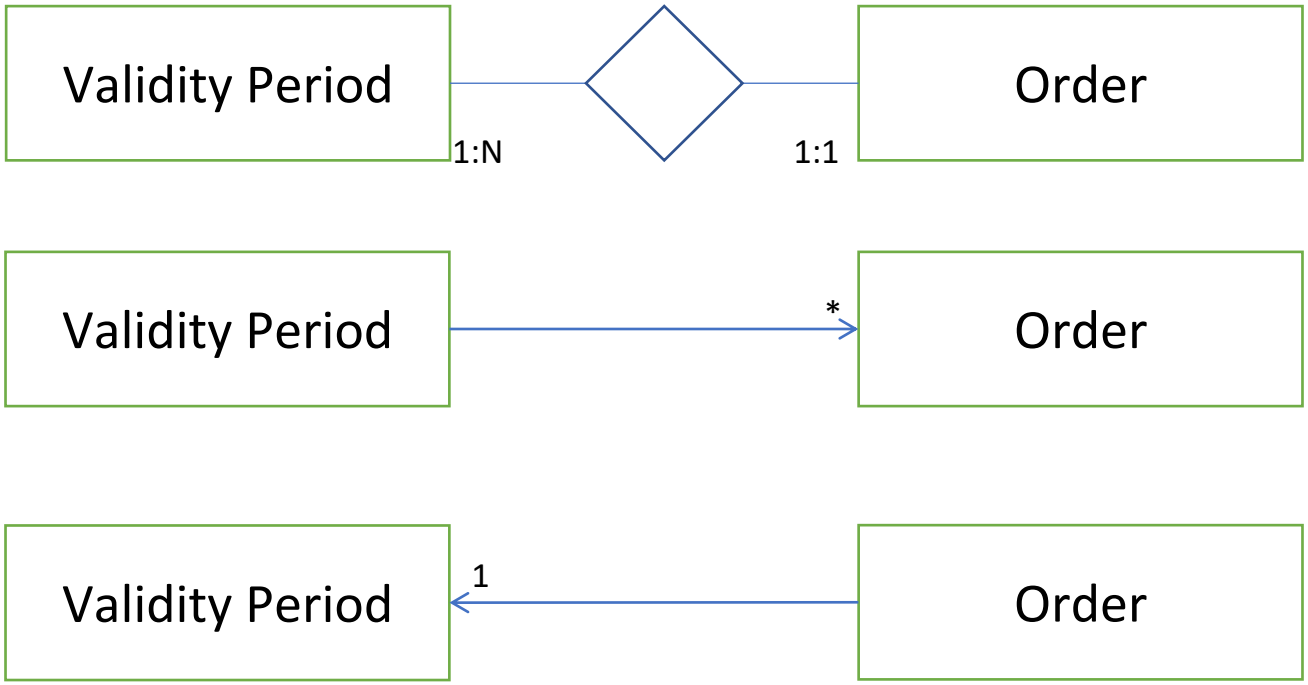
@OneToMany

not mapped

- Order →ServicePack

@ManyToOne is necessary to associate an order to the chosen service package

Relationship «chosen»



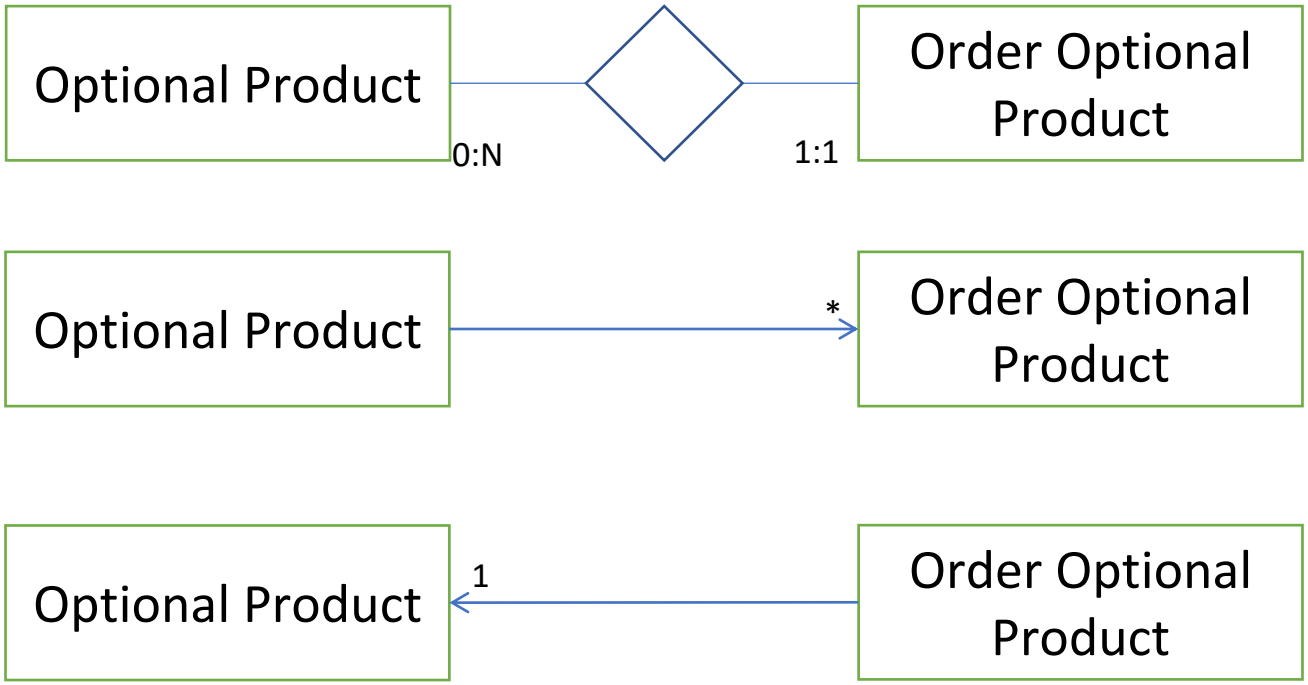
- Validity Period → Order

  not mapped

- Order→ Validity Period

@ManyToOne is necessary to associate an order to the chosen validity period of the service pack

Relationship «associated»



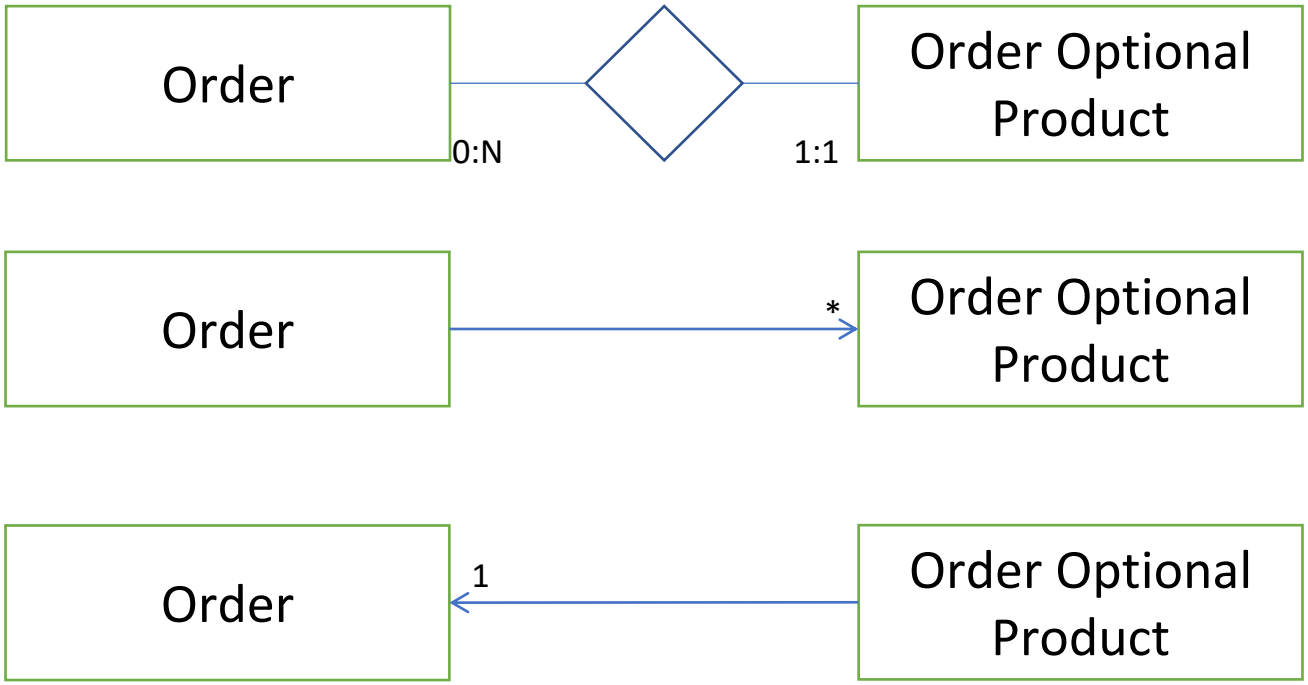
- OptionalProduct → Order Optional Product

not mapped

- Order Optional Product →OptionalProduct

@ ManyToOne is necessary to associate optional products to the order that contains them

