Online Shop App

Software Engineering Lab SS23

Abdelrahman Fetih (3092731), Adham Youssef(3178333), Hao Cui(3178999), Ahmed Nasrallah (3088516), Mohamed Elnahrawy (3102673)

Lab Time: Do. 12-14

Supervisor: Marcel Schweikert

Date of submission: 06.06.2023

A1.1

Requirements:

- R1: To use the web application, Customers need to register and login using a unique username to use the shopping cart and submit orders.
- R2: A product list page showing all products offered by the supermarket, and customers should be able to view the product list and individual product pages and the product list should be sorted in a descending order by price.
- R3: Registered Customers should be able to add products to their shopping cart from the product detailpage and they should only be able to add up to 5 units of a product to their shopping cart.
- R4: An automated process should remove products from the list if they are out of stock.
- R5: Each product should have a name, a unique product number, a description, a price, and a stock quantity.
- R6: Employees should be able to add new products to the list.
- R7: Employees should be able to remove existing products.
- R8: After submitting an order, a new order should be created with the purchased items andthe customer's information.

Assumptions:

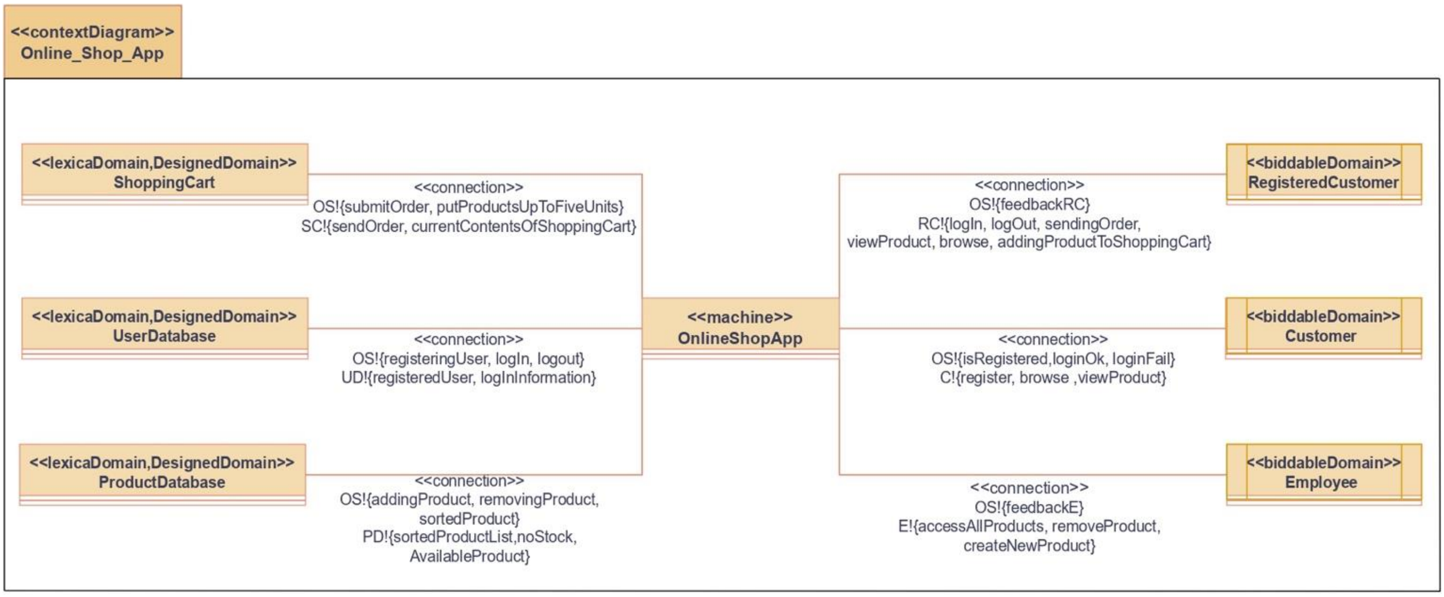
- A1: Customers will only add products to their shopping cart, which they want to buy.
- A2: Customers will use a safe password.
- A3: When customers send an order out of the shopping cart, a new order should be created.
- A4: Employees will not create a product that already exists in the list of the products.
- A5: The web application is usable by all customers, regardless of age.
- A6: the store will not send the order until the customer has paid.

Facts:

- F1: A product consists of a name, a unique product number, a description, a price, and a stock quantity.
- F2: A shopping cart can be only holding up to five units.
- F3: the web creator is not it's administrator.

