ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

Τεχνολογία Διοίκησης Επιχειρησιακών Διαδικασιών (ΠΜΣ 541)

Χειμερινό Εξάμηνο 2014 – 2015

Ανάπτυξη υπηρεσιοστρεφών επιχειρησιακών διαδικασιών ηλεκτρονικού καταστήματος

Χαράλαμπος Παπαδόπουλος – Μ1353

labisp@di.uoa.gr

Γεώργιος Σταμούλης – Μ1224

gstam@di.uoa.gr

Αθήνα, Σεπτέμβριος 2015

Πίνακας Περιεχομένων

1. Εισαγωγή

Σκοπός της παρούσης εργασίας είναι η υπηρεσιοστρεφής ανάπτυξη επιχειρησιακών διαδικασιών που θα υποστηρίζουν την απαιτούμενη λειτουργικότητα του ηλεκτρονικού καταστήματος μιας αλυσίδας πώλησης ηλεκτρονικών ειδών.

Η συγκεκριμένη αλυσίδα καταστημάτων έχει σαν στόχο:

- να επαναχρησιμοποιήσει μια σειρά υπηρεσιών και λειτουργιών που υποστηρίζουν την λειτουργία των τυπικών λιανικών καταστημάτων τους. Παρακάτω ακολουθεί περιγραφή αυτών των υπηρεσιών και λειτουργιών καθώς και των βοηθητικών κλάσεων τους.
- να υλοποιήσει επιπλέον υπηρεσίες πάνω από τις ήδη υπάρχουσες για την υποστήριξη του ηλεκτρονικού καταστήματος. Επίσης ακολουθεί περιγραφή των επιπλέων υπηρεσιών.

2. Περιγραφή υπηρεσιών και λειτουργιών

1) **Κατάλογος προϊόντων του καταστήματο**ς: Για κάθε προϊόν η συγκεκριμένη υπηρεσία επιστρέφει πληροφορία, όπως:
    - τον κωδικό του,
    - το όνομα του,
    - την κατηγορία του,
    - την τιμή του,
    - την τεχνική περιγραφή του,
    - την διαθέσιμοτητα του,
    - το κατάστημα ή την αποθήκη στην οποία είναι πιθανά διαθέσιμο

2) **Καλάθι αγορών:** Για κάθε πελάτη του καταστήματος διατηρεί και επιστρέφει πληροφορία σχετικά με:
    - τα αντικείμενα που έχει επιλέξει ο πελάτης,
    - την ποσότητα του κάθε αντικειμένου,
    - τον τρόπο πληρωμής της παραγγελίας του πελάτη και
    - την επιλεγμένη διεύθυνση αποστολής

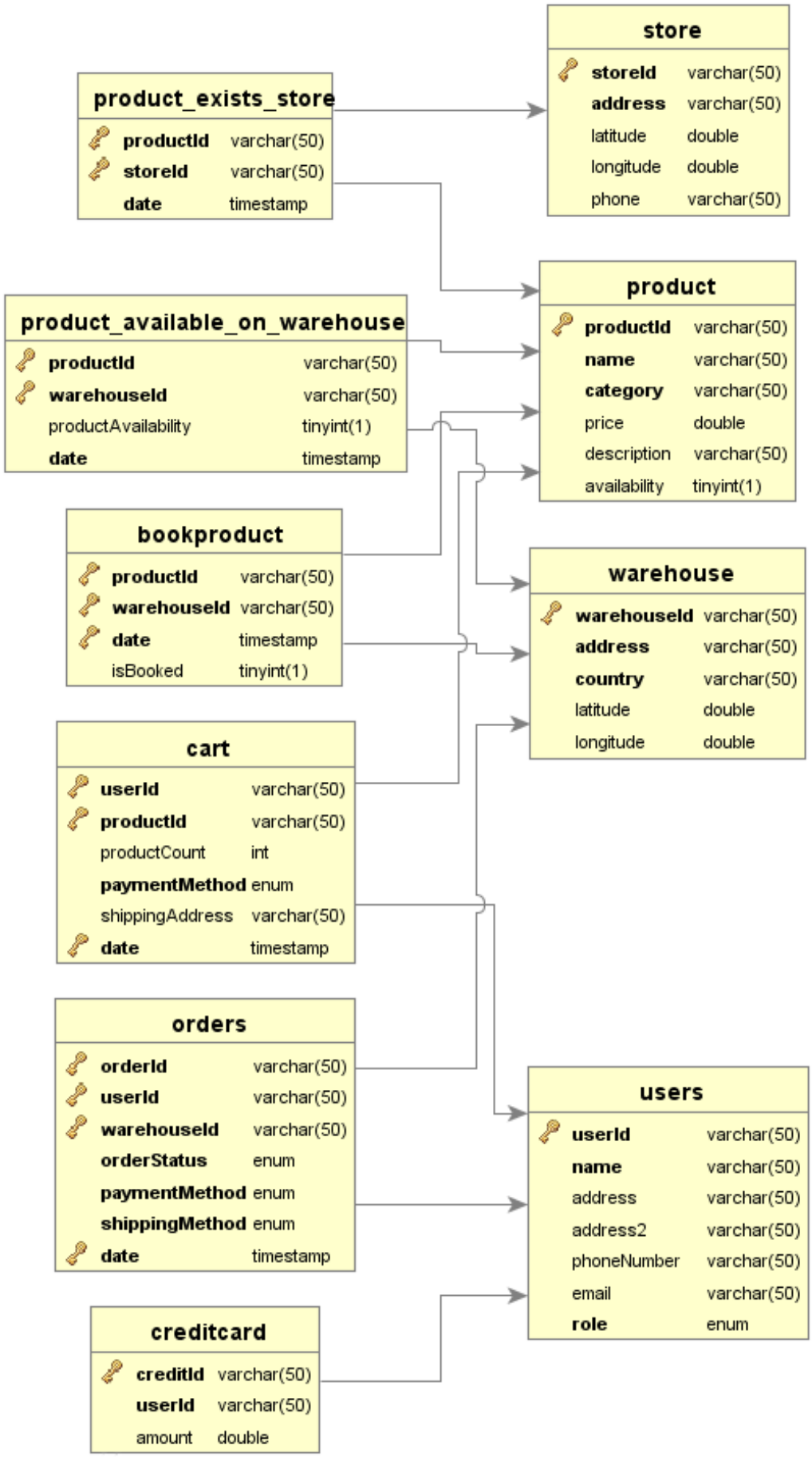
3) **Διαχείριση πελατών:** Για κάθε πελάτη του καταστήματος διατηρεί πληροφορία σχετικά με:
    - τις αποθηκευμένες πιστωτικές κάρτες,
    - τις αποθηκευμένες διευθύνσεις αποστολής παραγγελιών,
    - το ιστορικό παραγγελιών

4) **Διαχείριση αποθήκης:** Το κατάστημα διαθέτει μια λίστα από έξι αποθήκες που είναι τοποθετημένες σε διάφορες περιοχές (σε επίπεδο πόλης ή/και χώρας). Οι λειτουργίες που προσφέρονται από την κάθε αποθήκη περιλαμβάνουν:
    - Τον έλεγχο διαθεσιμότητας κάθε αντικειμένου,
    - Τη κράτηση κάποιου αντικειμένου για λογαριασμό μιας παραγγελίας,
    - Την ακύρωση μιας κράτησης για κάποιο αντικείμενο

5) **Υπηρεσία πίστωσης:** Η υπηρεσία αυτή προσφέρεται από το συνεργαζόμενο πιστωτικό ίδρυμα με ασύγχρονο τρόπο. Έχει επιλεγεί το συγκεκριμένο μοντέλο επικοινωνίας από το πιστωτικό ίδρυμα για να μπορέσουν να πραγματοποιηθούν όλοι οι απαιτούμενοι έλεγχοι χωρίς να μπλοκάρει η λειτουργία του πελάτη της υπηρεσίας. Η υπηρεσία προσφέρει λειτουργίες που επιτρέπουν:
    - τη χρέωση της πιστωτικής κάρτας του πελάτη με το ποσό της κάθε αγοράς και τη μεταφορά των σχετικών πόρων στο λογαριασμό του εμπόρου,
    - την επιστροφή των χρημάτων του πελάτη σε περίπτωση ακύρωσης.