Relationship «include»



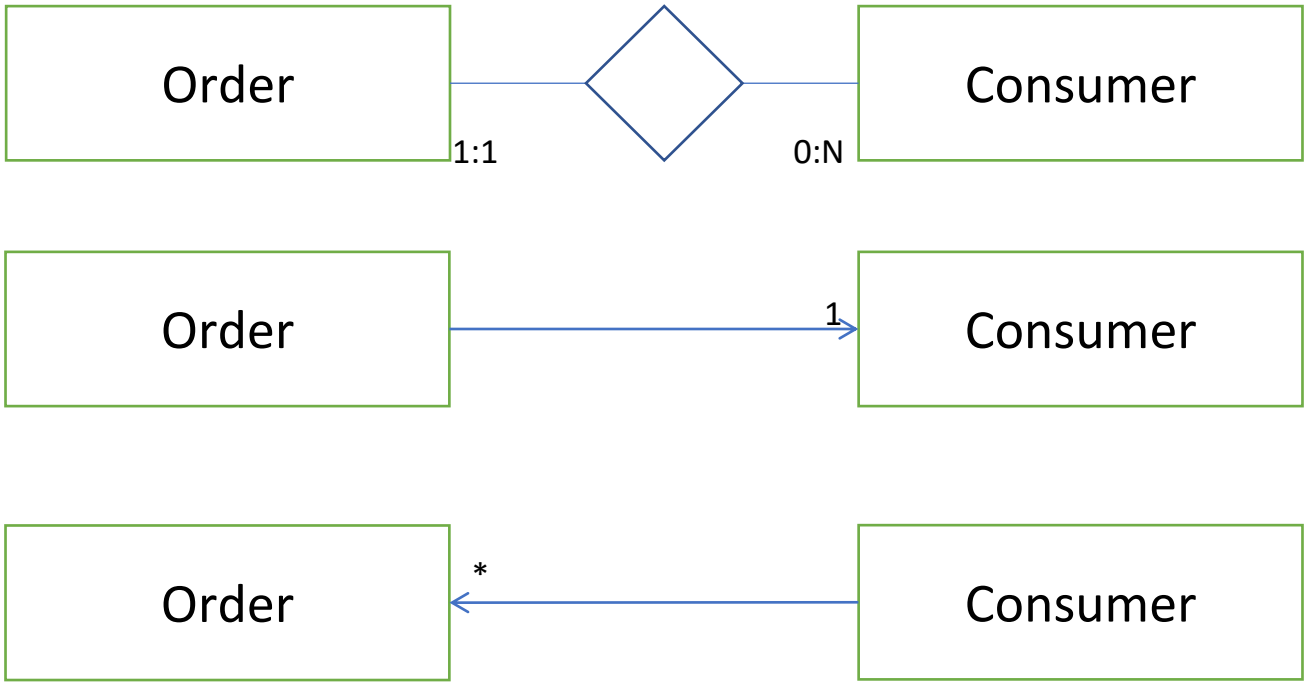
- Order → Order Optional Product

@OneToMany is necessary to associate an order to its optional products

- Order Optional Product →Order

@ ManyToOne is necessary to associate optional products to the correspondent order

Relationship «made by»



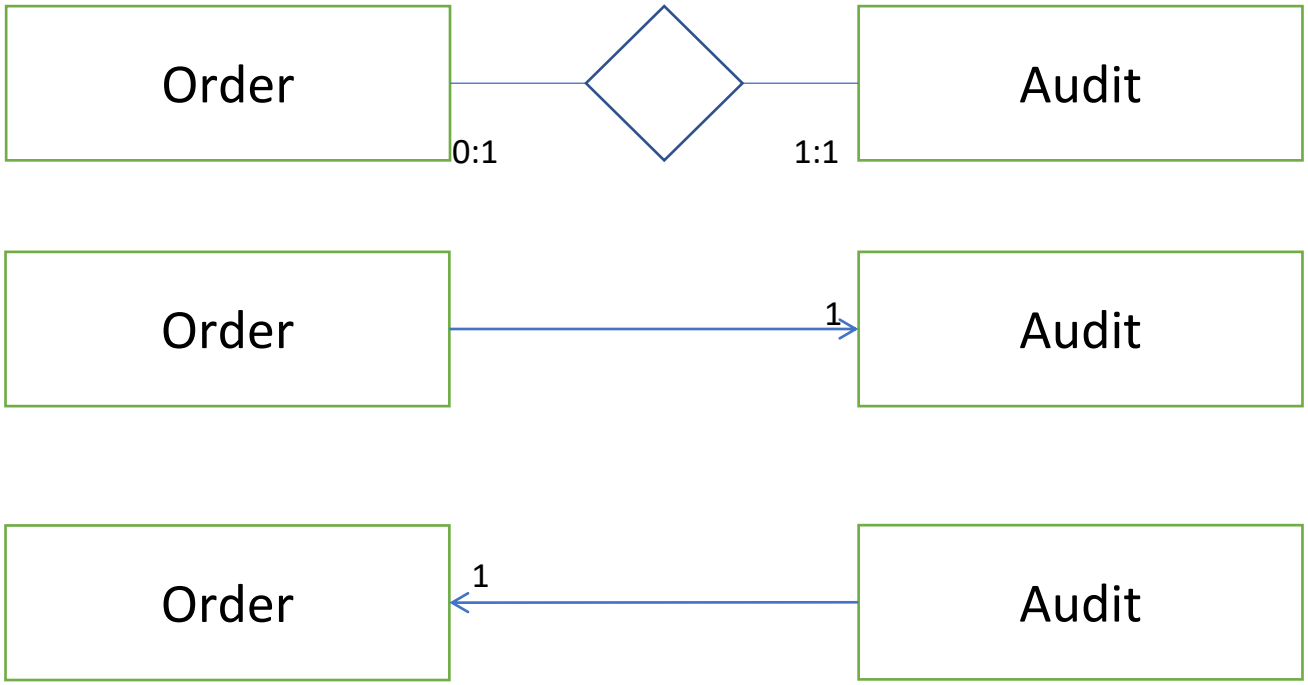
- Order → Consumer

@ManyToOne is necessary to associate orders to the correspondent user

- Consumer →Order

not mapped

Relationship «caused by»