A1.2

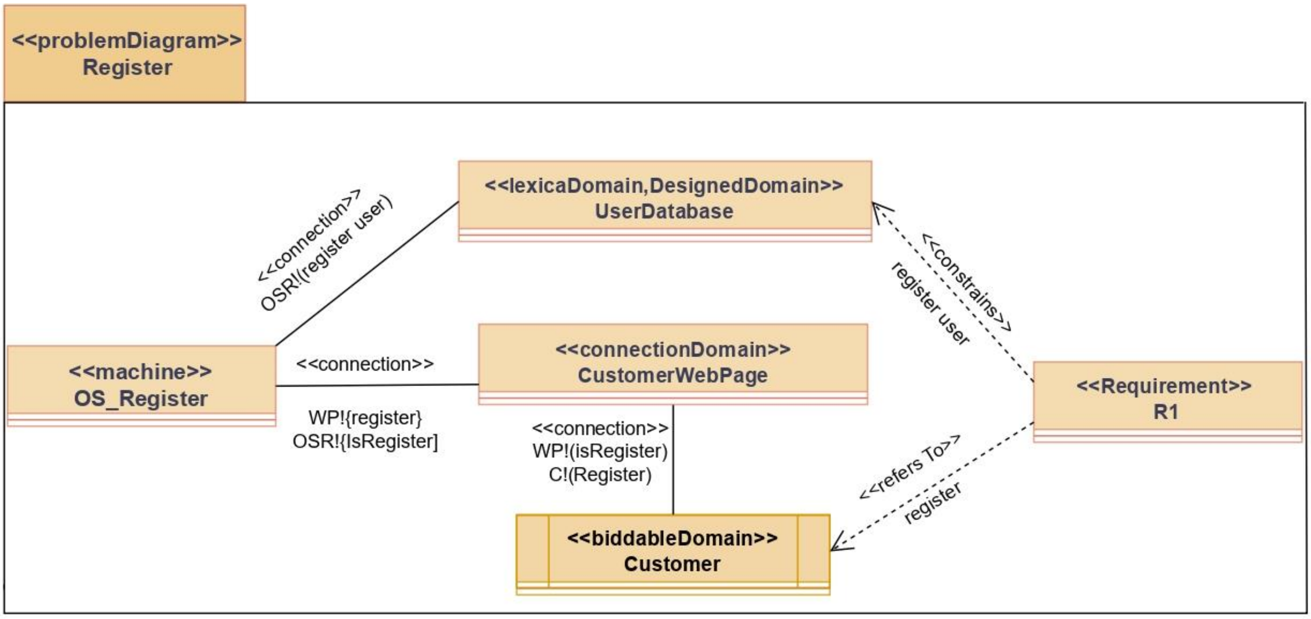
Context Diagram

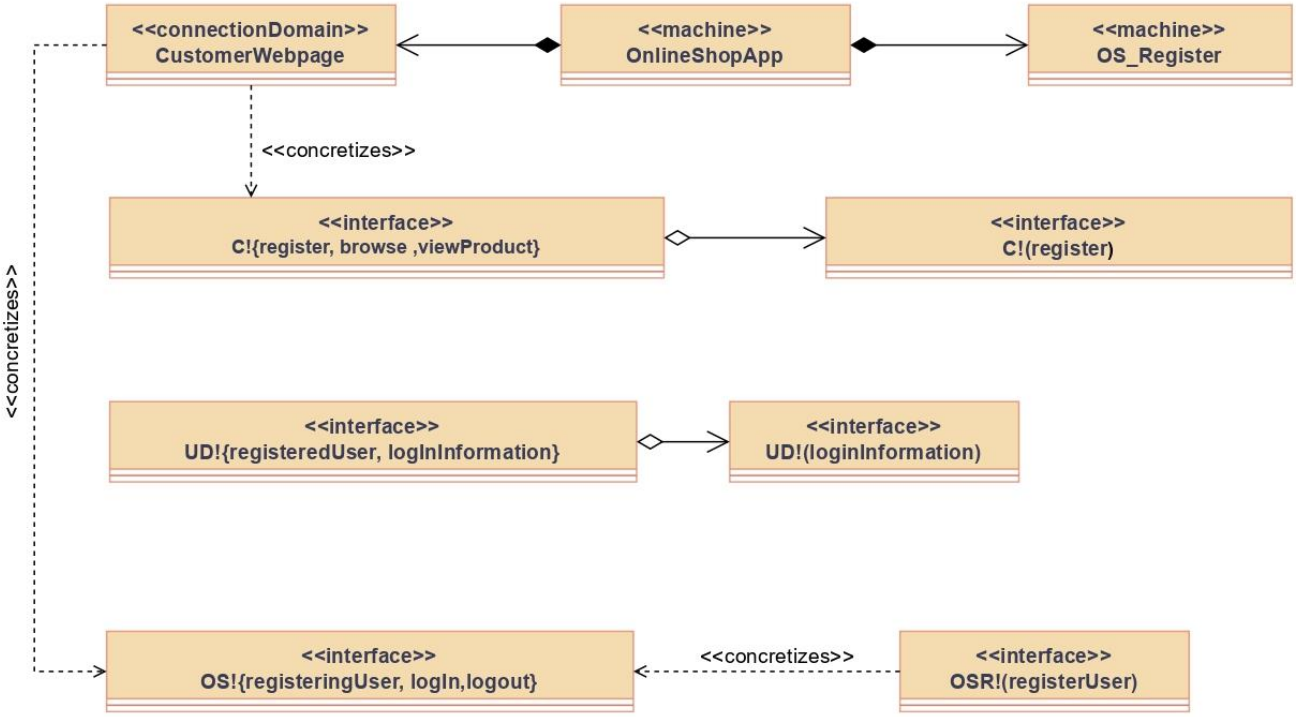

<<contextDiagram>>
Online_Shop_App
<<lexicaDomain,DesignedDomain>>
ShoppingCart
<<connection>>
OS!{submitOrder, putProductsUpToFiveUnits}
SC!{sendOrder, currentContentsOfShoppingCart}
<<lexicaDomain,DesignedDomain>>
UserDatabase
<<connection>>
OS!{registeringUser, logIn, logout}
UD!{registeredUser, logInInformation}
<<lexicaDomain,DesignedDomain>>
ProductDatabase
<<connection>>
OS!{addingProduct, removingProduct, sortedProduct}
PD!{sortedProductList,noStock, AvailableProduct}
<<machine>>
OnlineShopApp
<<connection>>
OS!{feedbackRC}
RC!{logIn, logOut, sendingOrder, viewProduct, browse, addingProductToShoppingCart}
<<biddableDomain>>
RegisteredCustomer
<<connection>>
OS!{isRegistered,loginOk, loginFail}
C!{register, browse ,viewProduct}
<<biddableDomain>>
Customer
<<connection>>
OS!{feedbackE}
E!{accessAllProducts, removeProduct, createNewProduct}
<<biddableDomain>>
Employee

A2.1

Customer Registration

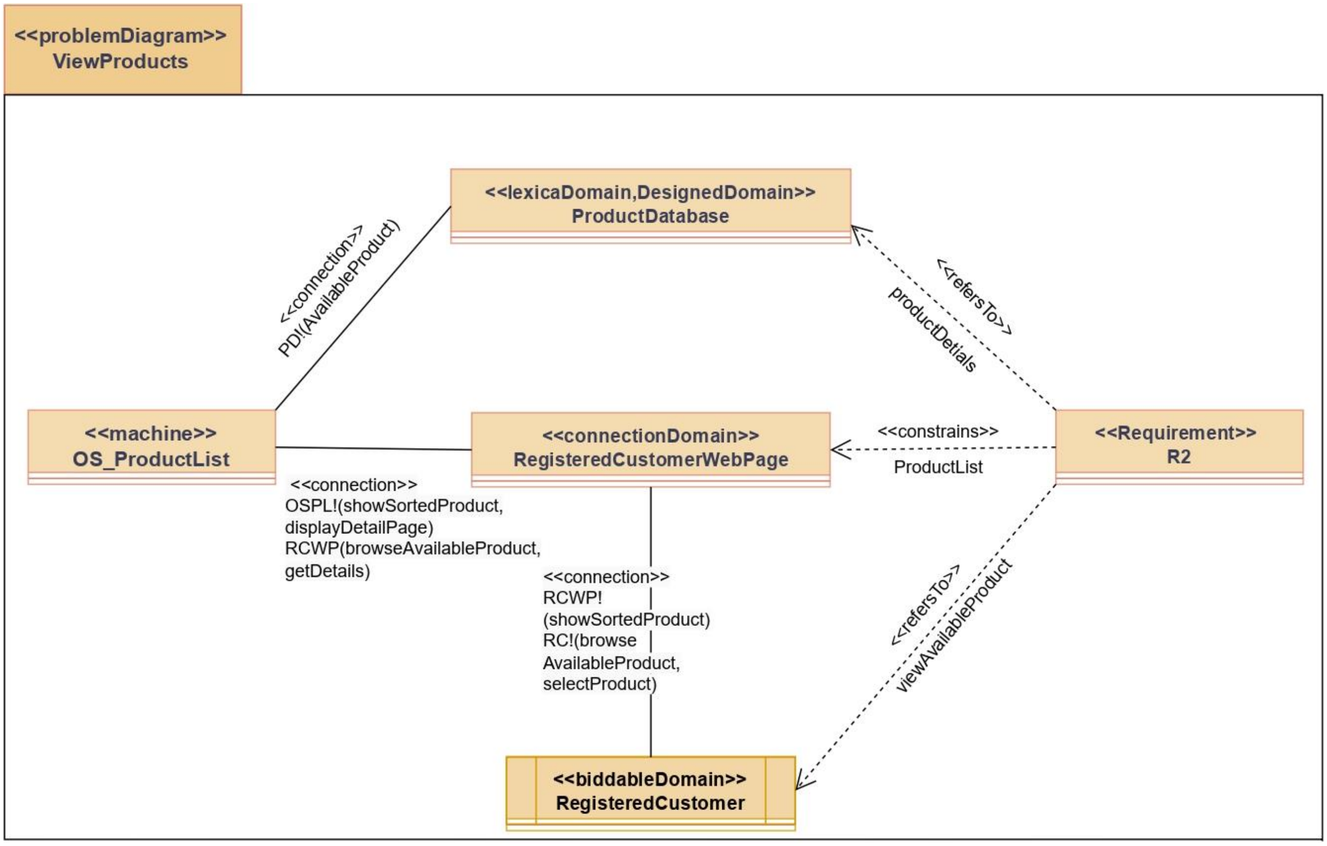
R1: To use the web application, Customers need to register and login using a unique username to use the shopping cart and submit orders.