Βοηθητικές Κλάσεις

Για την αποθήκευση των δεδομένων σχεδιάστηκε και υλοποιήθηκε MySQL βάση δεδομένων της οποίας το σχήμα περιγράφεται παρακάτω:


store
storeId varchar(50)
address varchar(50)
latitude double
longitude double
phone varchar(50)
product_exists_store
productId varchar(50)
storeId varchar(50)
date timestamp
product
productId varchar(50)
name varchar(50)
category varchar(50)
price double
description varchar(50)
availability tinyint(1)
product_available_on_warehouse
productId varchar(50)
warehouseId varchar(50)
productAvailability tinyint(1)
date timestamp
bookproduct
productId varchar(50)
warehouseId varchar(50)
date timestamp
isBooked tinyint(1)
warehouse
warehouseId varchar(50)
address varchar(50)
country varchar(50)
latitude double
longitude double
cart
userId varchar(50)
productId varchar(50)
productCount int
paymentMethod enum
shippingAddress varchar(50)
date timestamp
orders
orderId varchar(50)
userId varchar(50)
warehouseId varchar(50)
orderStatus enum
paymentMethod enum
shippingMethod enum
date timestamp
users
userId varchar(50)
name varchar(50)
address varchar(50)
address2 varchar(50)
phoneNumber varchar(50)
email varchar(50)
role enum
creditcard
creditId varchar(50)
userId varchar(50)
amount double

Στο παραπάνω σχήμα υπάρχουν οι παρακάτω οντότητες:

- **Users:** Ένας χρήστης (User) έχει userId, name, address, address2, phoneNumber, email και role
- **CreditCard:** Μια πιστωτική κάρτα (CreditCard) έχει creditId, userId και amount
- **Product:** Ένα προϊόν (Product) έχει productId, name, category, price, description και availability
- **Store:** Ένα κατάστημα (Store) έχει storeId, address, latitude, longitude και phone
- **WareHouse:** Μια αποθήκη (WareHouse) έχει warehouseId, address, country, latitude και longitude

Οι αντίστοιχες συσχετίσεις τους είναι:

- **Cart:** Ένας χρήστης κάνει προσθήκη στο καλάθι (Cart) ένα ή περισσότερα προϊόντα και κάθε προϊόν προστίθεται από ένα ή περισσότερους χρήστες. Το καλάθι έχει: userId, productId, productCount, paymentMethod, shippingAddress και date.
- **product_exists_store:** Ένα προϊόν υπάρχει σε ένα ή περισσότερα καταστήματα και κάθε κατάστημα έχει ένα ή περισσότερα ίδιου τύπου προϊόντα. Το product_exists_store έχει productId, storeId και date.
- **product_available_on_warehouse:** Ένα προϊόν είναι διαθέσιμο σε ένα ή περισσότερες αποθήκες αλλά και κάθε αποθήκη έχει διαθέσιμο ένα ή περισσότερα προϊόντα ίδιου τύπου. Το product_available_on_warehouse έχει productId, warehouseId, productAvailability και date.
- **bookProduct:** Σ’ ένα προϊόν γίνεται κράτηση σε μια ή περισσότερες αποθήκες αλλά και κάθε αποθήκη κάνει κράτηση σε ένα ή περισσότερα προϊόντα. Το bookProduct έχει productId, warehouseId, date και isBooked.
- **Orders:** Ένας χρήστης κάνει κράτηση κάποιου προϊόντος σε μια ή περισσότερες αποθήκες και κάθε αποθήκη μπορεί να κάνει κράτηση

για ένα ή περισσότερους χρήστες. Η συσχέτιση Orders έχει orderId, userId, warehouseId, orderStatus, paymentMethod, shippingMethod, date.

Για την διαχείριση της βάσης δεδομένων υλοποιήθηκε η παρακάτω κλάση:

```
public final class DatabaseManager {
    ...
    private static final Logger LOGGER = Logger
                .getLogger(DatabaseManager.class.getName());
    private static DatabaseManager databaseManager;
    private Connection connection;

    public static synchronized DatabaseManager getInstance()
                                                throws IOException {
        if (databaseManager == null) {
            final Properties properties = new Properties();
            properties.load(new FileInputStream(
                                        "databaseConfig.properties"));
            return new DatabaseManager(
                                properties.getProperty("jdbcDriver"),
                                properties.getProperty("jdbcUrl"));
        }
        return databaseManager;
    }

    private DatabaseManager(final String jdbcDriver,
                                                final String jdbcUrl) {
        try {
            Class.forName(jdbcDriver);
            connection = DriverManager.getConnection(jdbcUrl);
        } catch (final SQLException ex) {
            LOGGER.log(Level.SEVERE, null, ex);
        } catch (final ClassNotFoundException ex) {
            LOGGER.log(Level.SEVERE, null, ex);
        }
    }

    ...

    @Override
    public Object clone() throws CloneNotSupportedException {
        throw new CloneNotSupportedException(
                        "DatabaseManager does not support clone");
    }


    @Override
    protected void finalize() {
        try {
            connection.close();
            connection = null;
        } catch (final SQLException ex) {
            LOGGER.log(Level.SEVERE,
                        "Database connection do not closed", ex);
        }
    }
}
```

Το αντικείμενο DatabaseManager είναι ένα singleton αντικείμενο το οποίο δημιουργείται την πρώτη φορά που θα κληθεί από κάποιο service, όπως θα περιγράφει στη συνέχεια, και διαβάζει παραμέτρους σύνδεσης με την βάση δεδομένων από το databaseConfig.properties αρχείο.

Το DatabaseManager αντικείμενο επειδή είναι το μοναδικό αντικείμενο που διαχειρίζεται την βάση δεδομένων, δεν επιτρέπεται να γίνει cloned, γι' αυτό γίνεται Override η μέθοδος clone() έχοντας ως υλοποίηση την δημιουργία ενός CloneNotSupportedException().

Επίσης όταν επρόκειτο να γίνει finalize ο DatabaseManager, καλείται αυτόματα η μέθοδος finalize() και κλείνει το connection με την βάση δεδομένων.

Τέλος μέσα στην κλάση DatabaseManager έχουν υλοποιηθεί όλες οι απαραίτητες μέθοδοι οι οποίες είναι υπεύθυνες για την ανάκτηση πληροφορίας από την βάση δεδομένων και ενημέρωση αυτής.

Για την περιγραφή των οντοτήτων της βάσης δεδομένων χρησιμοποιήθηκαν οι παρακάτω κλάσεις (containers):

- Users.java

```
public class Users implements Serializable {
    private String userId;
    private String name;
    private String address;
    private String address2;
    private String phoneNumber;
    private String email;
    private Role role;
    private List<CreditCard> creditCart;
    private List<Product> orderHistory;

    public Users() {}

    public Users(final String userId, final String name, final String
address, final String address2, final String phoneNumber, final
String email, final Role role, final List<CreditCard> creditCart,
final List<String> savedAddresses, final List<Product> orderHistory){
        this.userId = userId;
        this.name = name;
        this.address = address;
        this.address2 = address2;
        this.phoneNumber = phoneNumber;
        this.email = email;
        this.role = role;
        this.creditCart = creditCart;
        this.orderHistory = orderHistory;
    }
```

```
    // Getters - Setters
}
```

Η παραπάνω κλάση αναπαριστά τους πελάτες (χρήστες) της εφαρμογής.

- Role.java

```
public enum Role {
    CUSTOMER,
    OWNER
}
```

Το παραπάνω enumeration αναπαριστά τους ρόλους των χρηστών της εφαρμογής.

- CreditCart.java

```
public class CreditCard implements Serializable {
    private String creditId;
    private String customer;
    private double amount;

    public CreditCard() {}

    public CreditCard(final String creditId, final String customer,
final double amount) {
        this.creditId = creditId;
        this.customer = customer;
        this.amount = amount;
    }
    // Getters - Setters
}
```

Η παραπάνω κλάση αναπαριστά την πιστωτική κάρτα ενός χρήστη.

- Product.java

```
public class Product implements Serializable {
    private String productId;
    private String name;
    private String category;
    private double price;
    private String description;
    private Boolean availability;

    public Product() {}

    public Product(final String productId, final String name, final
String category, final double price, final String description, final
Boolean availability) {
```

```
        this.productId = productId;
        this.name = name;
        this.category = category;
        this.price = price;
        this.description = description;
        this.availability = availability;
    }
    // Getters - Setters
}
```

Η παραπάνω κλάση αναπαριστά ένα προϊόν.

- Cart.java

```
public class Cart implements Serializable {
    private String userId;
    private List<String> productId;
    private Map<Product, Integer> productCount;
    private PaymentMethod paymentMethod;
    private String shippingAddress;

    public Cart() {}

    public Cart(final String userId, final List<String> productId,
final Map<Product, Integer> productCount, final PaymentMethod
paymentMethod, final String shippingAddress) {
        this.userId = userId;
        this.productId = productId;
        this.productCount = productCount;
        this.paymentMethod = paymentMethod;
        this.shippingAddress = shippingAddress;
    }
    // Getters - Setters
}
```

Η παραπάνω κλάση αναπαριστά το καλάθι ενός πελάτη.

- PaymentMethod.java

```
public enum PaymentMethod {
    CASH,
    BANK_PAY,
    CREDIT_CARD,
    PAYPAL
}
```

Το παραπάνω enumeration αναπαριστά τους τρόπους πληρωμής που μπορεί να χρησιμοποιήσει κάποιος πελάτης.