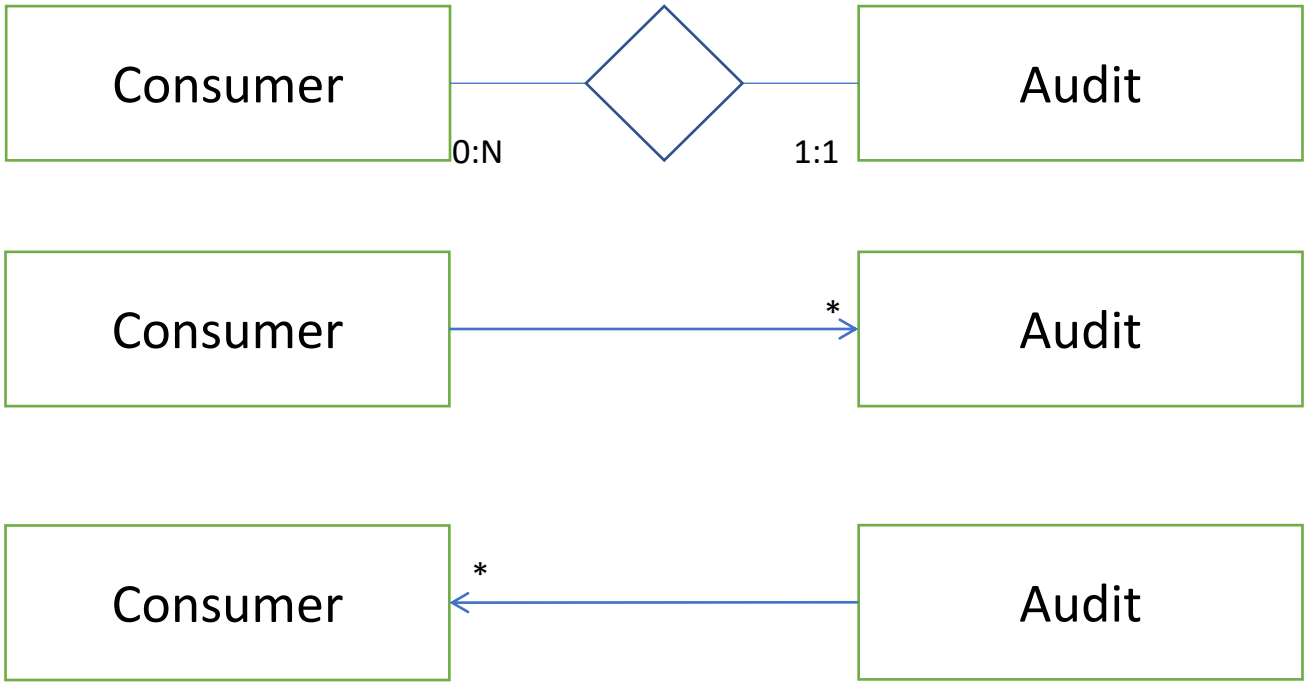
- Order →Audit

  not mapped

- Audit → Order

@ OneToOne is necessary to associate an audit to the order that caused it.

Relationship «caused by»



- Consumer →Audit

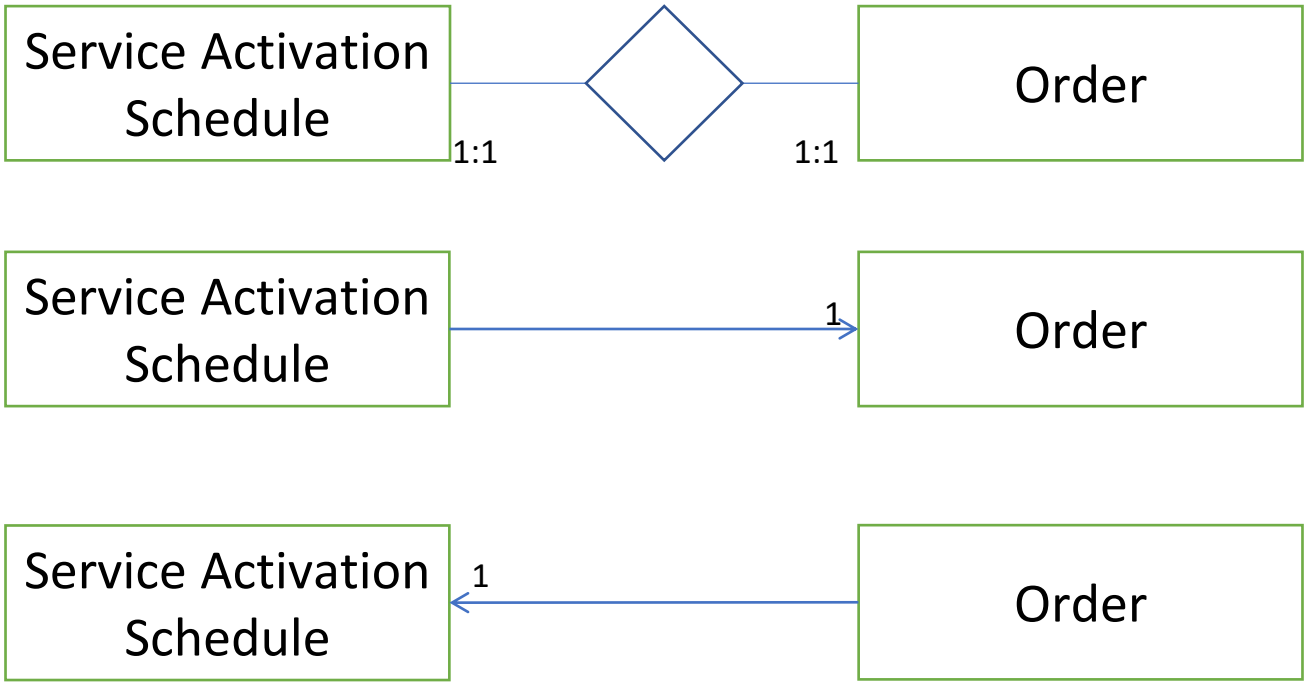
@OneToMany

not mapped

- Audit → Consumer

@ManyToOne is necessary to associate an audit with the consumer that caused it.

Relationship «creates»



- Order →ServiceActivationSchedule

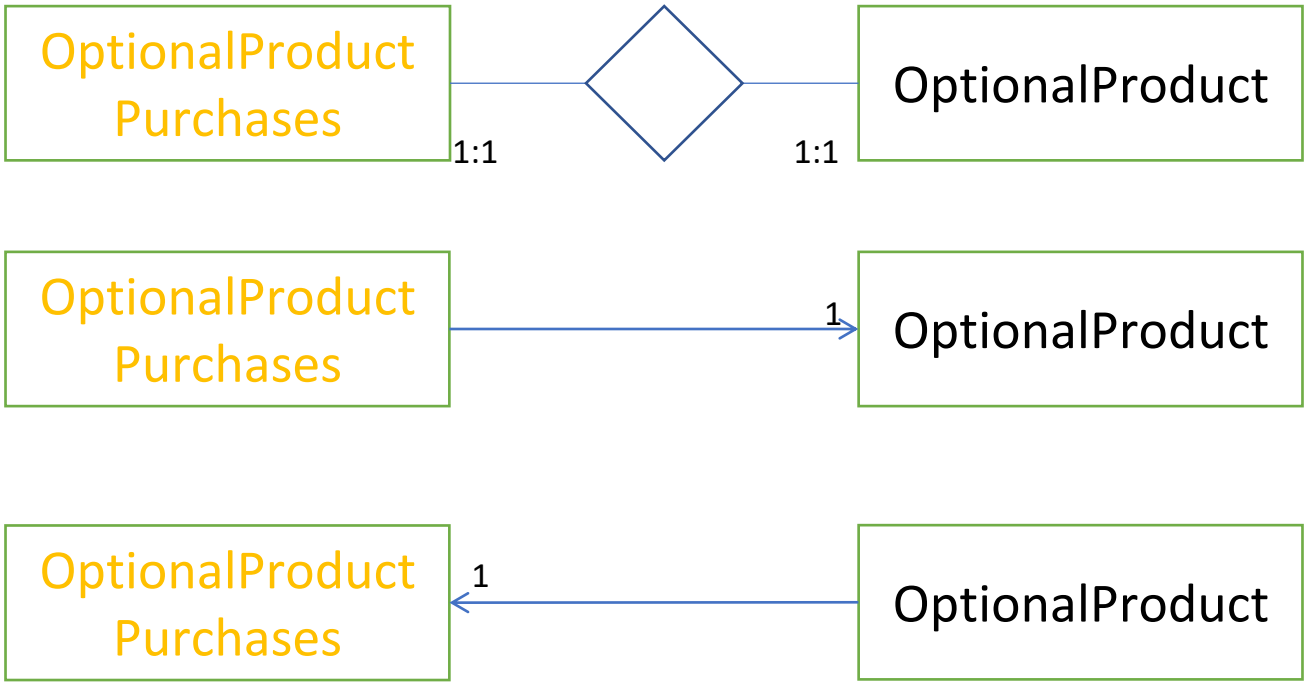
@OneToOne

not mapped

- ServiceActivationSchedule → Order

@OneToOne is necessary to associate a service activation schedule with the correspondent order