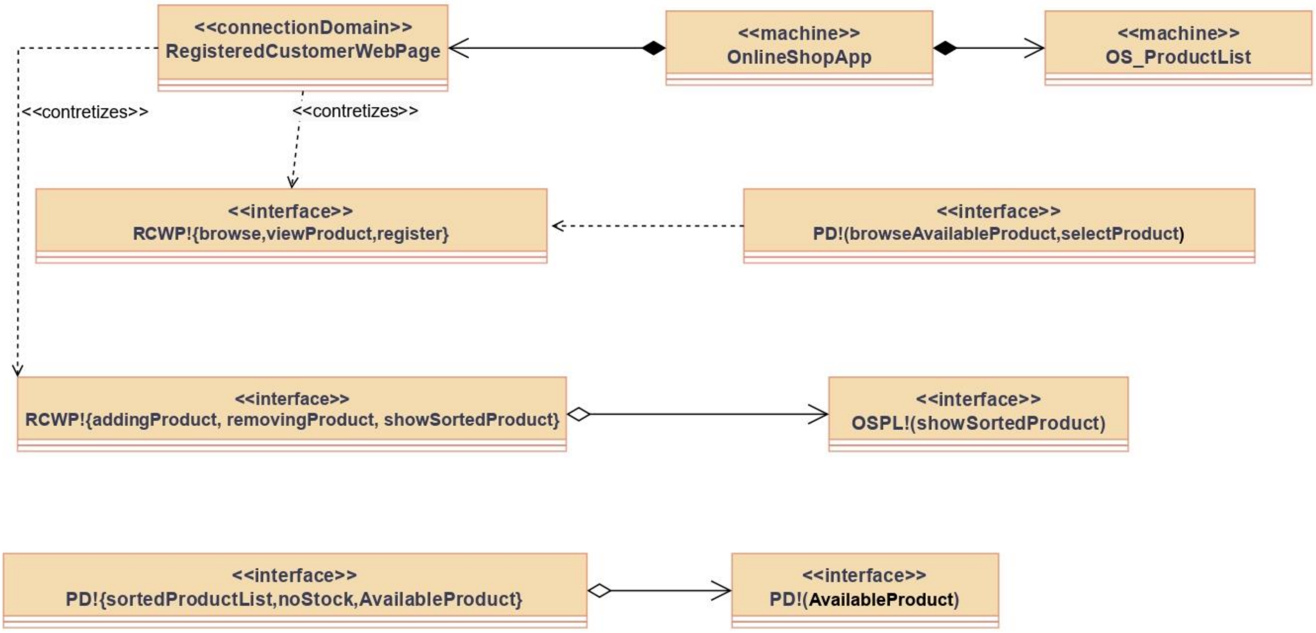




ViewProductList

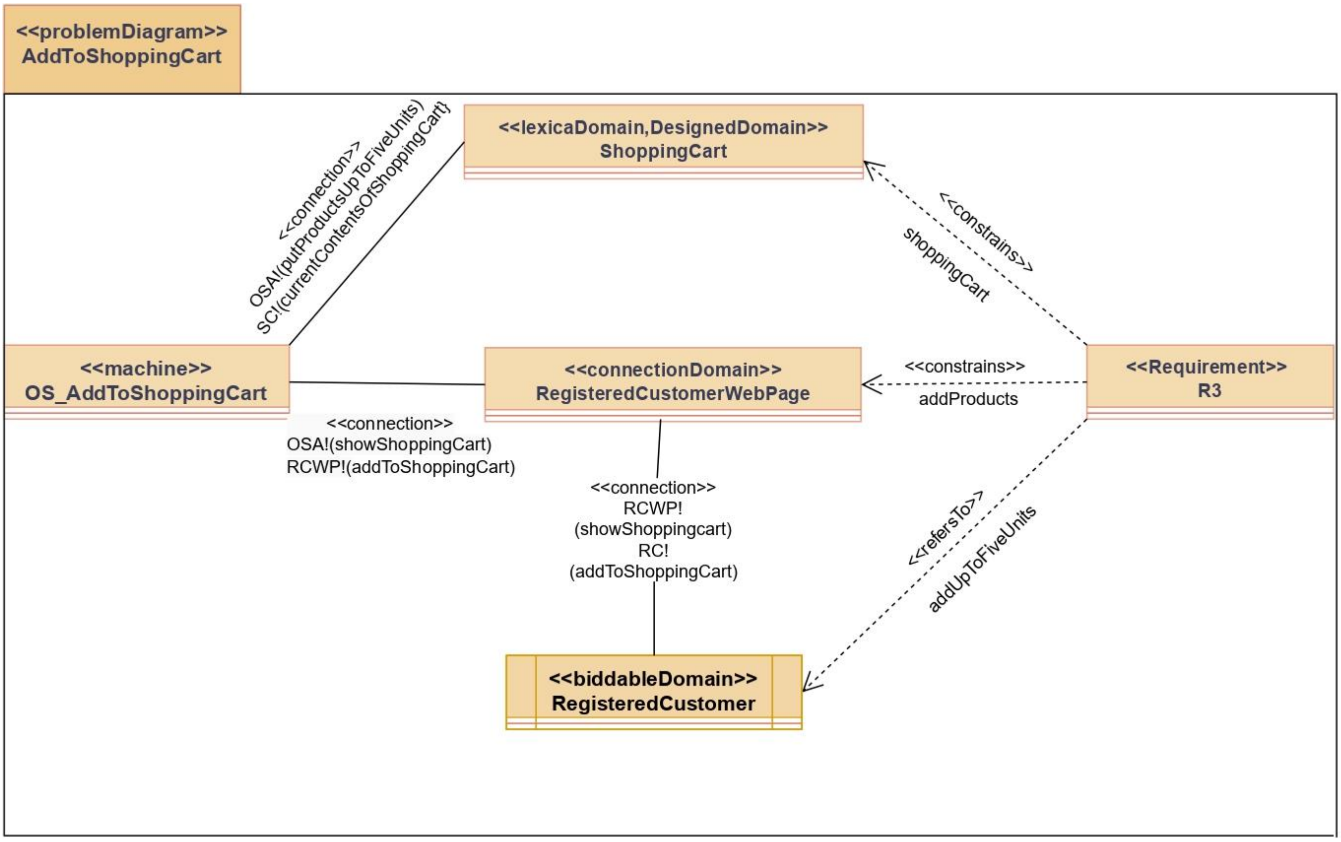
R2: A product list page showing all products offered by the supermarket, and customers should be able to view the product list and individual product pages and the product list should be sorted in a descending order by price.