- Order.java

```
public class Order implements Serializable {
    private String orderId;
    private List<String> productId;
    private OrderStatus orderStatus;
    private PaymentMethod paymentMethod;
    private ShippingMethod shippingMethod;
    private String warehouseId;

    public Order() {}

    public Order(final String orderId, final List<String> productId,
final OrderStatus orderStatus,final PaymentMethod paymentMethod,
final ShippingMethod shippingMethod, final String warehouseId) {
        this.orderId = orderId;
        this.productId = productId;
        this.orderStatus = orderStatus;
        this.paymentMethod = paymentMethod;
        this.shippingMethod = shippingMethod;
        this.warehouseId = warehouseId;
    }
    // Getters - Setters
}
```

Η παραπάνω κλάση αναπαριστά την παραγγελία ενός πελάτη

- OrderStatus.java

```
public enum OrderStatus {
    SUBMITTED,
    PENDING,
    CANCELED
}
```

Το παραπάνω enumeration αναπαριστά το status το οποίο μπορεί να βρίσκεται μια παραγγελία.

- ShippingMethod.java

```
public enum ShippingMethod {
    COURIER,
    FROM_STORE
}
```

Το παραπάνω enumeration αναπαριστά την μέθοδο που μπορεί να επιλέξει κάποιος πελάτης για να παραλάβει κάποιο προϊόν.

- Store.java

```
public class Store implements Serializable {
    private String storeId;
    private String address;
    private double latitude;
    private double lognitude;
    private String phone;
    private List<String> products;

    public Store() {}

    public Store(final String storeId, final String address, final
double latitude, final double lognitude, final String phone, final
List<String> products) {
        this.storeId = storeId;
        this.address = address;
        this.latitude = latitude;
        this.lognitude = lognitude;
        this.phone = phone;
        this.products = products;
    }
    // Getters - Setters
}
```

Η παραπάνω κλάση αναπαριστά το κατάστημα.

- WareHouse.java

```
public class WareHouse implements Serializable {
    private String warehouseId;
    private String address;
    private String country;
    private double latitude;
    private double lognitude;
    private Map<String, Boolean> productAvailability;
    private Map<Integer, List<Date>> bookingProduct;
    private Map<Integer, List<Date>> cancelBooking;
    private List<String> products;

    public WareHouse() {}

    public WareHouse(final String warehouseId, final String address,
final String country,final double latitude, final double lognitude,
final Map<String, Boolean> productAvailability, final Map<Integer,
List<Date>> bookingProduct, final Map<Integer, List<Date>>
cancelBooking, final List<String> products) {
        this.warehouseId = warehouseId;
        this.address = address;
        this.country = country;
        this.latitude = latitude;
        this.lognitude = lognitude;
        this.productAvailability = productAvailability;
        this.bookingProduct = bookingProduct;
        this.cancelBooking = cancelBooking;
        this.products = products;
    }
```

```
    // Getters - Setters
}
```

Η παραπάνω κλάση αναπαριστά την αποθήκη.

Web Services

Τα web services που αναπτύχθηκαν αντιπροσωπεύουν τις πέντε υπηρεσίες που αναφέρθηκαν προηγουμένως (κλάσεις Service1-5.java) και για την υλοποίηση αυτών χρησιμοποιήθηκαν επίσης βοηθητικές κλάσεις (κλάσεις Service1-5Impl.java) οι οποίες καλούνται από αυτά. Παρακάτω περιγράφονται τα web services μαζί με τις αντίστοιχες βοηθητικές κλάσεις τους:

- Service1.java

```
@WebService()
public class Service1 {

    @WebMethod(operationName = "findProduct")
    public Product findProduct(@WebParam(name = "productId") String
productId) throws IOException {
        return new Service1Impl().findProduct(productId);
    }
}
```

Η παραπάνω κλάση περιγράφει το Service1 της εφαρμογής το οποίο έχει την μέθοδο: findProduct().

- Service1Impl.java

```
public class Service1Impl {

    private DatabaseManager databaseManager;

    public Service1Impl() throws IOException {
        databaseManager = DatabaseManager.getInstance();
    }

    public Product findProduct(String productId) {
        return databaseManager.findProduct(productId);
    }
}
```

Η παραπάνω κλάση περιγράφει την υλοποίηση του Service1 η οποία καλείται από το Service1. Χρησιμοποιείται το singleton αντικείμενο DatabaseManager,

το οποίο εκτός από το να διαχειρίζεται την βάση δεδομένων, υλοποιεί και όλες τις μεθόδους που καλούνται από τις υλοποιήσεις των Services. Έτσι στην Service1Impl κλάση καλείται η μέθοδος findProduct() για να επιστραφεί το αντικείμενο Product που αντιστοιχεί σε συγκεκριμένο productId.

- Service2.java

```
@WebService()
public class Service2 {

    @WebMethod(operationName = "findCart")
    public Cart findCart(@WebParam(name = "orderId") String orderId)
throws IOException {
        return new Service2Impl().findCart(orderId);
    }
}
```

Η παραπάνω κλάση περιγράφει το Service2 της εφαρμογής το οποίο έχει την μέθοδο: findCart().

- Service2Impl.java

```
public class Service2Impl {

    private DatabaseManager databaseManager;

    public Service2Impl() throws IOException {
        databaseManager = DatabaseManager.getInstance();
    }

    public Cart findCart(String orderId) {
        return databaseManager.findCart(orderId);
    }
}
```

Η παραπάνω κλάση περιγράφει την υλοποίηση του Service2 όπου με χρήση του DatabaseManager καλείται η findCart() και επιστρέφει το καλάθι του πελάτη.

- Service3.java

```
@WebService()
public class Service3 {

    @WebMethod(operationName = "getCreditCards")
    public List<CreditCard> getCreditCards(@WebParam(name = "userId")
String userId) throws IOException {
        return new Service3Impl().getCreditCards(userId);
    }

    @WebMethod(operationName = "getAddresses")
    public List<String> getAddresses(@WebParam(name = "userId")
String userId) throws IOException {
        return new Service3Impl().getAddresses(userId);
    }

    @WebMethod(operationName = "getOrderHistory")
    public List<Order> getOrderHistory(@WebParam(name = "userId")
String userId) throws IOException {
        return new Service3Impl().getOrderHistory(userId);
    }
}
```

Η παραπάνω κλάση περιγράφει το Service3 το οποίο έχει τις μεθόδους: getCreditCards(), getAddresses() και getOrderHistory().

- Service3Impl.java

```
public class Service3Impl {

    private DatabaseManager databaseManager;

    public Service3Impl() throws IOException {
        databaseManager = DatabaseManager.getInstance();
    }

    public List<CreditCard> getCreditCards(String userId) {
        return databaseManager.getCreditCards(userId);
    }

    public List<String> getAddresses(String userId) {
        return databaseManager.getAddresses(userId);
    }

    public List<Order> getOrderHistory(String userId) {
        return databaseManager.getOrderHistory(userId);
    }
}
```

Η παραπάνω κλάση περιγράφει την υλοποίηση του Service3 όπου μέσω του αντικειμένου DatabaseManager καλείται η μέθοδος getCreditCards() για να επιστρέψει μια λίστα από CreditCards για συγκεκριμένο userId. Επίσης

καλείται η μέθοδος getAddresses() για να επιστρέψει λίστα από addresses ενός χρήστη με συγκεκριμένο userId. Τέλος καλείται η μέθοδος getOrderHistory() για να επιστρέψει λίστα από Orders ενός χρήστη με συγκεκριμένο userId.

- Service4.java

```
@WebService()
public class Service4 {

    @WebMethod(operationName = "isProductAvailable")
    public boolean isProductAvailable(@WebParam(name = "productId")
String productId) throws IOException {
        return new Service4Impl().isProductAvailable(productId);
    }

    @WebMethod(operationName = "bookProduct")
    public boolean bookProduct(@WebParam(name = "productId") String
productId) throws IOException {
        return new Service4Impl().bookProduct(productId);
    }

    @WebMethod(operationName = "cancelBookProduct")
    public boolean cancelBookProduct(@WebParam(name = "productId")
String productId) throws IOException {
        return new Service4Impl().cancelBookProduct(productId);
    }
}
```

Η παραπάνω κλάση περιγράφει το Service4 το οποίο έχει τις μεθόδους: isProductAvailable(), bookProduct() και cancelBookProduct().