Relationship «count»



- OptionalProductPurchases→ OptionalProduct

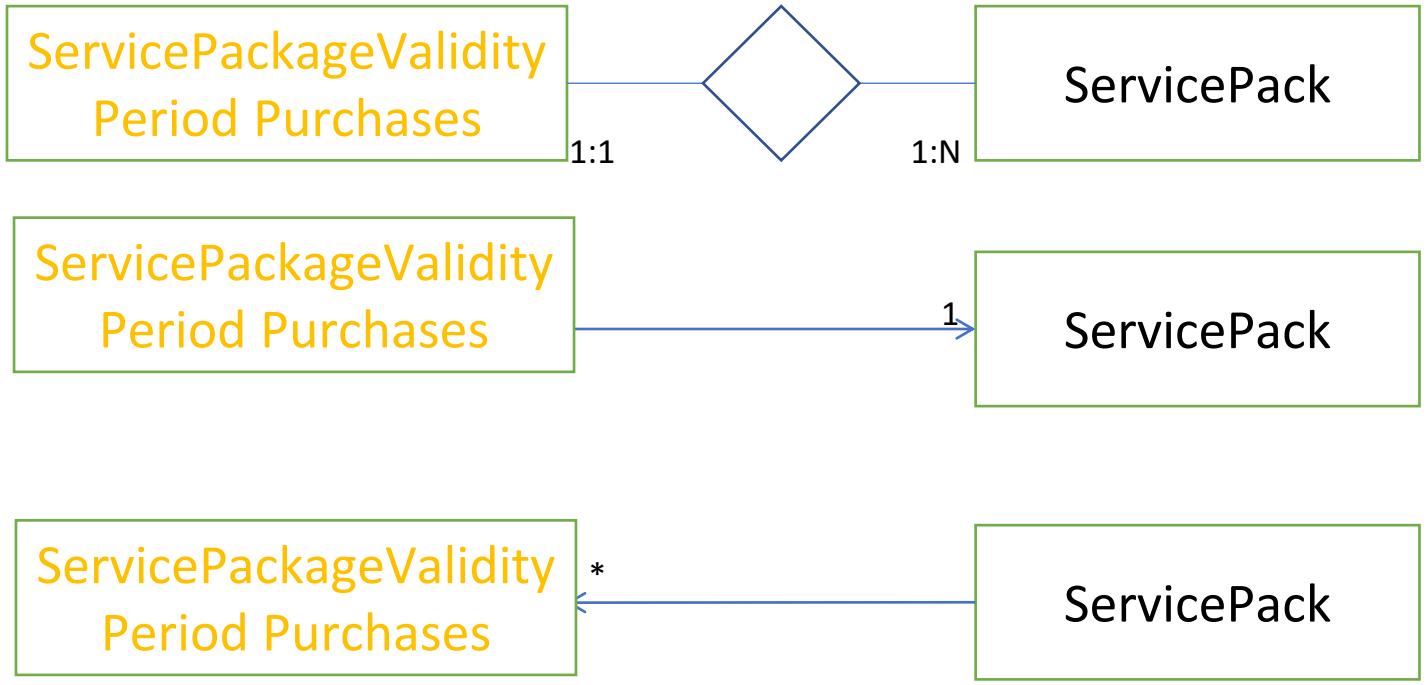
@OneToOne is necessary to associate a purchase to the correspondent optional product

- OptionalProduct → OptionalProductPurchases

@OneToOne

not mapped

Relationship «count»



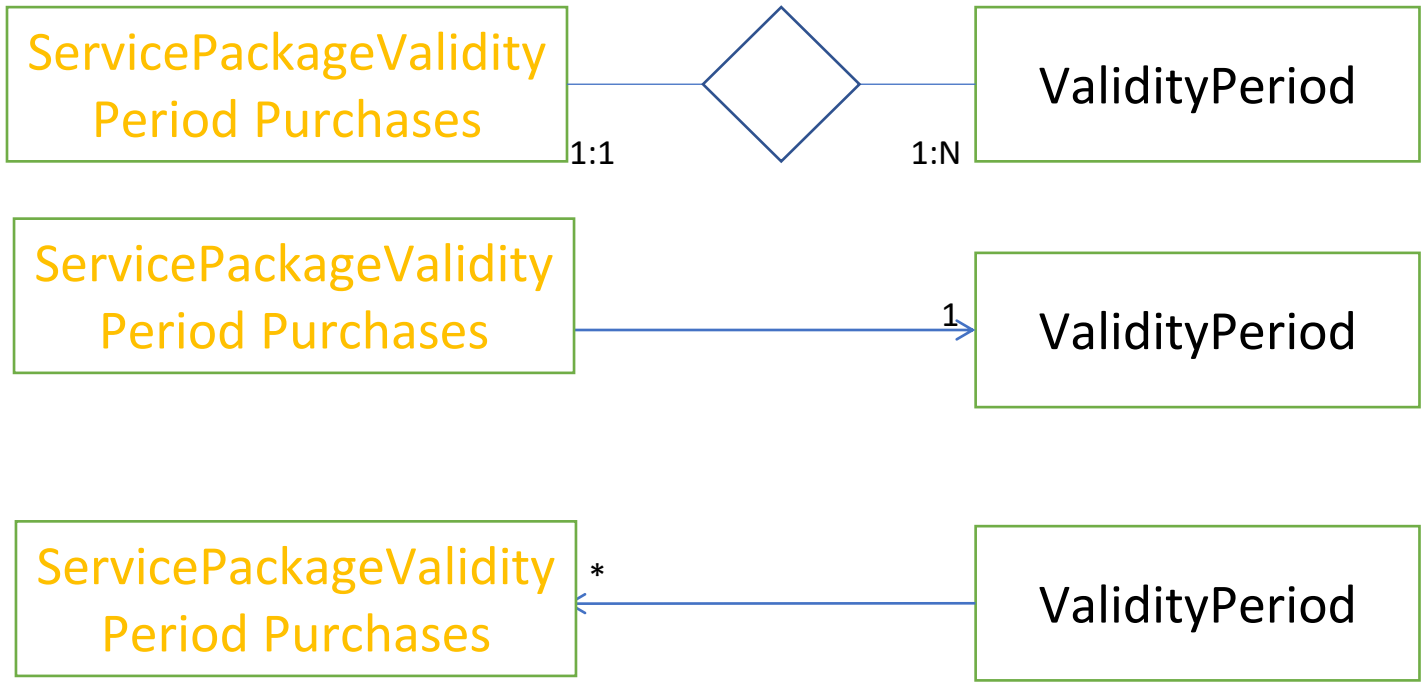
- ServicePack→ ServicePackageValidityPeriod

@OneToMany not mapped

- ServicePackageValidityPeriod → ServicePack

@ManyToOne is necessary to associate the purchases to the correspondent service pack

Relationship «associated»