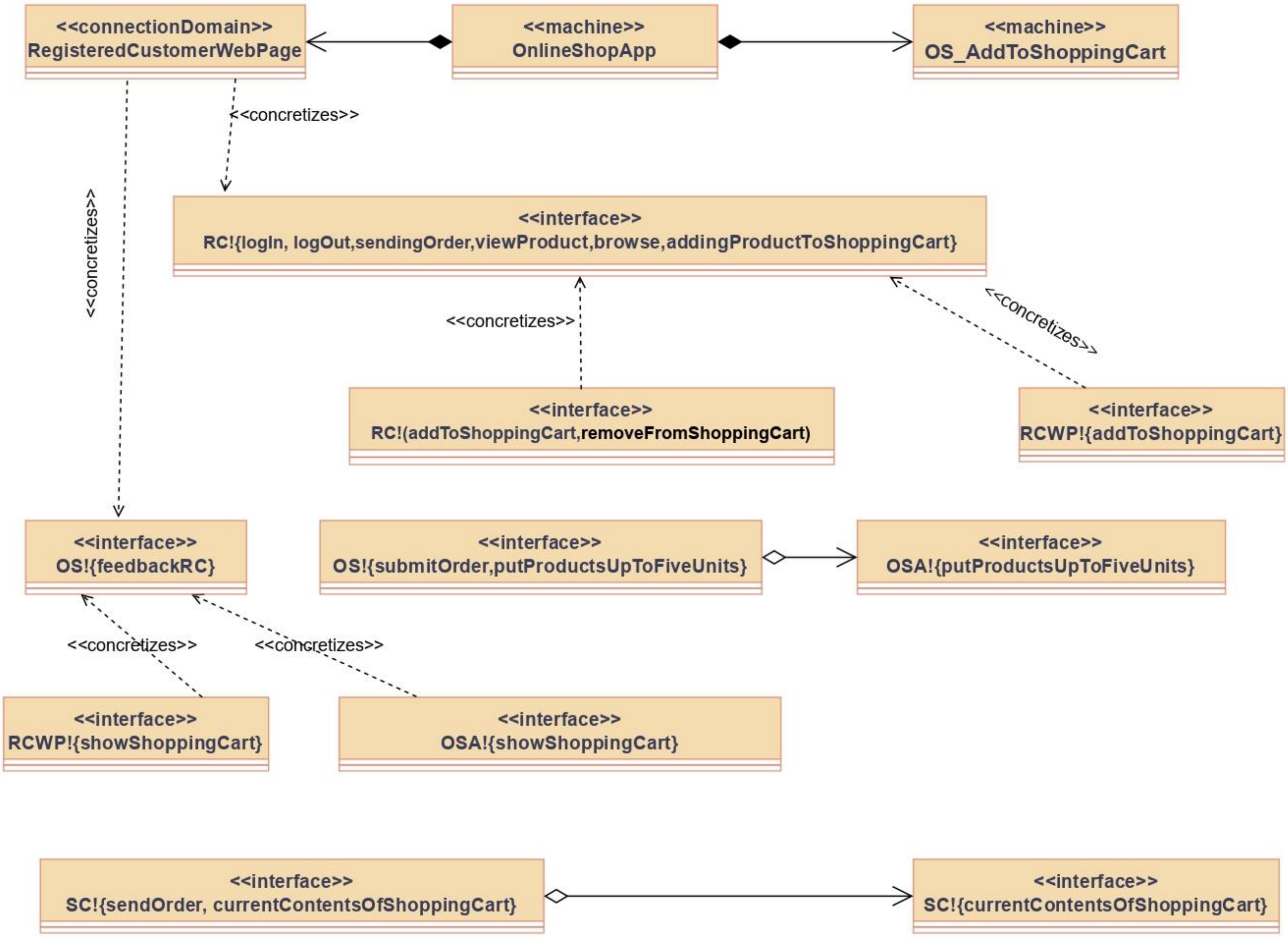




AddToShoppingCart

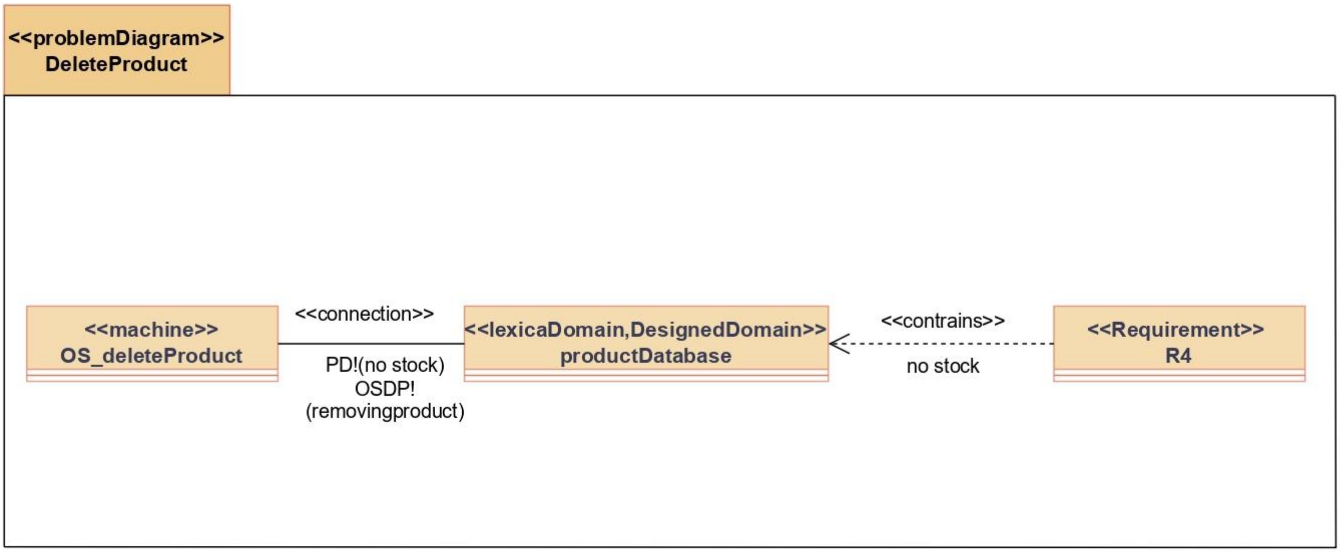
R3: Registered Customers should be able to add products to their shopping cart from the product detail page and they should only be able to add up to 5 units of a product to their shopping cart.

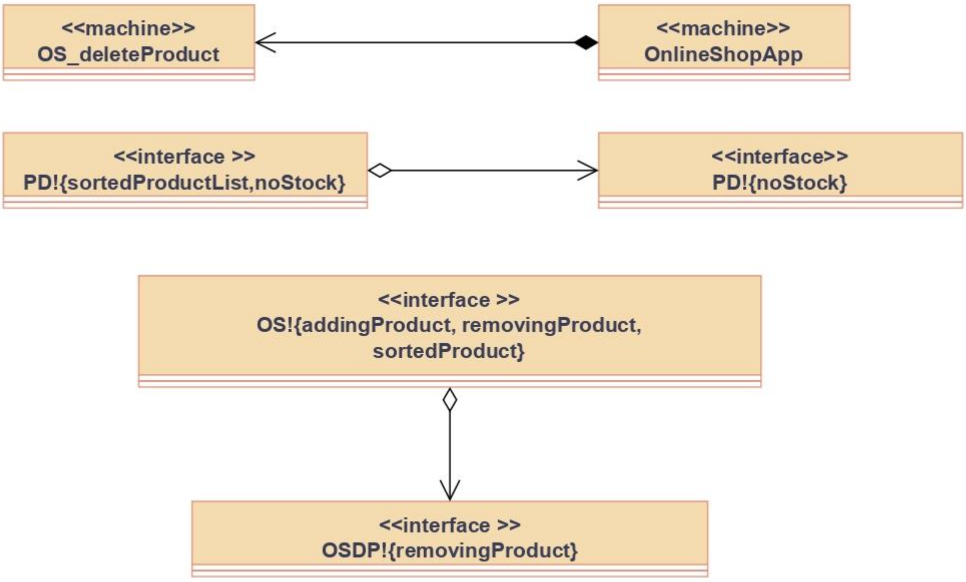




DeleteProducts

R4: An automated process should remove products from the list if they are out of stock.





A2.2 Problem Frames

- For the Problem Diagram 1.2.1 where the problem Register is handled (from Requirements R08), the problem frame that fits is **update** (2).

- For the Problem Diagram 1.2.2 where the problem ProductsList is handled (from Requirement R01), the problem frame that fits is **update** (2).

- For the Problem Diagram 1.2.3 where the problem AddToShoppingCart is handled (from Requirement R07), the problem frame that fits is **update** (2).