- Service4Impl.java

```
public class Service4Impl {

    private DatabaseManager databaseManager;

    public Service4Impl() throws IOException {
        databaseManager = DatabaseManager.getInstance();
    }

    public boolean isProductAvailable(String productId) {
        return databaseManager.isProductAvailable(productId);
    }

    public boolean bookProduct(String productId) {
        return databaseManager.bookProduct(productId);
    }

    public boolean cancelBookProduct(String productId) {
        return databaseManager.cancelBookProduct(productId);
```

```
    }
}
```

Η παραπάνω κλάση περιγράφει την υλοποίηση του Service4 όπου μέσω του αντικειμένου DatabaseManager καλείται η μέθοδος isProductAvailable() για να επιστρέψει true ή false ανάλογα με το αν είναι διαθέσιμο κάποιο product με συγκεκριμένο productId. Επίσης καλείται η μέθοδος bookProduct() για να κάνει κράτηση σ' ένα συγκεκριμένο product με κάποιο productId και επιστρέφει true ή false ανάλογα αν ολοκληρώθηκε με επιτυχία ή όχι η κράτηση. Τέλος καλείται η μέθοδος cancelBookProduct() για να ακυρώσει την κράτηση κάποιου προϊόντος με συγκεκριμένο productId και επιστρέφει true ή false ανάλογα με το αν έγινε με επιτυχία ή όχι η ακύρωση.

- Service5.java

```
@WebService()
public class Service5 {

    @WebMethod(operationName = "cartCost")
    public double cartCost(@WebParam(name = "orderId") String
orderId) throws IOException {
        return new Service5Impl().cartCost(orderId);
    }

    @WebMethod(operationName = "chargeCard")
    public boolean chargeCard(@WebParam(name = "creditId") String
creditId, @WebParam(name = "cost") double cost) throws IOException {
        return new Service5Impl().chargeCard(creditId, cost);
    }

    @WebMethod(operationName = "refund")
    public boolean refund(@WebParam(name = "creditId") String
creditId, @WebParam(name = "cost") double cost) throws IOException {
        return new Service5Impl().refund(creditId, cost);
    }
}
```

Η παραπάνω κλάση περιγράφει το Service5 το οποίο έχει τις μεθόδους: cartCost(), chargeCard() και refund().

- Service5Impl.java

```
public class Service5Impl {

    private DatabaseManager databaseManager;

    public Service5Impl() throws IOException {
        databaseManager = DatabaseManager.getInstance();
    }

    public double cartCost(String orderId) {
        return databaseManager.cartCost(orderId);
    }

    public boolean chargeCard(String creditId, double cost) {
        return databaseManager.chargeCard(creditId, cost);
    }

    public boolean refund(String creditId, double cost) {
        return databaseManager.refund(creditId, cost);
    }
}
```

Η παραπάνω κλάση περιγράφει την υλοποίηση του Service5 όπου μέσω του αντικειμένου DatabaseManager καλείται η μέθοδος cartCost() για συγκεκριμένο orderId και επιστρέφει το κόστος του καλαθιού. Επίσης καλείται η μέθοδος chargeCard() για συγκεκριμένο creditId και έχοντας υπολογίσει το κόστος του καλαθιού από την προηγούμενη μέθοδο, επιστρέφει true ή false ανάλογα με το αν έγινε με επιτυχία ή όχι η χρέωση της πιστωτικής κάρτας του χρήστη. Τέλος καλείται η refund() με όμοια ορίσματα δηλαδή το creditId και το κόστος του καλαθιού και επιστρέφει true ή false ανάλογα με το αν έγινε με επιτυχία ή όχι η επιστροφή των χρημάτων στην πιστωτική κάρτα του χρήστη.