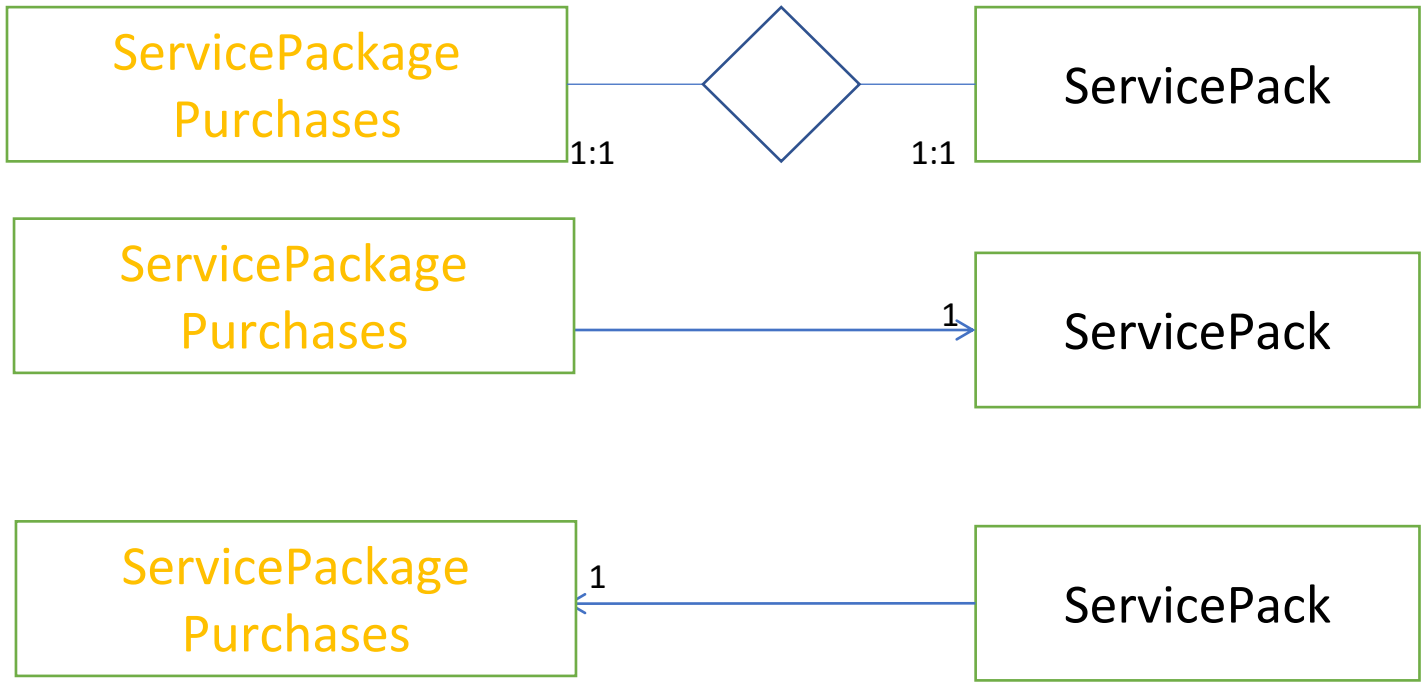
- ValidityPeriod → ServicePackageValidityPeriodPurchases

@OneToMany not mapped

- ServicePackageValidityPeriodPurchases → ValidityPeriod

@ManyToOne is necessary to associate the purchases to the correspondent validity period

Relationship «associated»



- ServicePack → ServicePackagePurchases

@OneToOne

not mapped

- ServicePackagePurchases→ ServicePack

@OneToOne is necessary to associate the purchases to the correspondent service pack

ORM design motivations

In the specification of the entity

- In some entities we omitted the setter and getter methods to simplify the design
- In the ‘Service’ entity, the tag :
  - @Inheritance specifies that this entity is the upper class of the other services type like FixedPhone …
  - @DiscriminatorColumn specifies the column that distinguishes between the subclasses of ‘Service’
- The orange entitities are those populated by the triggers and some of them are weak entities
  - The tag @PrimaryKeyJoinColumn is used to take the entire object and not only the attribute contained in the data base

Entity Consumer

```
@Entity
@Table(name = "consumer", schema = "db_test")
@NamedQueries({
    @NamedQuery(
            name = "Consumer.checkCredentials",
            query = "SELECT r FROM Consumer r  WHERE r.username = ?1 and r.password = ?2 and r.email = ?3" ),
    @NamedQuery(
            name = "Consumer.checkDuplicate",
            query = "SELECT r FROM Consumer r  WHERE r.username = ?1"),

})

public class Consumer implements Serializable{
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String username;
    private String password;
    private String email;
    private int is_insolvent;
    private int failed_payments;
```

Entity Service

```java
@Entity
@Table(name = "service", schema = "db_test")
//to declare the upper class
@Inheritance
@DiscriminatorColumn(name = "s_type")
public abstract class Service implements Serializable{
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Column(name="s_type")
    private String serviceType;
    @Column(name="nr_gb")
    private int numberGB;
    @Column(name="fee_extragb")
    private int feeExtraGB;
    @Column(name="nr_min")
    private int numberMinutes;
    @Column(name="nr_sms")
    private int numberSMS;
    @Column(name="fee_extramin")
    private int feeExtraMin;
    @Column(name="fee_extrasms")
    private int feeExtraSMS;
```

Entity FixedInternet

```java
@Entity
public class FixedInternet extends Service implements Serializable{
    private static final long serialVersionUID = 1L;
    @Column(name="nr_gb")
    private int numberGB;
    @Column(name="fee_extragb")
    private int feeExtraGB;

    @Override
    public void setNumberMinutes(int numberMinutes) {

    }

    @Override
    public void setNumberSMS(int numberSMS) {

    }

    @Override
    public void setFeeExtraMin(int feeExtraMin) {

    }

    @Override
    public void setFeeExtraSMS(int feeExtraSMS) {

    }
```

```java
@Override
public int getNumberGB() {
    return this.numberGB;
}

@Override
public void setNumberGB(int numberGB) {
    this.numberGB=numberGB;
}

@Override
public int getFeeExtraGB() {
    return this.feeExtraGB;
}

@Override
public void setFeeExtraGB(int feeExtraGB) {
    this.feeExtraGB=feeExtraGB;
```

Entity FixedPhone

```java
@Entity
public class FixedPhone extends Service implements Serializable{
    private static final long serialVersionUID = 1L;

    @Override
    public void setNumberMinutes(int numberMinutes) {

    }

    @Override
    public void setNumberSMS(int numberSMS) {

    }

    @Override
    public void setFeeExtraMin(int feeExtraMin) {

    }

    @Override
    public void setFeeExtraSMS(int feeExtraSMS) {

    }

    @Override
    public void setNumberGB(int numberGB) {

    }

    @Override
    public void setFeeExtraGB(int feeExtraGB) {

    }
```

Entity MobileInternet

```java
@Entity
public class MobileInternet extends Service implements Serializable{
    private static final long serialVersionUID = 1L;
    @Column(name="nr_gb")
    private int numberGB;
    @Column(name="fee_extragb")
    private int feeExtraGB;

    @Override
    public void setNumberMinutes(int numberMinutes) {

    }

    @Override
    public void setNumberSMS(int numberSMS) {

    }

    @Override
    public void setFeeExtraMin(int feeExtraMin) {

    }

    @Override
    public void setFeeExtraSMS(int feeExtraSMS) {

    }
```

```java
@Override
public void setFeeExtraMin(int feeExtraMin) {

}

@Override
public void setFeeExtraSMS(int feeExtraSMS) {

}

@Override
public int getNumberGB() {
    return this.numberGB;
}

@Override
public void setNumberGB(int numberGB) {
    this.numberGB=numberGB;
}

@Override
public int getFeeExtraGB() {
    return this.feeExtraGB;
}

@Override
public void setFeeExtraGB(int feeExtraGB) {
    this.feeExtraGB=feeExtraGB;
}
```

Entity MobilePhone

```java
@Entity
public class MobilePhone extends Service implements Serializable{
    private static final long serialVersionUID = 1L;
    @Column(name="nr_min")
    private int numberMinutes;
    @Column(name="nr_sms")
    private int numberSMS;
    @Column(name="fee_extramin")
    private int feeExtraMin;
    @Column(name="fee_extrasms")
    private int feeExtraSMS;

    @Override
    public int getNumberMinutes() {
        return this.numberMinutes;
    }

    @Override
    public void setNumberMinutes(int numberMinutes) {
        this.numberMinutes=numberMinutes;
    }

    @Override
    public int getNumberSMS() {
        return this.numberSMS;
    }
```

```java
    @Override
    public void setNumberSMS(int numberSMS) {
        this.numberSMS=numberSMS;
    }

    @Override
    public int getFeeExtraMin() {
        return this.feeExtraMin;
    }

    @Override
    public void setFeeExtraMin(int feeExtraMin) {
        this.feeExtraMin=feeExtraMin;
    }

    @Override
    public int getFeeExtraSMS() {
        return this.feeExtraSMS;
    }

    @Override
    public void setFeeExtraSMS(int feeExtraSMS) {
        this.feeExtraSMS=feeExtraSMS;
    }

    @Override
    public void setNumberGB(int numberGB) {

    }

    @Override
    public void setFeeExtraGB(int feeExtraGB) {
```