- For the Problem Diagram 1.2.4 where the problem AutomaticDelete is handled (from Requirements R03), the problem frame that fits is **simple transformation**.

requirements	Covered in	Contained domain	Domain type	constrained	Coctrolled phenomena
R01	pdRegister	OS_Register	machine		registeredUser
		Customer	biddable		register
		UserDatabase	Lexical, designed	X	registerUser
		webPage	connection		isRegistered
R02	pdProductList	OS_ProductList	machine		showSortedProducts
		Customer	biddable		Browse, availableProduct, selectProduct
		ProductDatabase	Lexical, designed		productDetails
		webPage	connection	X	sortedProductsIn DesendingOrder
R03	pdAddTo ShoppingCart	OS_addTo ShoppingCart	machine		showShoppingCart
		Registered Customer	biddable		addToShoppingCart RemoveFromShoppingCart
		shoppingCart	Lexical, designed	X	currentContentOf ShoppingCart
		webPage	connection		showShoppingCart
R04	pdDeleteProduct	OS_DeleteProduct	machine		removingProduct
		ProductDatabase	Lexical, designed	X	noStock

Abstract software specification & Sequence Diagram:

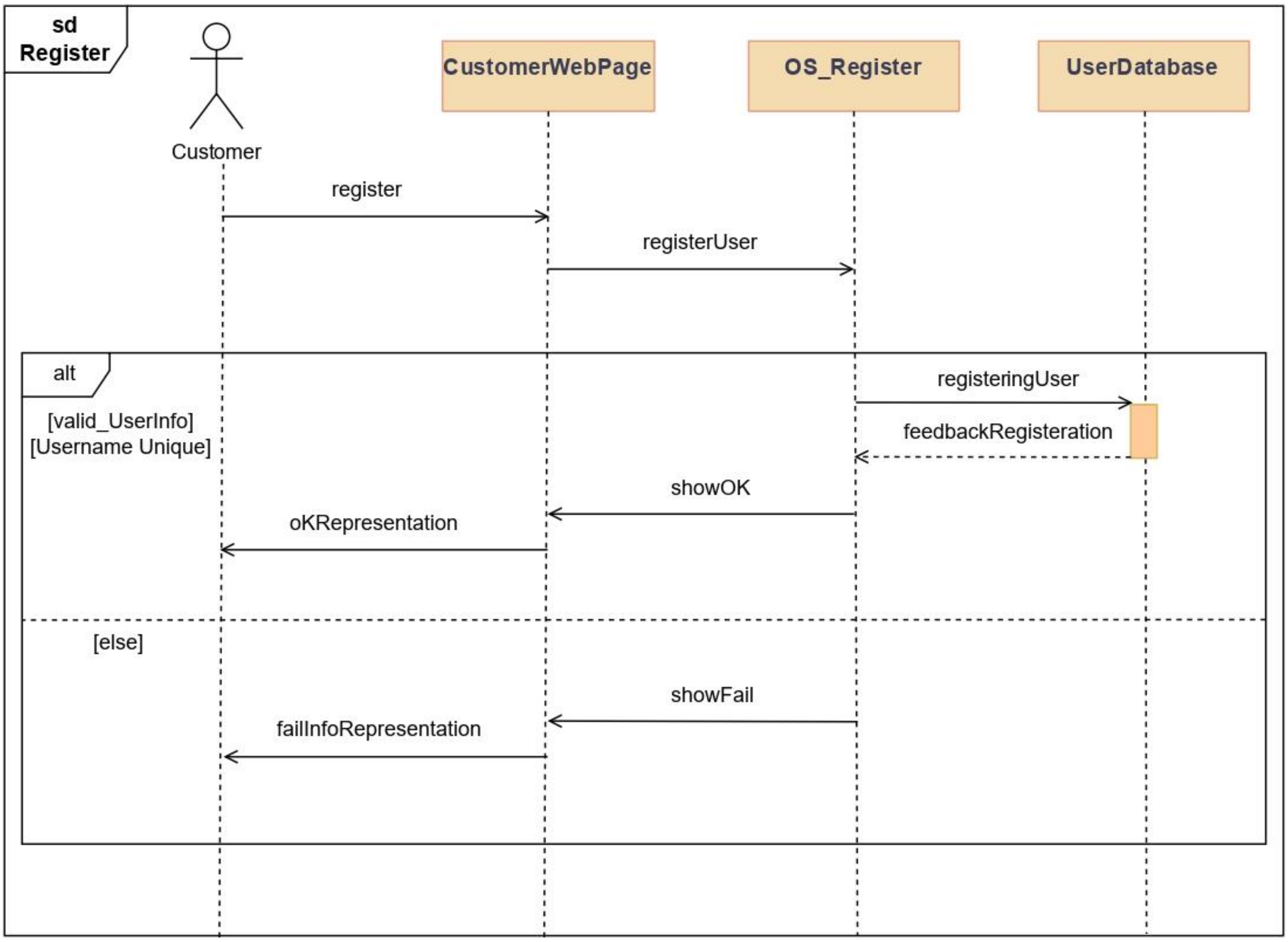
Specification and Sequence Diagram for OS_Register:

R01: In order to use a shopping cart, customers must register beforehand. To do this, they register with a login name and a password. The login name must be unique for all customers.
RegisteredCustomerWebpage (S01a): When the webpage receives the command ‘’ Register ’’, then the command is forwarded to the machine with the command ‘‘ Register‘‘. The results are received via the command ‘‘isRegister‘‘ and shown to the customer by ‘‘ isregister‘‘.

OS_Register (S01b): When the machine receives the command ,‘‘ Register‘‘, then the command is forwarded to the Userdatabase with the command ,‘‘ registerUser‘‘. The results are returned via the command ,‘‘isregister‘‘.

UserDatebase (S01c): After receiving the command ‘’ registerUser’’ the result are returned as the data ,‘‘ isRegister‘‘.

Correctness condition (S01a) Λ (S01b) Λ (S01c) =⇒ (R01)



Specification and Sequence Diagram for OS_ProductList :

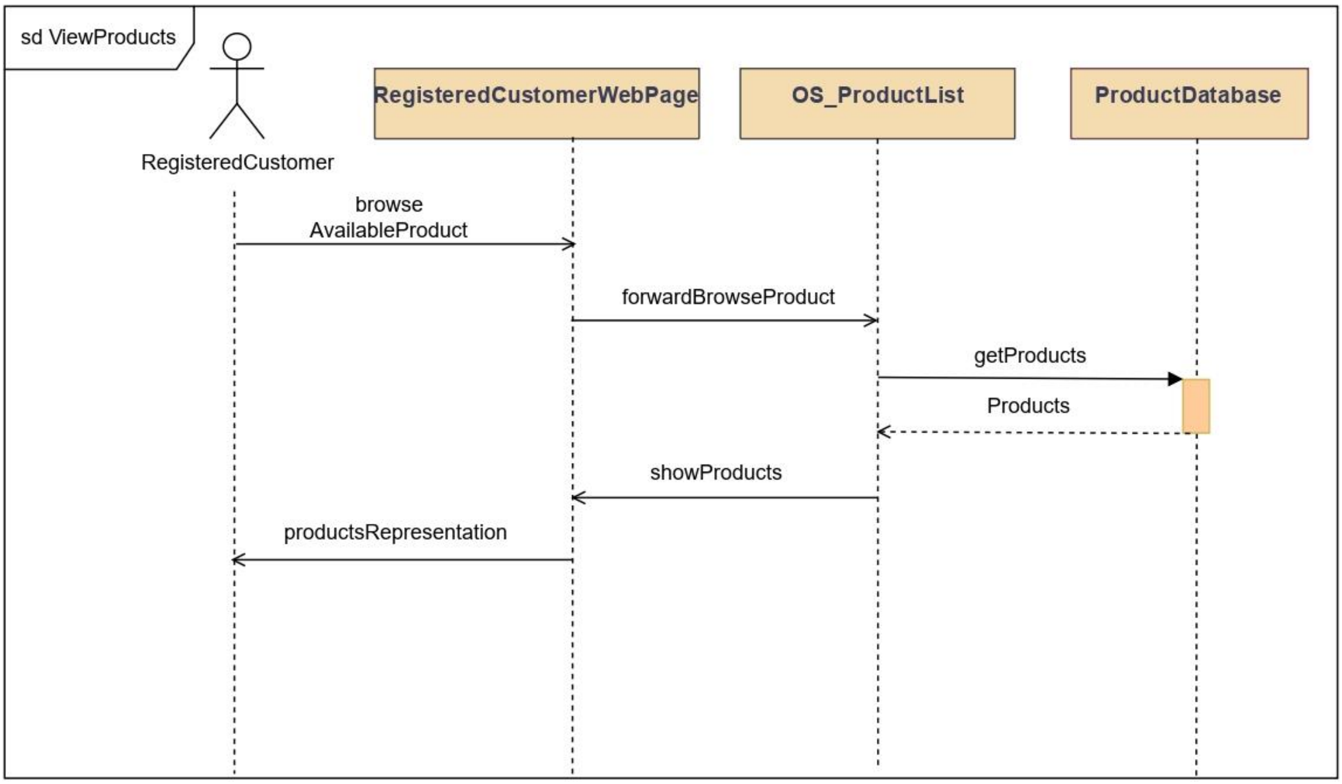
R02 : On the supermarket's website, it should be possible to see a list of the products in the range. The customers themselves should be able to view the products. They can either do this on the overview page or display an extra page of an individual product. They can reach this page by clicking on a product in the list. The list itself should be sorted in descending order by price.