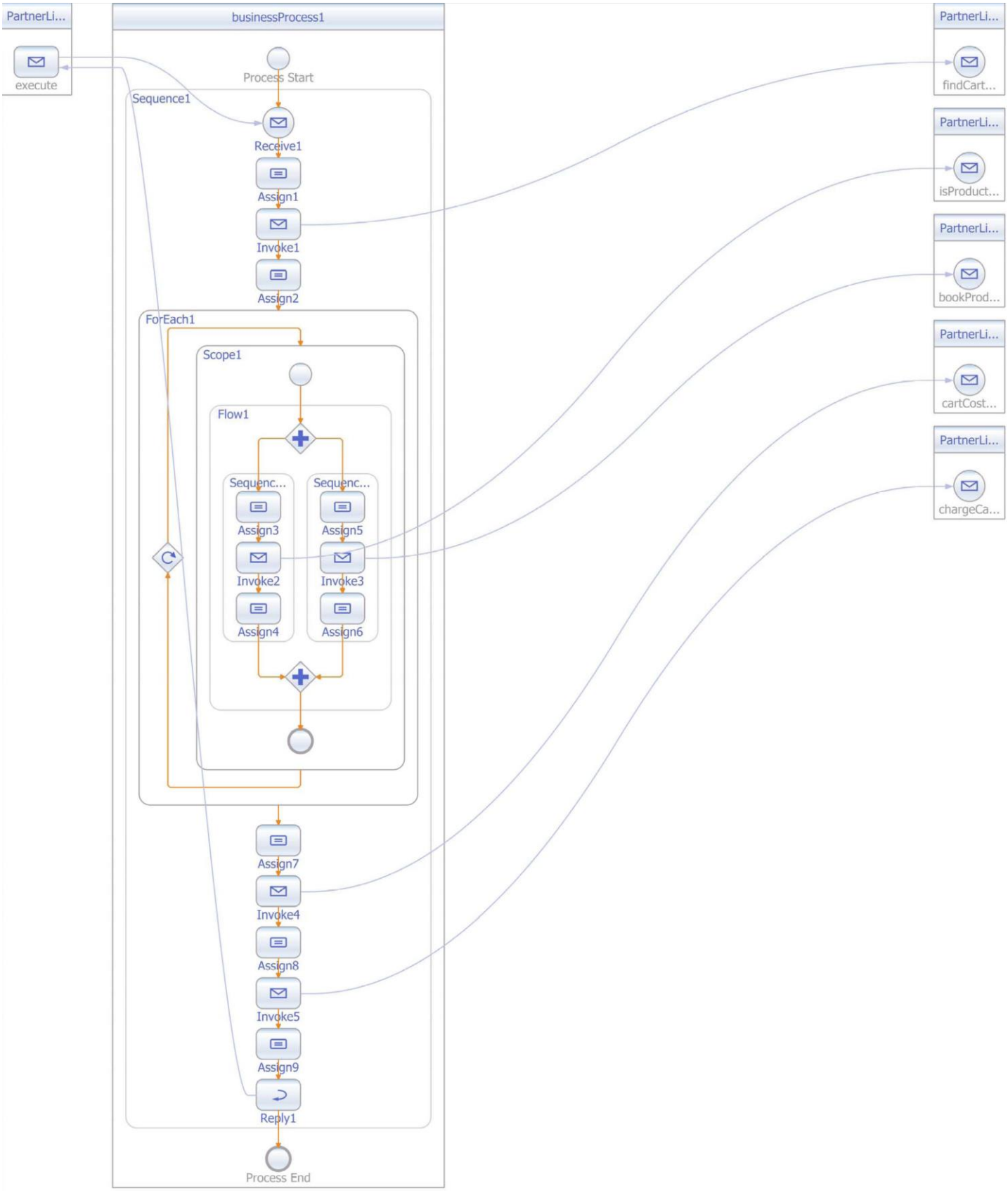
Επιχειρησιακές Διαδικασίες

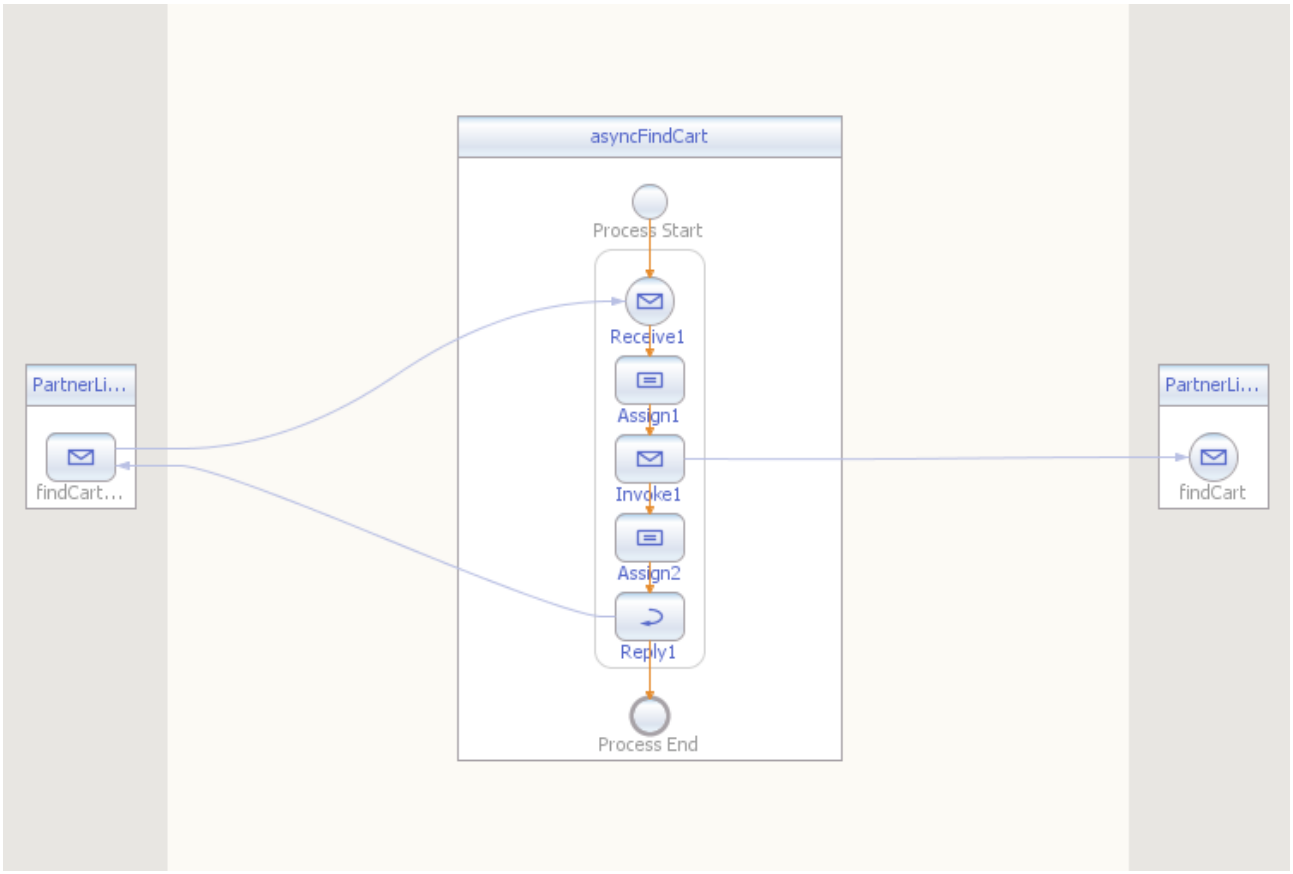
1) **Επεξεργασία Παραγγελίας:** Η επεξεργασία της κάθε παραγγελίας ξεκινά με τη λήψη του κωδικού της παραγγελίας. Η διαδικασία περιλαμβάνει βήματα για:
    - την αναζήτηση των προϊόντων του καλαθιού αγορών,
    - τον έλεγχο των διαθέσιμων αποθεμάτων στις αποθήκες,
    - την κράτηση των αντικειμένων στις αποθήκες,
    - τον υπολογισμό του κόστους και
    - τη χρέωση του πελάτη με την βοήθεια του πιστωτικού ιδρύματος.

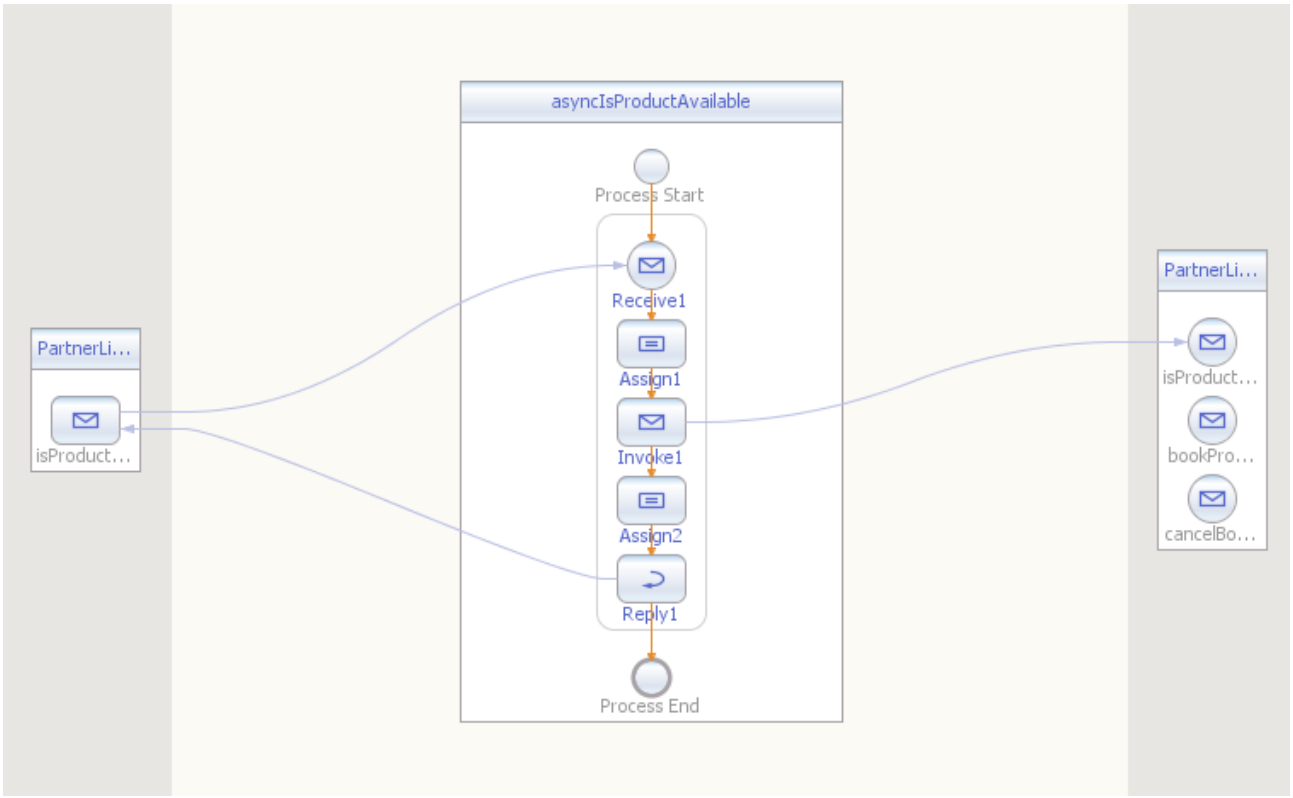
Για την επιτάχυνση της διαδικασίας εξυπηρέτησης παραγγελίας και την αύξηση του αριθμού των ταυτόχρονα εκτελούμενων διεργασιών έχει επιλεγεί:

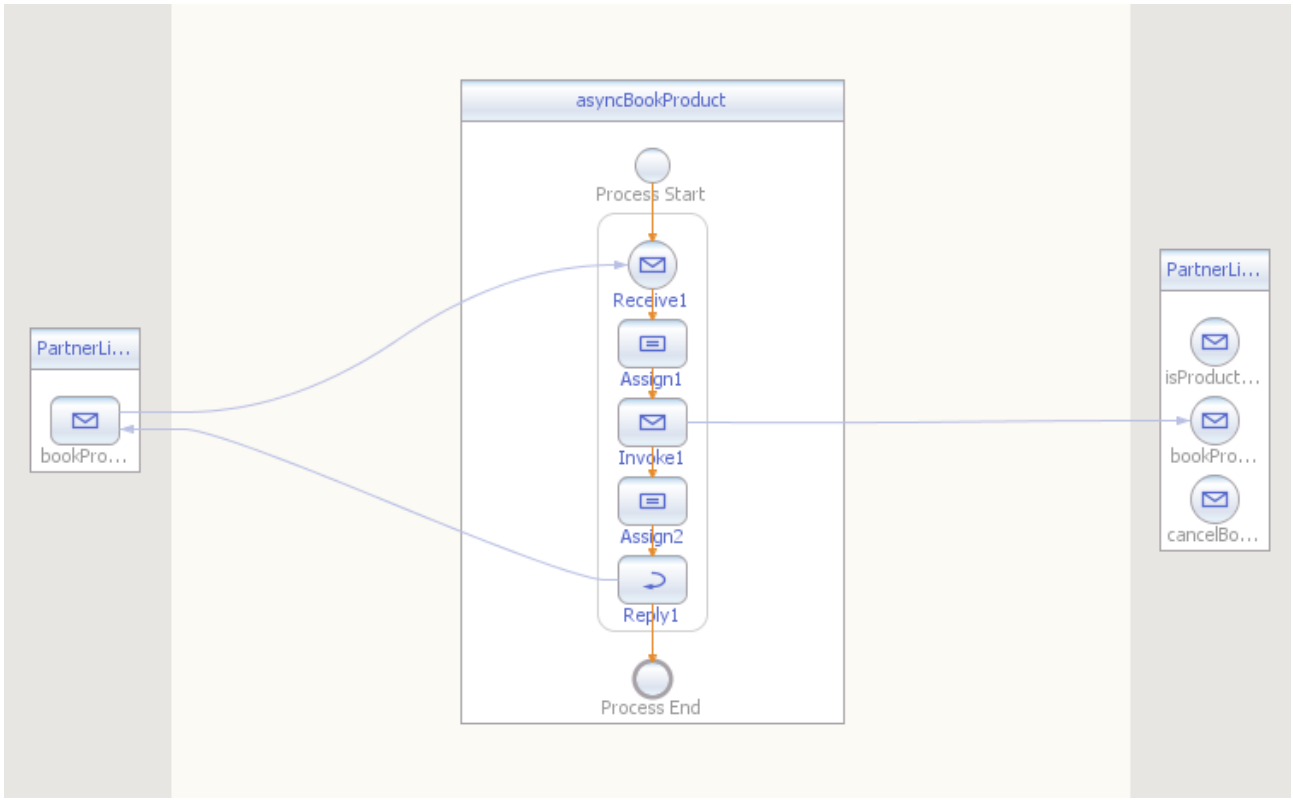
- η παράλληλη εκτέλεση των ελέγχων και των κρατήσεων των αντικειμένων στις αποθήκες, όπως παρουσιάζεται στα διαγράμματα της 1ης επιχειρησιακής διαδικασίας παρακάτω, και
- η ασύγχρονη κλήση της υπηρεσίας του πιστωτικού ιδρύματος.