Entity ValidityPeriod

```
@Entity
@Table(name = "validity_period", schema = "db_test")
@NamedQuery(name = "ValidityPeriod.findAll", query = "SELECT vp FROM ValidityPeriod vp")

public class ValidityPeriod implements Serializable{
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private int months;
    private int fee;
```

Entity OptionalProduct

```
@Entity
@Table(name = "optional_product", schema = "db_test")
@NamedQuery(name = "OptionalProduct.findAll", query = "SELECT op FROM OptionalProduct op")

public class OptionalProduct implements Serializable{
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    private int fee;
```

Entity ServicePackage

```java
@Entity
@Table(name = "service_package", schema = "db_test")
@NamedQuery(name = "ServicePackage.findAll", query = "SELECT sp FROM ServicePackage sp")
public class ServicePackage implements Serializable{
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name="servicepack_service", joinColumns =@JoinColumn(name="sp_id"), inverseJoinColumns = @JoinColumn(name="service_id"))
    private List<Service> services;
    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name="servicepack_optprod", joinColumns = @JoinColumn(name="sp_id"), inverseJoinColumns = @JoinColumn(name = "optprod_id"))
    private List<OptionalProduct> optionalProducts;
    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name ="servicepack_validityperiod", joinColumns= @JoinColumn(name="sp_id"), inverseJoinColumns= @JoinColumn(name="vp_id"))
    private List<ValidityPeriod> validityPeriods;
```

Entity Order

```java
@Entity
@Table(name = "order", schema = "db_test")
@NamedQuery(name = "Order.findRejectedOrders", query = "SELECT o FROM Order o WHERE o.status=0 and o.consumer=?1")

public class Order implements Serializable{
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name="creation_timestamp")
    private Date timestamp;
    private int total;
    @Temporal(TemporalType.DATE)
    @Column(name="start_date")
    private Date startDate;
    private int status;
    @ManyToOne
    @JoinColumn(name="consumer_id")
    private Consumer consumer;
    @ManyToOne
    @JoinColumn(name = "vp_id")
    private ValidityPeriod validityPeriod;
    @ManyToOne
    @JoinColumn(name = "sp_id")
    private ServicePackage servicePackage;
    @OneToMany(fetch= FetchType.EAGER, mappedBy="order", cascade= CascadeType.ALL)
    private List<OrderOptionalProducts> orderOptProds;
```

Entity OrderOptionalProducts

```java
@Entity
@Table(name = "order_optprod", schema = "db_test")

public class OrderOptionalProducts implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @ManyToOne
    @JoinColumn(name="order_id")
    private Order order;

    @ManyToOne
    @JoinColumn(name="optprod_id")
    private OptionalProduct optionalProduct;

    @Column(name="order_status")
    private int status;
```

Entity Audit

```
@Entity
@Table(name = "audit", schema = "db_test")
@NamedQuery(name = "Audit.findFailedOrder",
            query = "SELECT a FROM Audit a WHERE a.consumer = ?1 AND a.order = ?2" )

public class Audit implements Serializable{
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name="rejection_timestamp")
    private Date rejectionTimestamp;
    @Column (name = "amount")
    private int totalPrice;
    @OneToOne
    @JoinColumn(name="order_id")
    private Order order;
    @OneToOne
    @JoinColumn(name="consumer_id")
    private Consumer consumer;
```

Entity Service Activation Schedule

```java
@Entity
@Table(name = "service_activation_schedule", schema = "db_test")
public class ServiceActivationSchedule implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @OneToOne
    @JoinColumn(name="order_id")
    private Order order;

    @Temporal(TemporalType.DATE)
    @Column(name = "activation_date")
    private Date activation;

    @Temporal(TemporalType.DATE)
    @Column(name = "deactivation_date")
    private Date deactivation;

    @ManyToMany
    @JoinTable(name="schedule_service", joinColumns = @JoinColumn(name = "schedule_id"),
            inverseJoinColumns = @JoinColumn(name = "service_id"))
    private List<Service> services;

    @ManyToMany
    @JoinTable(name="schedule_optprod", joinColumns = @JoinColumn(name = "schedule_id"),
            inverseJoinColumns = @JoinColumn(name = "optprod_id"))
    private List<OptionalProduct> optionalProducts;
```

Entity Employee

```
@Entity
@Table(name = "employee", schema = "db_test")
@NamedQuery(name = "Employee.checkCredentials",
            query = "SELECT r FROM Employee r  WHERE r.username = ?1 and r.password = ?2")

public class Employee implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String username;
    private String password;
```

Entity OptionalProductPurchases

```java
@Entity
@Table(name = "op_purchases", schema = "db_test")
@NamedQuery(name = "OptionalProductPurchases.getMax",
            query = "SELECT opp FROM OptionalProductPurchases opp WHERE opp.opSales= (SELECT MAX(opp1.opSales) FROM OptionalProductPurchases opp1)")
public class OptionalProductPurchases implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="op_id")
    private int id;

    @OneToOne
    @PrimaryKeyJoinColumn(name ="op_id")
    private OptionalProduct optionalProduct;
    @Column(name="op_sales")
    private int opSales;
```

Entity ServicePackagePurchases

```java
@Entity
@Table(name = "package_purchases", schema = "db_test")
@NamedQuery(name = "ServicePackagePurchases.getAll", query = "SELECT spp FROM ServicePackagePurchases spp")
public class ServicePackagePurchases implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "sp_id")
    private int sp_id;

    @OneToOne
    @PrimaryKeyJoinColumn(name ="sp_id")
    private ServicePackage pack;
    @Column(name="total_purchases")
    private int totalPurchases;
    @Column(name="total_sales_op")
    private int totalSalesOp;
    @Column(name="total_sales_not_op")
    private int totalSalesNotOp;
    @Column(name="average_op")
    private float averageOp;
```

Entity ServicePackageValidityPeriodPurchases

```java
@Entity
@Table(name = "package_purchases", schema = "db_test")
@NamedQuery(name = "ServicePackagePurchases.getAll", query = "SELECT spp FROM ServicePackagePurchases spp")
public class ServicePackagePurchases implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "sp_id")
    private int sp_id;

    @OneToOne
    @PrimaryKeyJoinColumn(name ="sp_id")
    private ServicePackage pack;
    @Column(name="total_purchases")
    private int totalPurchases;
    @Column(name="total_sales_op")
    private int totalSalesOp;
    @Column(name="total_sales_not_op")
    private int totalSalesNotOp;
    @Column(name="average_op")
    private float averageOp;
```

Functional analysis of the interaction

We have used the textual notation, where the pages are underlined in red, the view components in green, the actions in brown and the events in blue.

Textual notation

TELCO SERVICE APPLICATIONS

A telco company offers pre-paid online services to web users. Two client applications using the same database need to be developed.

CONSUMER APPLICATION

The consumer application has a public Landing page with a form for login and a form for registration. Registration requires a username, a password and an email. Login leads to the Home page of the consumer application. Registration leads back to the landing page where the user can log in.

The user can log in before browsing the application or browse it without logging in. If the user has logged in, his/her username appears in the top right corner of all the application pages.

The Home page of the consumer application displays the service packages offered by the telco company.

A service package has an ID and a name (e.g., “Basic”, “Family”, “Business”, “All Inclusive”, etc). It comprises one or more services. Services are of four types: fixed phone, mobile phone, fixed internet, and mobile internet. The mobile phone service specifies the number of minutes and SMSs included in the package plus the fee for extra minutes and the fee for extra SMSs. The mobile and fixed internet services specify the number of Gigabytes included in the package and the fee for extra Gigabytes. A service package must be associated with one validity period. A validity period specifies the number of months (12, 24, or 36). Each validity period has a different monthly fee (e.g., 20€/month for 12 months, 18€/month for 24 months, and 15€ /month for 36 months). A package may be associated with one or more optional products (e.g., an SMS news feed, an internet TV channel, etc.). The validity period of an optional product is the same as the validity period that the user has chosen for the service package. An optional product has a name and a monthly fee independent of the validity period duration. The same optional product can be offered in different service packages.