Webpage (S02a) : When the webpage receives the command ‘’ browseAvailableProduct’’ and ‘’ selectProduct ‘’, then the command is forwarded to the machine with the command ‘’ browserAvailableProduct’’. The results are received via the command ‘’ showSortedProduct’’ and ‘’ displayDetailPage’’ and shown to the customer by ‘’showSortedProduct’’ and ‘’displayDetailPage’’.

OS_ProductList (S02b) : When the machine receives the command ‘’ browseAvailableProduct’’, the available Product are selected with the command ‘’browse_availableProduct’’ and received as the data ‘’availableProduct’’.The results are returned via the command ‘’ showsortedProduct’’ and ‘’displayDetailPage’’.

ProductDatabase (S02c) : After receiving the command ‘’ browserAvailableProduct’’ the results are returned as the data ‘’availableProduct’’.

Correctness condition (S02a) Λ (S02b) Λ (S02c) =⇒ (R02)



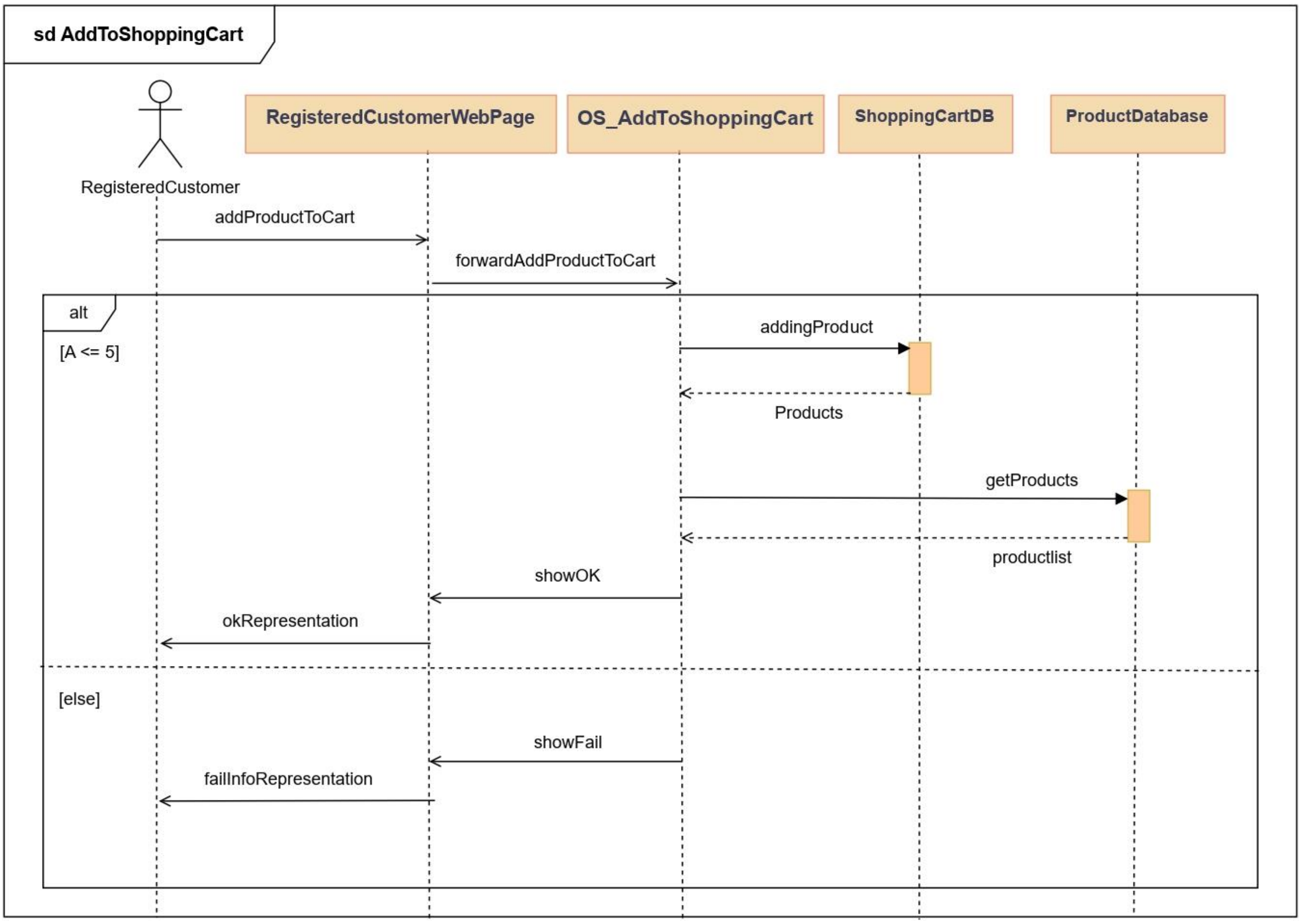
Specification and Sequence Diagram for OS_AddShoppingCart:

R03 : When customers want to buy a product, they can add it to their shopping cart on the detail page. To avoid toilet paper overspending, customers can add a maximum of 5 units of a product to their shopping cart.
Webpage (S03a) : When the webpage receives the command ''addToShoppingCart'', then the command is forwarded to the machine with the command '' addToShoppingCart''. The results are received via the command '' showShoppingCart''and shown to the customer by ''showShoppingCart''.

OS_AddShoppingCart (S03b) : When the machine receives the command '' addToShoppingCart'', the Product are adding with the command ''Add_ToShoppingCart'' and ''putProductsUpToFiveUnits'' received as the data ''currentContentsOfShoppingCart''. The results are returned via the command '' showShoppingCart''.

ShoppingCart (S03c) : : After receiving the command ''Add_ToShoppingCart'' and ''putProductsUpToFiveUnits'' the results are returned as the data ''currentContentsOfShoppingCart''.