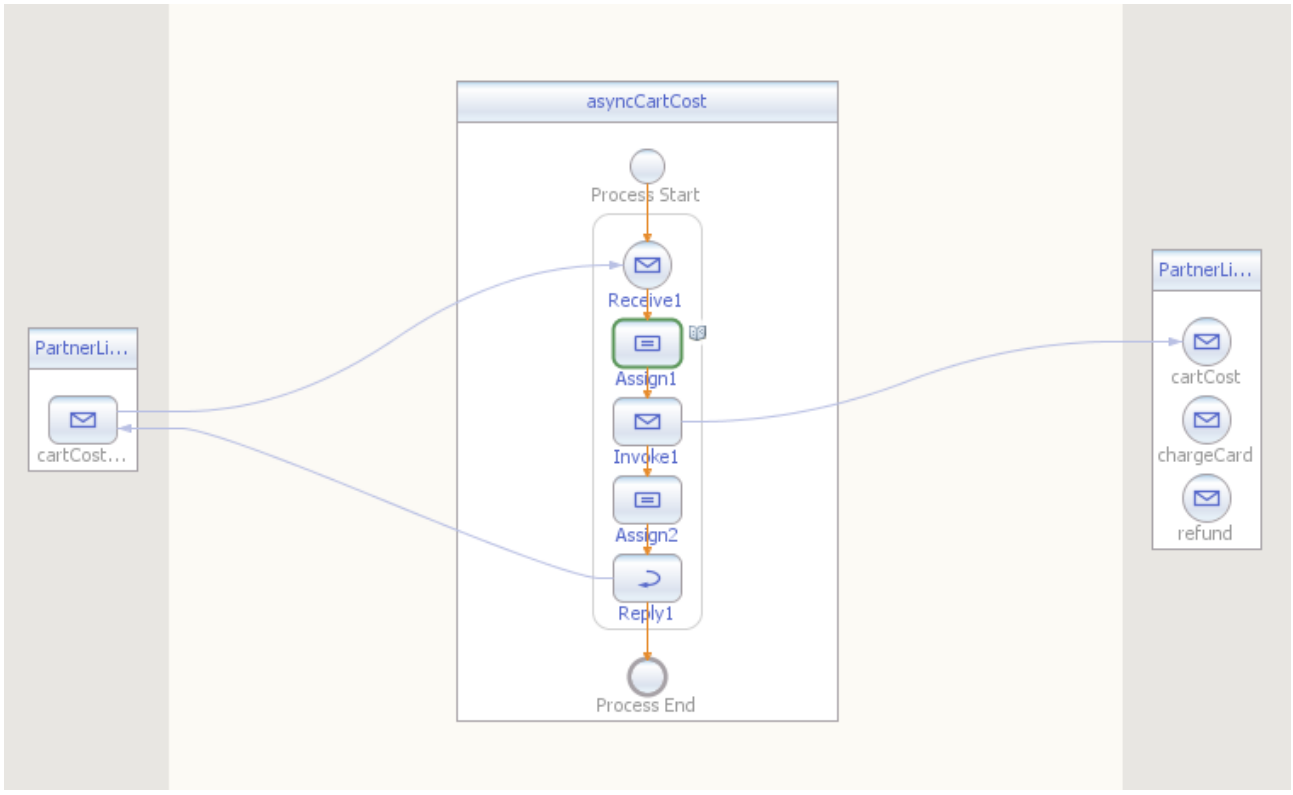
Διαγράμματα 1ης επιχειρησιακής διαδικασίας

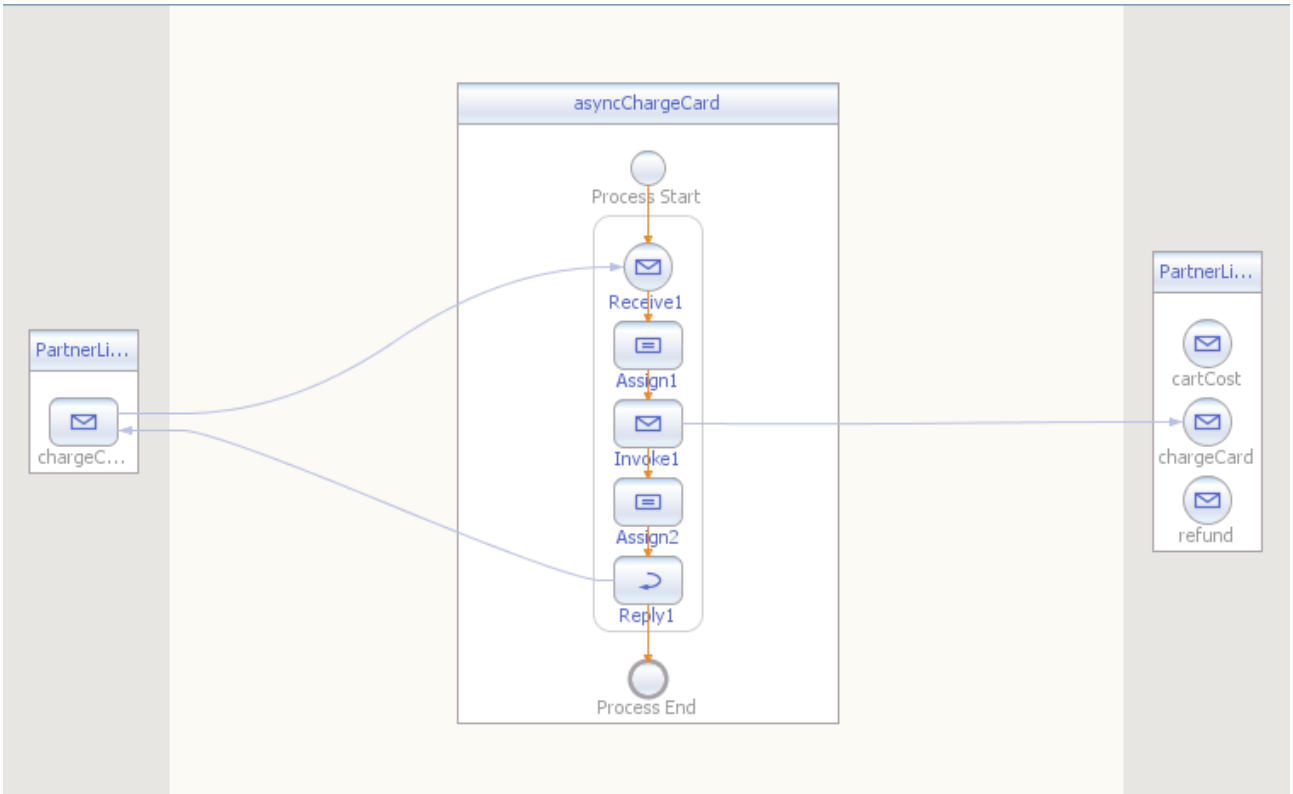



asyncFindCart
Process Start
Receive1
Assign1
Invoke1
Assign2
Reply1
Process End
PartnerLi...
findCart...
PartnerLi...
findCart


asyncIsProductAvailable
Process Start
Receive1
Assign1
Invoke1
Assign2
Reply1
Process End
PartnerLi...
isProduct...
PartnerLi...
isProduct...
bookPro...
cancelBo...


asyncBookProduct
Process Start
Receive1
Assign1
Invoke1
Assign2
Reply1
Process End
PartnerLi...
bookPro...
PartnerLi...
isProduct...
bookPro...
cancelBo...