Textual notation

From the Home page, the user can access a Buy Service page for purchasing a service package and thus creating a service subscription. The Buy Service page contains a form for purchasing a service package. The form allows the user to select one package from the list of available ones and choose the validity period duration and the optional products to buy together with the chosen service. The form also allows the user to select the start date of his/her subscription. After choosing the service packages, the validity period and (0 or more) optional products, the user can press a CONFIRM button. The application displays a CONFIRMATION page that summarizes the details of the chosen service package, the validity period, the optional products and the total price to be pre-paid: (monthly fee of service package * number of months) + (sum of monthly fees of options * number of months).

If the user has already logged in, the CONFIRMATION page displays a BUY button. If the user has not logged in, the CONFIRMATION page displays a link to the login page and a link to the REGISTRATION page. After either logging in or registering and immediately logging in, the CONFIRMATION page is redisplayed with all the confirmed details and the BUY button.

When the user presses the BUY button, an order is created. The order has an ID and a date and hour of creation. It is associated with the user and with the service package, its validity period and the chosen optional products. It also contains the total value (as in the CONFIRMATION page) and the start date of the subscription. After creating the order, the application bills the customer by calling an external service. If the external service accepts the billing, the order is marked as valid and a service activation schedule is created for the user. A service activation schedule is a record of the services and optional products to activate for the user with their date of activation and date of deactivation.

If the external service rejects the billing, the order is put in the rejected status and the user is flagged as insolvent. When an insolvent user logs in, the home page also contains the list of rejected orders. The user can select one of such orders, access the CONFIRMATION page, press the BUY button and attempt the payment again. When the same user causes three failed payments, an alert is created in a dedicated auditing table, with the user Id, username, email, and the amount, date and time of the last rejection.

Textual notation

EMPLOYEE APPLICATION

The employee application allows the authorized employees of the telco company to log in. In the Home page, a form allows the creation of service packages, with all the needed data and the possible optional products associated with them. The same page lets the employee create optional products as well.

A Sales Report page allows the employee to inspect the essential data about the sales and about the users over the entire lifespan of the application:

- Number of total purchases per package.
- Number of total purchases per package and validity period.
- Total value of sales per package with and without the optional products.
- Average number of optional products sold together with each service package.
- List of insolvent users, suspended orders and alerts.
- Best seller optional product, i.e. the optional product with the greatest value of sales across all the sold service packages.

Components

- Consumer Application Components
  - CheckLogin: checks the login
  - Registration: checks the registration
  - Logout: close the session of the user
  - Home: shows the list of service packages
  - Buy: shows the available validity periods and optional products and the activation date (to be selected)
  - Confirmation: shows order of the user and a button ‘buy’ if the user is logged, otherwise shows ‘login’
  - Payment: randomly decides if a payment is successful or not
  - index.html : there are two forms one for registration and one for login
  - home.html : the service package can be selected
  - buypage.html : the validty period, optional product and the activation date of a service package can be selected
  - confirmationpage.html : shows a button ‘buy’ if the user is logged otherwise ‘login’

Components

- Consumer Application Back End Components
  - Entity
    - Consumer
    - Service (omitted FixedPhone,FixedInternet,MobilePhone,MobileInternet)
    - Service Package
    - Order
    - OptionalProduct
    - ValidityPeriod
    - OrderOptionalProduct
    - Audit
    - ServiceActivationSchedule

Components

- Consumer Application Back End Components
  - EJB services
    - AuditService
      - insertFailedOrder(Consumer consumer, Order order)
      - updateFailedOrder(Consumer consumer, Order order)
    - ConsumerService
      - checkCredentials(String username, String password, String email)
      - updateConsumer(Consumer consumer)
      - registerConsumer(String username, String password, String email)
    - OrderService
      - findRejectedOrdersByConsumer(Consumer consumer)
      - findOrder(int id)
      - insertOrder(Order order)
      - updateOrder(int id, int status)

Components

- Consumer Application Back End Components
  - EJB services
    - ServicePackageService
      - findAllServicePackages()
      - findServicePackage(int id)
    - ScheduleService
      - insertSchedule(ServiceActivationSchedule schedule)

Components

- Employee Application Components
  - CheckLogin: checks the login
  - CreateOptProd: creates an optional product
  - CreateServicePack: creates a service package
  - Home: sets the lists of validity periods, optional products and services already present
  - Logout: close the session
  - SalesReport: sets the insolvent consumers, the suspended orders,the package purchases and the optional products
  - Index.html: one form for a login
  - Home.html: shows the lists of the attributes setted in the Home
  - SalesReport.html: shows the lists of attributes setted in SalesReport

Components

- Employee Application Back End Components
  - Entity
    - Employee
    - Consumer
    - Service (omitted FixedPhone,FixedInternet,MobilePhone,MobileInternet)
    - Service Package
    - Order
    - OptionalProduct
    - ValidityPeriod
    - OrderOptionalProduct
    - Audit
    - ServicePackagePurchases
    - ServicePackageValidityPeriodPurchases
    - OptionalProductPurchases

Components

- Employee Application Back End Components
  - EJB services
    - EmployeeService
      - checkCredentials(String username, String password)
    - ServiceService
      - findAllServices()
      - findAllById(String[] sIds)
    - AuditService
      - getAlerts()
    - ConsumerService
      - getInsolvent()
    - OptionalProductService
      - findAllOptionalProducts()
      - findAllById(String[] optProdIds)
      - insertOptionalProduct(OptionalProduct op)

Components

- Employee Application Back End Components
  - EJB services
    - ValidityPeriodService
      - findAllValidityPeriod()
      - findAllById(String[] vpIds)
    - OrderService
      - findRejectedOrders (Consumer consumer)
    - ServicePackageService
      - insertServicePackage(ServicePackage sp)

Components

- Employee Application Back End Components
  - EJB services
    - ServicePackageValidityPeriodPurchases
      - getAll()
    - ServicePackagePurchasesService
      - getAll()
    - OptionalProductPurchases
      - getMax()

Motivations of the components design

- We split the project in two: one is the Consumer Application with its own EJB and WEB project, the other one is the Employee Application with its own EJB and WEB project. Obviously some entities are the same, even if the methods in the entities service are different.