Correctness condition (S03a) Λ (S03b) Λ (S03c) =⇒ (R03)

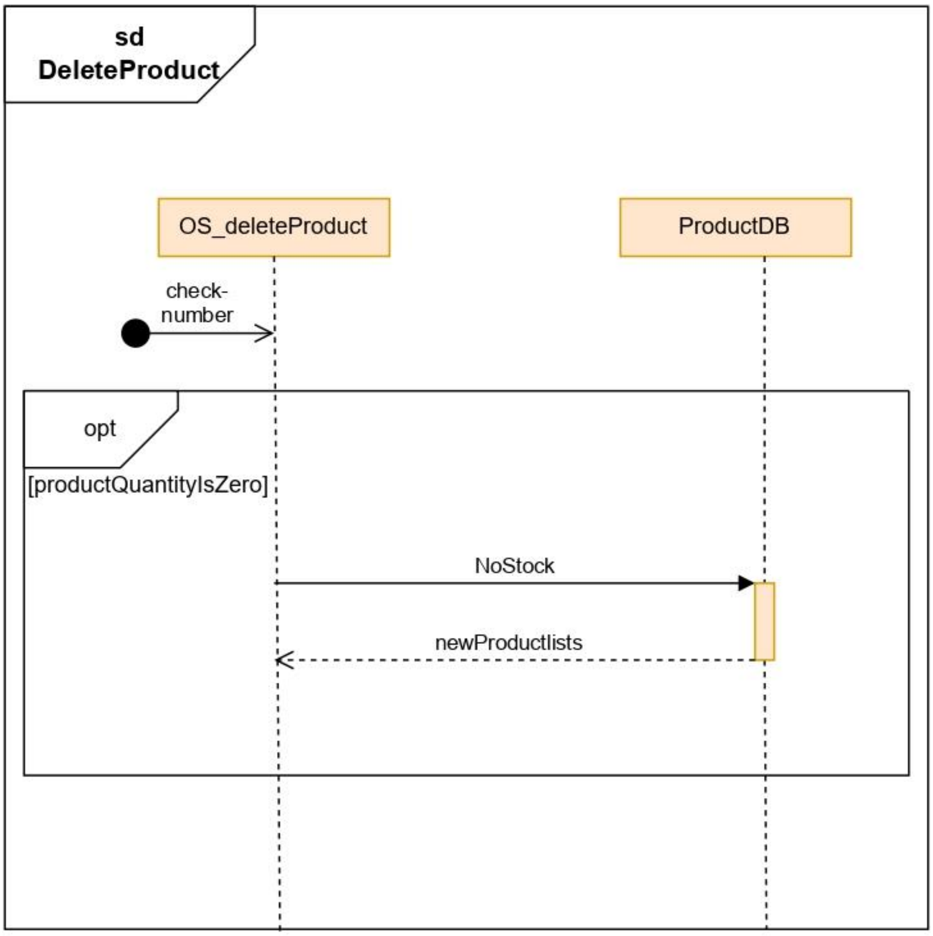


Specification and Sequence Diagram for OS_deleteProduct:

R04 : If a product no stock on the market, then an automated process should remove this product.
Employees (A05) : It should be possible for the employees to create new products by specifying the data, these will then be added to the list of the assortment.
OS_deleteProduct (S04a) : When receiving the command ‘’ no stock’’, a product is automatically removed using the command ‘’ removingProduct’’.

productDatabase (S04b) : When the command "removingProduct" is received, a product that no stock on the market is automatically removed.

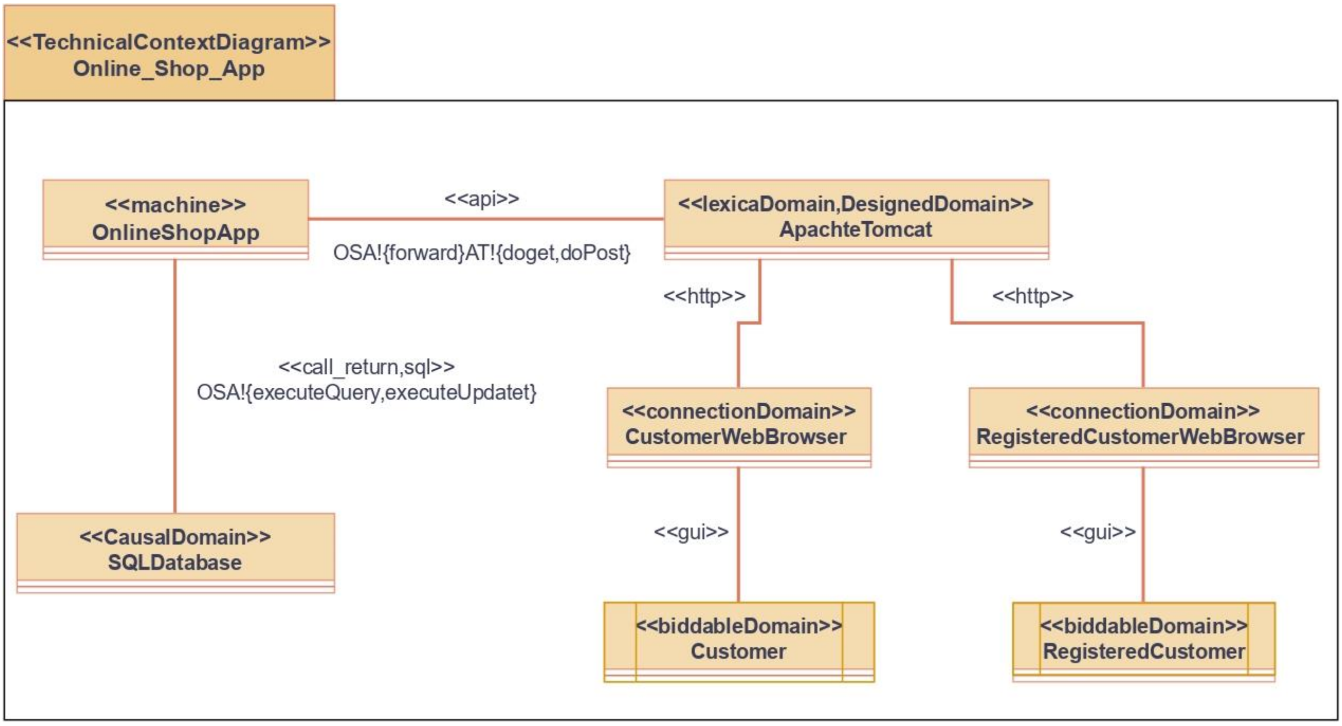
Correctness condition (A05) Λ (S04a) Λ (S04b) =⇒ (R04)



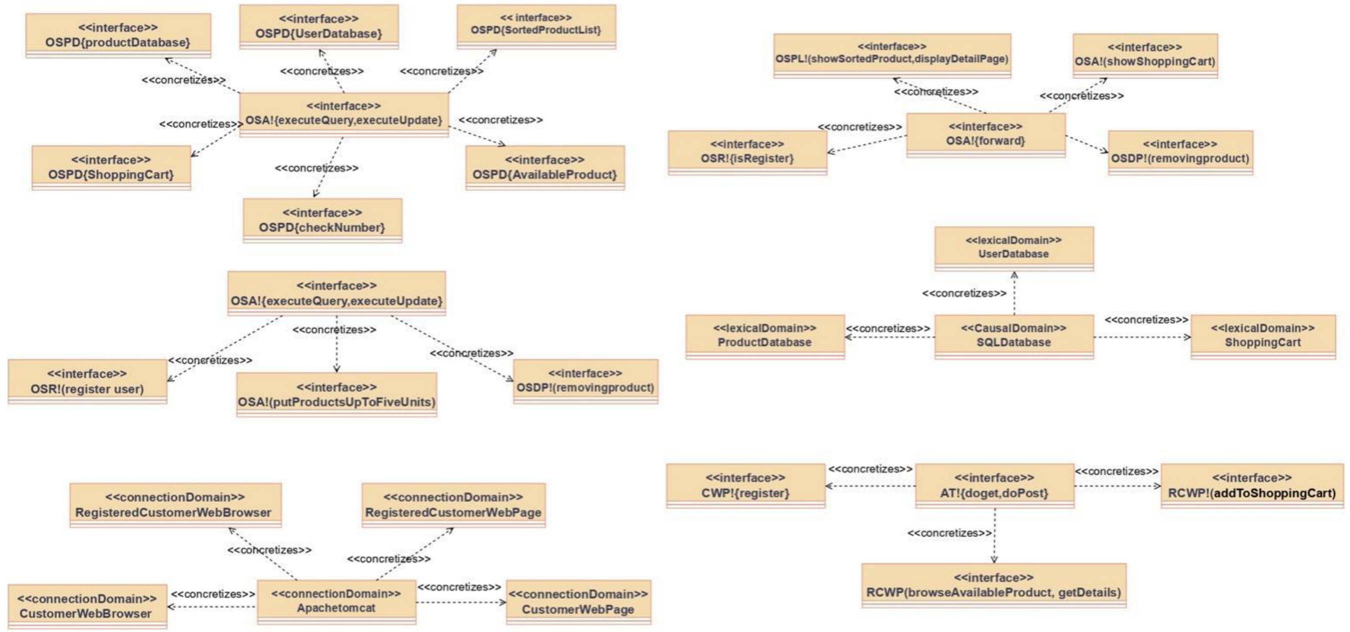
ShoppingCart, ProductDatabase, UserDatabase: Realized as SQLDatabase on the same computer as the machine. Therefore, the database is connected by a call-and-return interface and used with SQL commands.