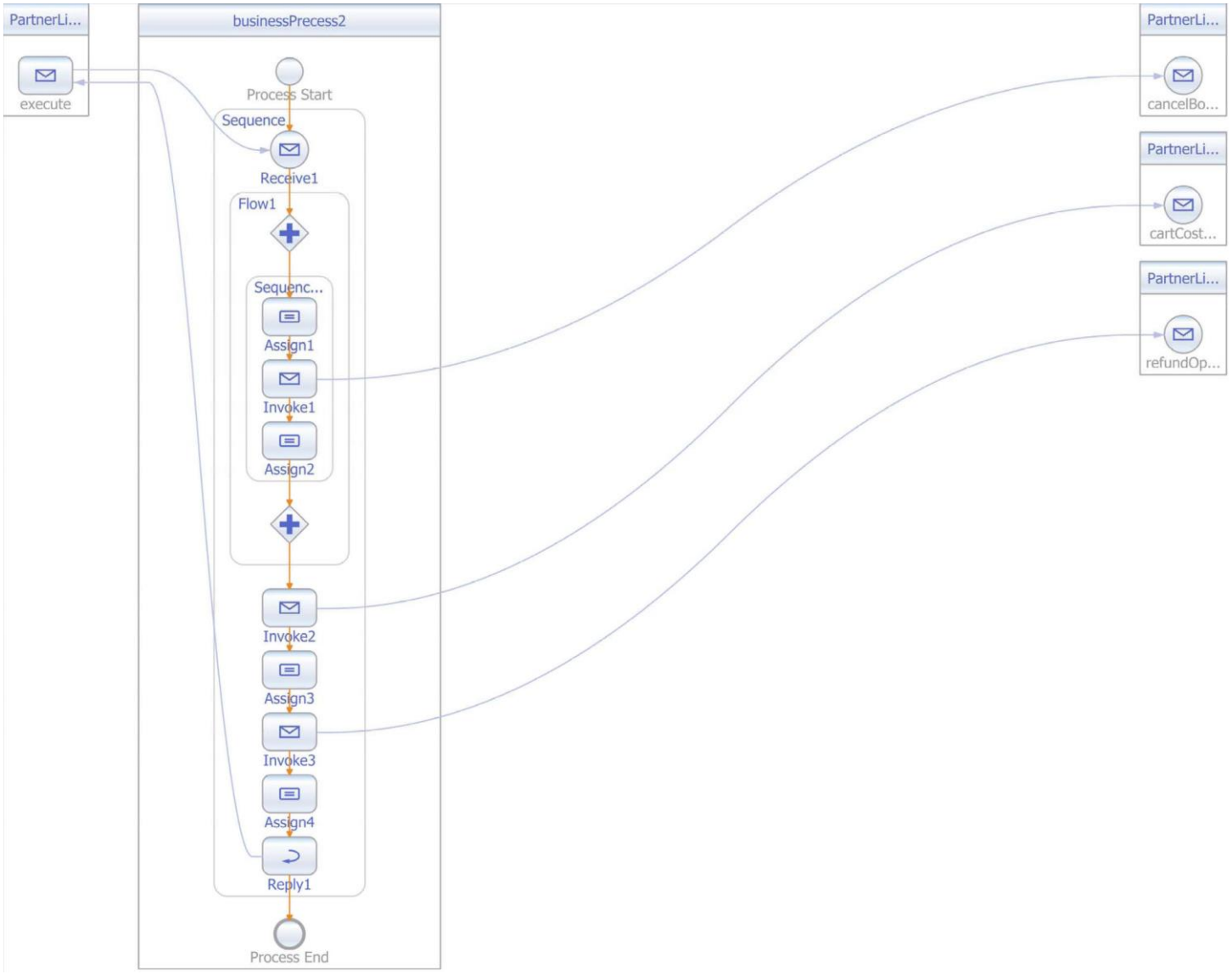

asyncCartCost
Process Start
Receive1
Assign1
Invoke1
Assign2
Reply1
Process End
PartnerLi...
cartCost...
PartnerLi...
cartCost
chargeCard
refund