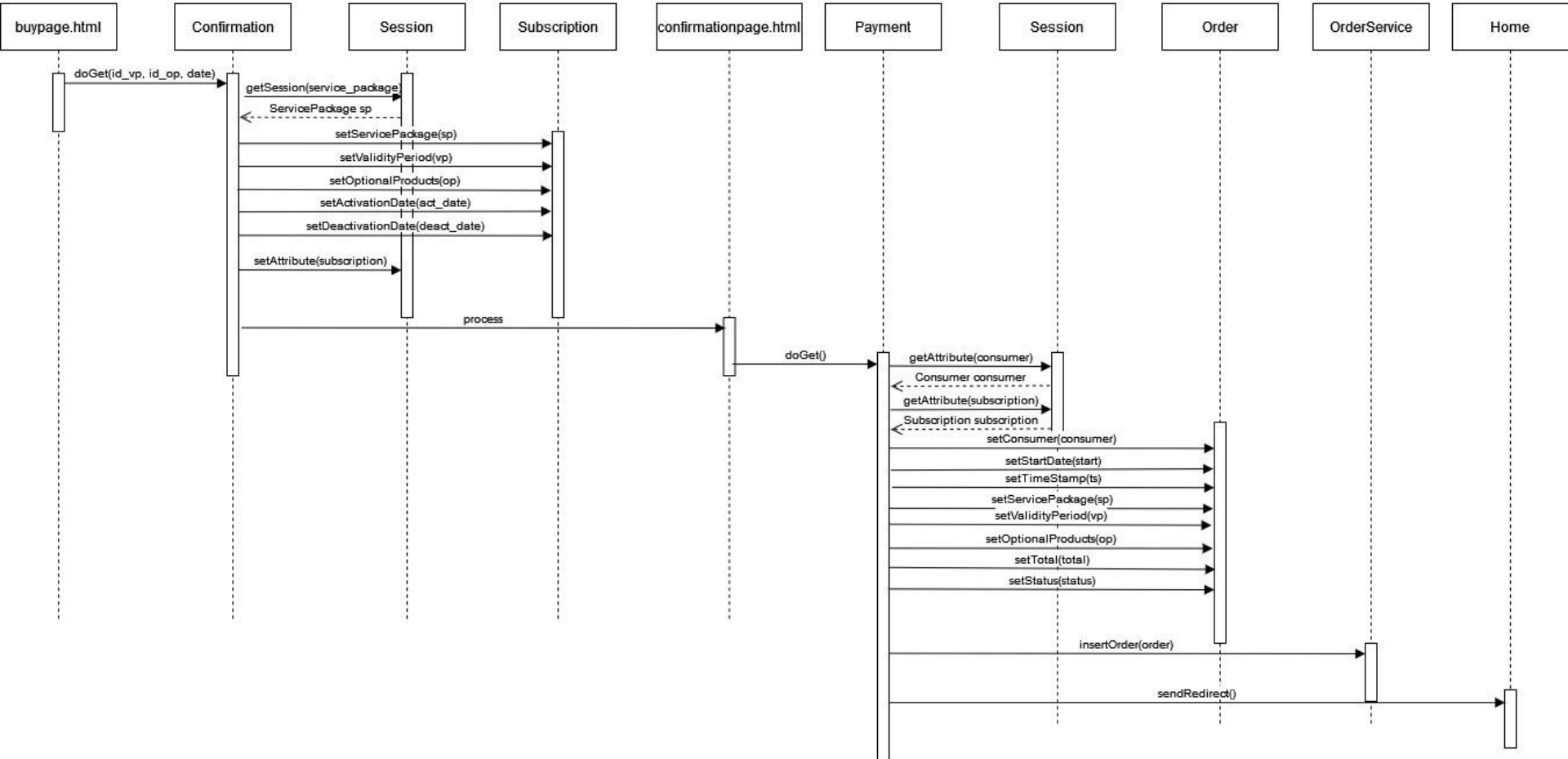
UML sequence diagrams

Observations

- We do not explain the login and registration event, because we have considered more relevant other events
- In the event ‘ creation order’ we start from the buypage and omit the home where the user selects the service package
- In the event ‘retry a failed order’ we omit the part of the three times failed order to simplify the design, but when this happens we insert the consumer in the audit table

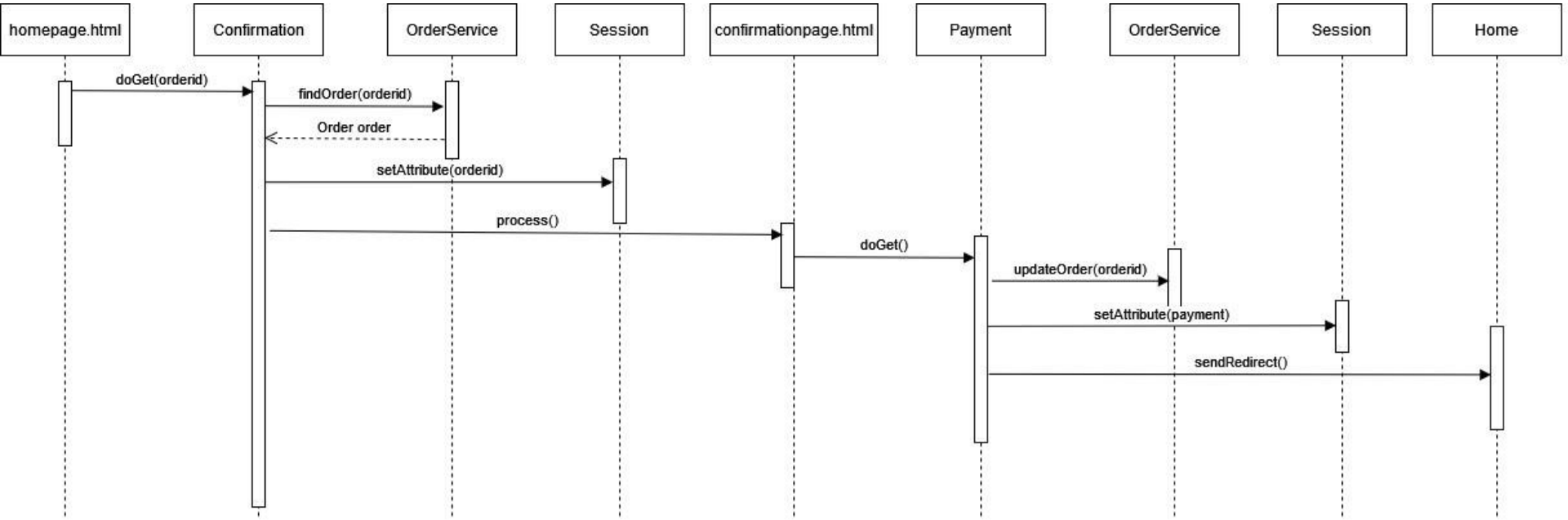
Consumer Application

Event: creation of an order



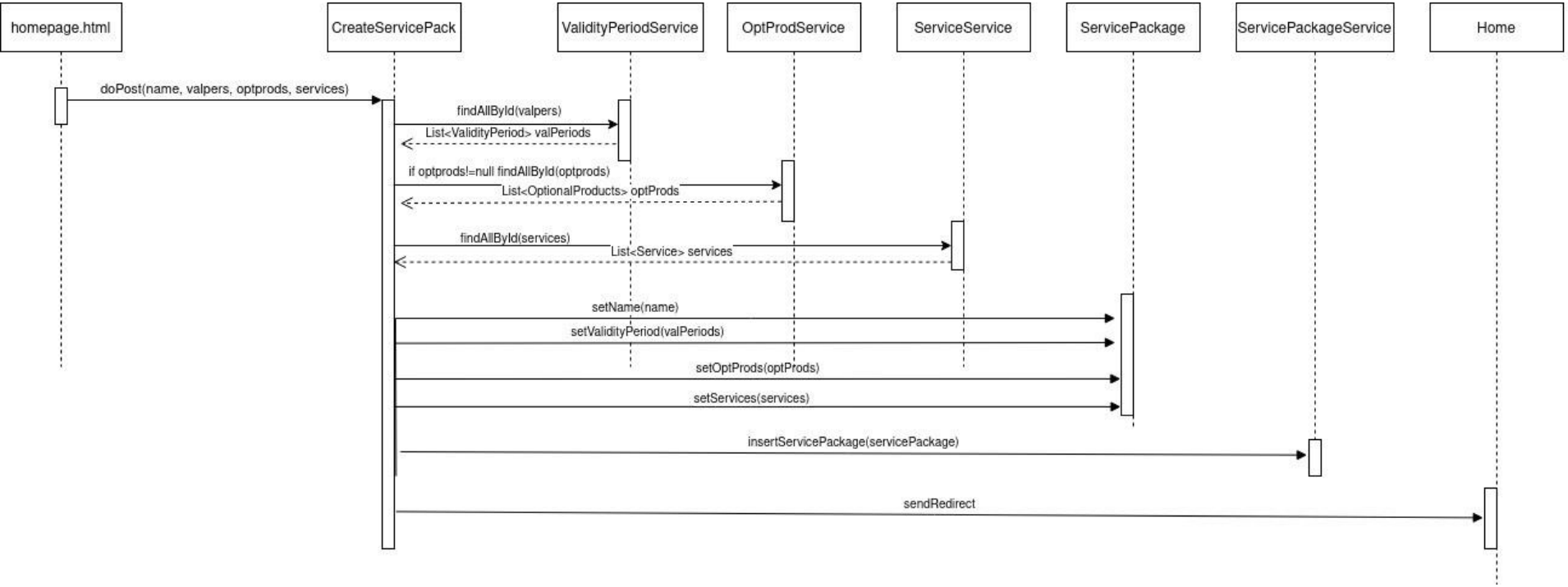
Consumer Application

Event: retry to pay a failed order



Employee Application

Event: creation of a service package



Employee Application

Event: view of sales reports page