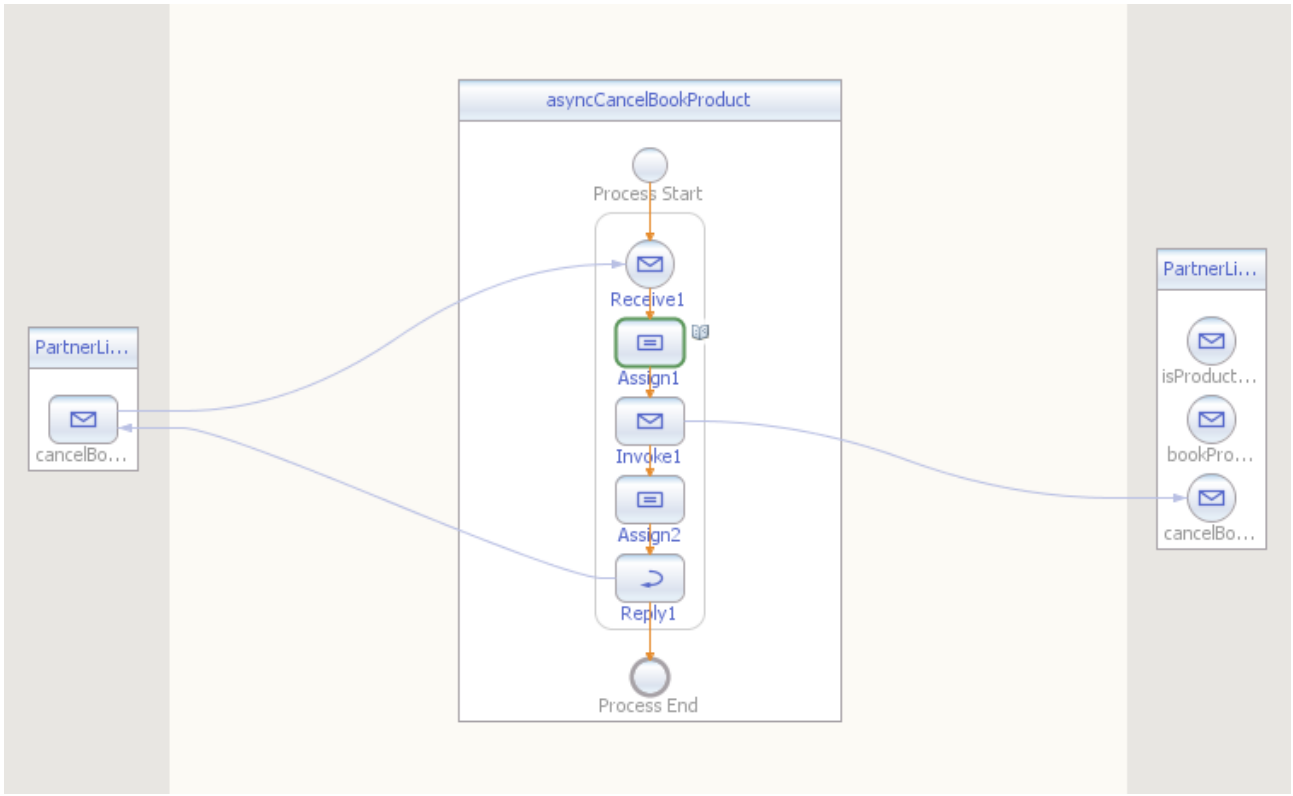


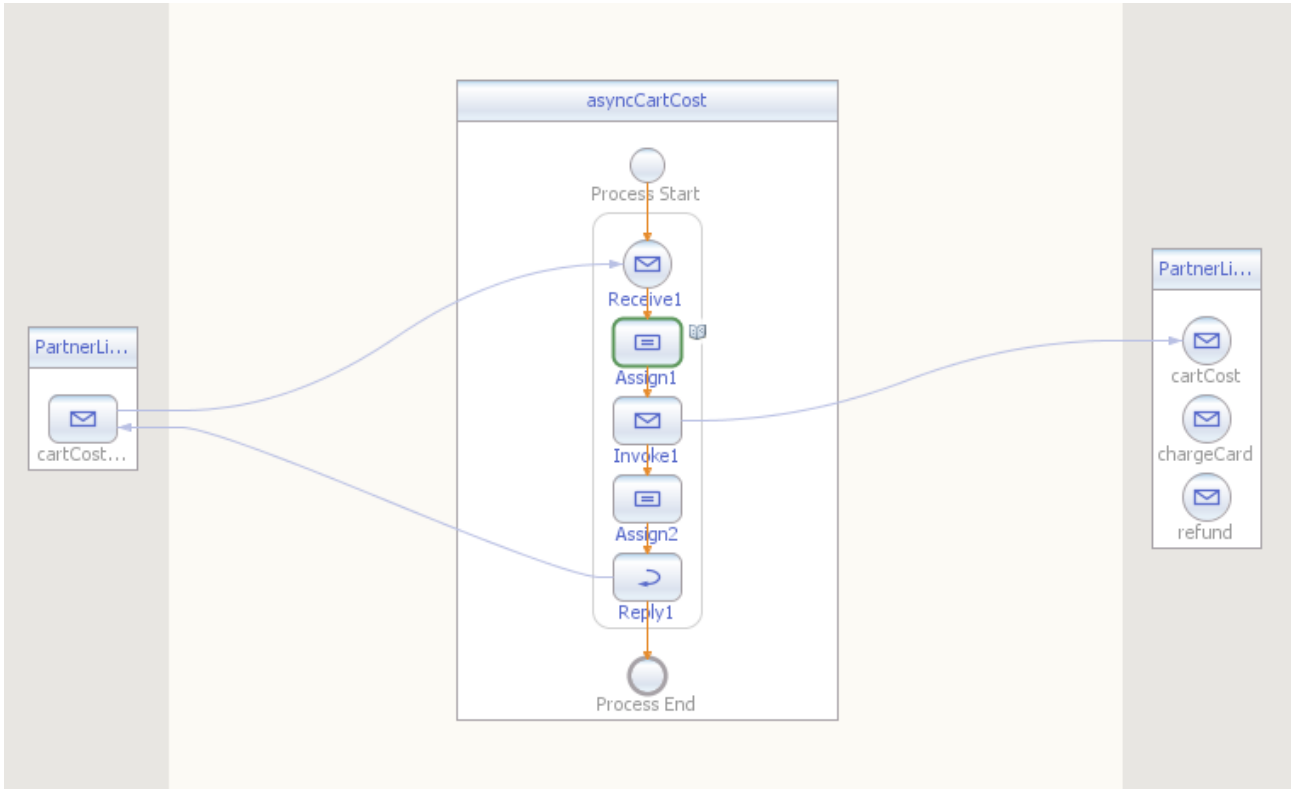
2) **Ακύρωση παραγγελίας:** Η διαδικασία της ακύρωσης κάποιας παραγγελίας περιλαμβάνει τα απαραίτητα βήματα για την αναίρεση των ενεργειών της διαδικασίας επεξεργασίας (ΕΔ1). Συγκεκριμένα η διαδικασία αυτή ξεκινά με τη λήψη του κωδικού της παραγγελίας και περιλαμβάνει βήματα τα οποία επιτρέπουν:
    - την ακύρωση των κρατήσεων που έχουν γίνει για λογαριασμό μιας παραγγελίας σε όλες τις εμπλεκόμενες αποθήκες
    - την ακύρωση των σχετικών πιστώσεων στην κάρτα του πελάτη.

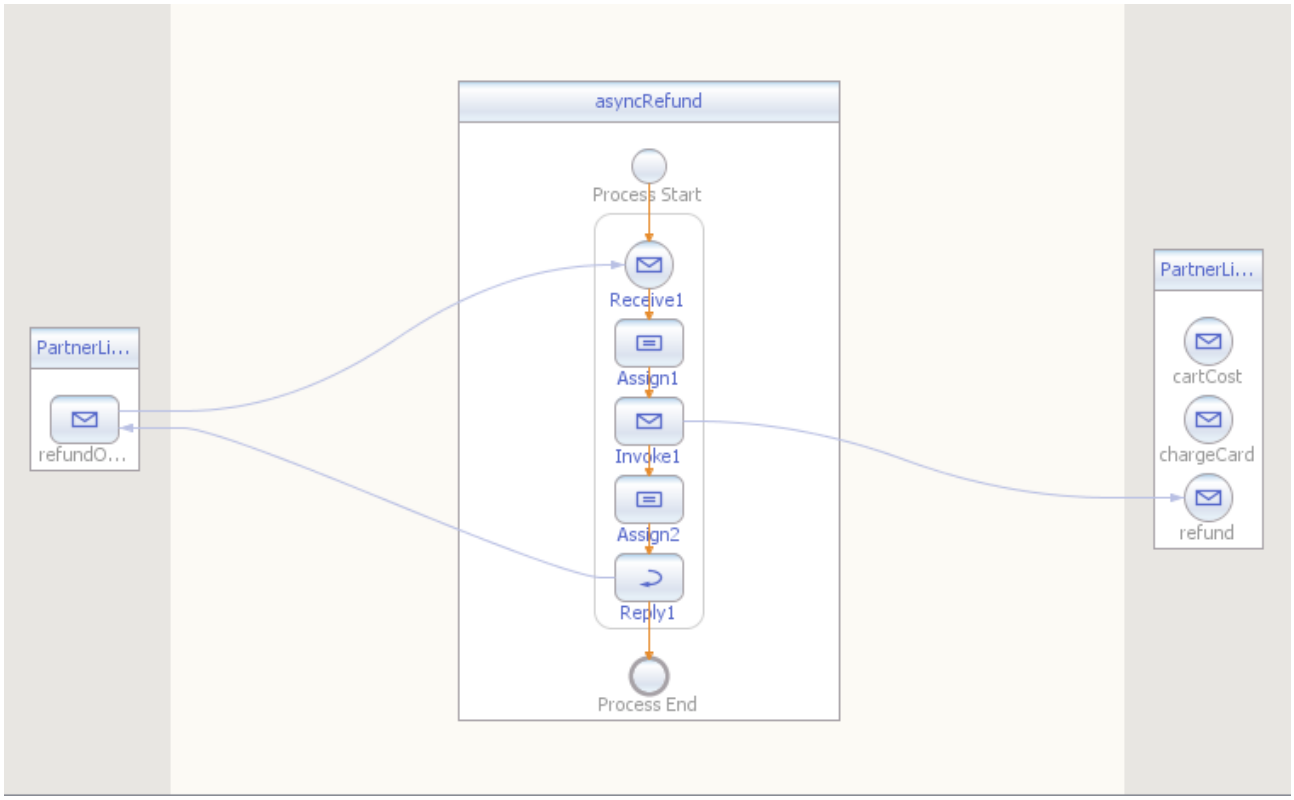
Όμοια και στην 2η επιχειρησιακή διαδικασία, στην διαδικασία ακύρωσης, οι σχετικές ακυρώσεις στις αποθήκες εκτελούνται παράλληλα ενώ η διαδικασία ακύρωσης της χρέωσης της πιστωτικής κάρτας εκτελείται ασύγχρονα.

Διαγράμματα 2ης επιχειρησιακής διαδικασίας




asyncCancelBookProduct
Process Start
Receive1
Assign1
Invoke1
Assign2
Reply1
Process End
PartnerLi...
cancelBo...
PartnerLi...
isProduct...
bookPro...
cancelBo...


asyncCartCost
Process Start
Receive1
Assign1
Invoke1
Assign2
Reply1
Process End
PartnerLi...
cartCost...
PartnerLi...
cartCost
chargeCard
refund


asyncRefund
Process Start
Receive1
Assign1
Invoke1
Assign2
Reply1
Process End
PartnerLi...
refundO...
PartnerLi...
cartCost
chargeCard
refund

3. Προβλήματα

- Κατά την εγκατάσταση περιβάλλοντος

Κατά την εγκατάσταση περιβάλλοντος χρειάστηκε, στον υπολογιστή που δουλεύαμε που είχε λειτουργικό σύστημα Windows 7, να κάνουμε downgrade την έκδοση της java που χρησιμοποιούσαμε σε 1.6 και να κατεβάσουμε μια την παλιά έκδοση του NetBeans 6.8. Στην συνέχεια για να σεταριστούν σωστά τα plug-ins του NetBeans, δεδομένου της παλιάς έκδοσής του, το link που δινόταν στα εργαστήρια δεν ήταν πλέον active. Έτσι μετά από αρκετό ψάξιμο στο διαδίκτυο, βρήκαμε το παρακάτω url: *http://updates.netbeans.org/netbeans/updates/6.7.1/uc/final/beta/catalog.xml.gz* και μπορέσαμε και κάναμε εγκατάσταση το SOA και XML Schema and WSDL.

- Κατά την εκτέλεση της εφαρμογής

Στην παρούσα εργασία χρησιμοποιήσαμε τον GlassFish v3 server, τον default server που έχει η 6.8 έκδοση του NetBeans. Κατά το deploy των web services, χτύπαγε error αν δεν υπήρχε κενός constructor στα domain αντικείμενά μας. Επίσης επειδή χρησιμοποιούμε και MySQL βάση δεδομένων, κατά την σύνδεση του DatabaseManager με την βάση εγείρει δυο Exceptions τα οποία είναι: ClassNotFoundException και SQLException. Αν αυτά τα exception γίνονται throw στα web services, χτυπάει ο GlassFish WebServiceException με αδυναμία να δημιουργήσει JAXBContext. Έτσι στα web services γίνεται throw μόνο IOException.

Επιπλέον όταν φτιάξαμε το CompositeApp project και προσπαθήσαμε να το κάνουμε deploy στον GlassFish server έκανε fail το Casa deployment με το εξής Exception: “java.lang.RuntimeException: The application server definition file \.netbeans\6.8\config\J2EE\InstalledServers\.nbattrs is corrupted or it doesn't contain the target server instance”. Αφού ψάξαμε αρκετά στο διαδίκτυο δεν βρήκαμε λύση στο παραπάνω πρόβλημα με αποτέλεσμα να μην καταφέρουμε να το κάνουμε deploy.