CustomerWebPage: Realized using ApacheTomcat and CustomerWebBrowser (browser of Customer) We decide to use ApacheTomcat as a server platform, because the customer realized other projects on this platform and requires a Java implementation.

RegisteredCustomerrWebPage: Realized using ApacheTomcat and RegisteredCustomerrWebBrowser (browser of RegisteredCustomerr) We decide to use ApacheTomcat as a server platform, because the customer realized other projects on this platform and requires a Java implementation.



Mapping Diagram



<<interface>> OSPD{productDatabase}
<<interface>> OSPD{UserDatabase}
<< interface>> OSPD{SortedProductList}
<<concretizes>>
<<concretizes>>
<<concretizes>>
<<concretizes>>
<<interface>> OSA!{executeQuery,executeUpdate}
<<concretizes>>
<<interface>> OSPD{ShoppingCart}
<<concretizes>>
<<interface>> OSPD{AvailableProduct}
<<interface>> OSPD{checkNumber}
<<interface>> OSPL!(showSortedProduct,displayDetailPage)
<<interface>> OSA!(showShoppingCart)
<<concretizes>>
<<concretizes>>
<<concretizes>>
<<interface>> OSR!{isRegister}
<<interface>> OSA!{forward}
<<interface>> OSDP!(removingproduct)
<<interface>> OSA!{executeQuery,executeUpdate}
<<concretizes>>
<<concretizes>>
<<concretizes>>
<<interface>> OSR!(register user)
<<interface>> OSA!(putProductsUpToFiveUnits)
<<interface>> OSDP!(removingproduct)
<<lexicalDomain>> UserDatabase
<<concretizes>>
<<lexicalDomain>> ProductDatabase
<<concretizes>>
<<CausalDomain>> SQLDatabase
<<concretizes>>
<<lexicalDomain>> ShoppingCart
<<connectionDomain>> RegisteredCustomerWebBrowser
<<connectionDomain>> RegisteredCustomerWebPage
<<concretizes>>
<<concretizes>>
<<connectionDomain>> CustomerWebBrowser
<<concretizes>>
<<connectionDomain>> Apachetomcat
<<concretizes>>
<<connectionDomain>> CustomerWebPage
<<interface>> CWP!{register}
<<concretizes>>
<<interface>> AT!{doget,doPost}
<<concretizes>>
<<interface>> RCWP!(addToShoppingCart)
<<concretizes>>
<<interface>> RCWP(browseAvailableProduct, getDetails)

Technical interfaces of the machine:

SQL Commands:
defined in FIPS PUB 127-2, (U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology, 1993) Operations executeQuery and executeUpdate are defined in interface java.sql.Statement (https://docs.oracle.com/javase/8/docs/api/index.html?java/sql/Statement.html)
API for ApacheTomcat:
(http: //tomcat.apache.org/tomcat-9.0-doc/index.html)
Operations doGet and doPost are defined in abstract class javax.servlet.http.HttpServlet (https://docs.oracle.com/javaee/7/api/javax/ servlet/http/HttpServlet.html)
Operation forward defined in interface javax.servlet.RequestDispatcher (http://docs.oracle.com/javaee/7/api/javax/servlet/RequestDispatcher.html)

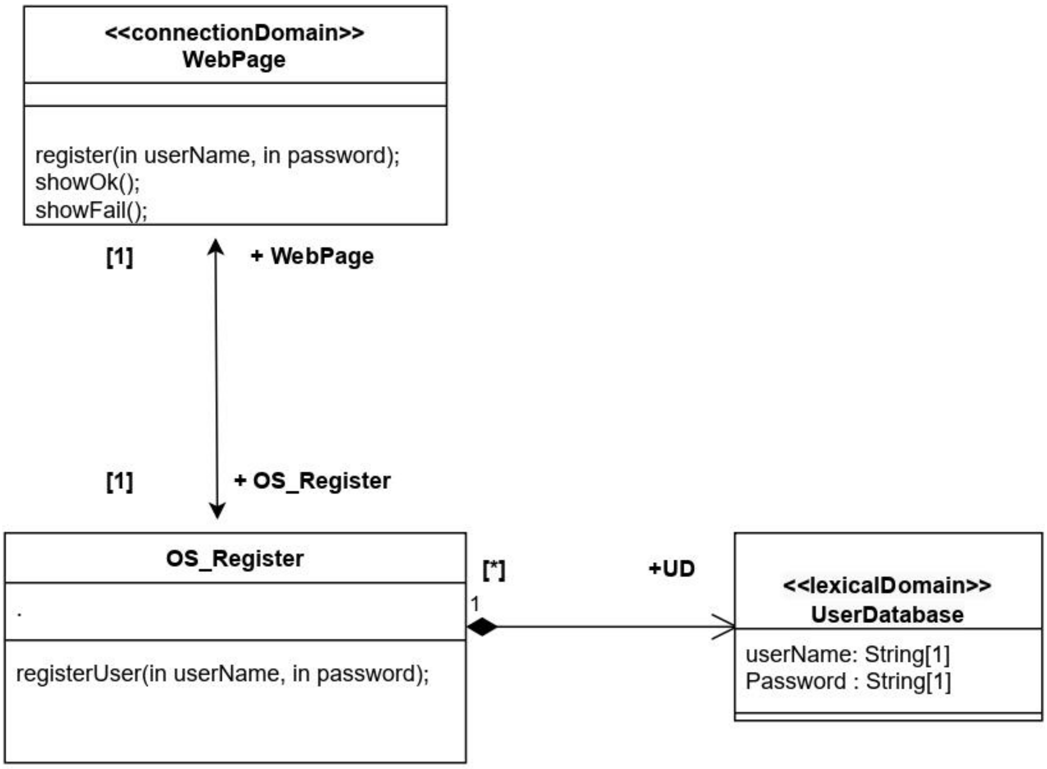
Technical interfaces in the environment:

HTTP (Hypertext Transfer Protocol):
defined in RFC 2616, (Network Working Group, 1999)
GUI:
User interfaces of HTML webpages (defined by https://www.w3.org/TR/html5/) presented by WebBrowser.

A5

The Operation Customer Registration

Class diagram:



OCL Expressions for Register Customers:

Context UserDatabaseOCL

Invariant:

```
Context UserDatabase (username must be unique)
inv : UserDatabase.allInstances() -> isUnique(username)
```

Operation Specification 1:

Name : register
Description : Forwards the register request from the Person to the machine.

```
Context PersonWebPage :: register(email : String ,username : String)
pre : true
post : OS_register ^ registerUser(email, username)
```

Operation Specification 2:

Name : usernameExists
Description : checks if the user doesn’t exist in the UserDataBase.

```
context UsersDatabase :: usernameExists(username : String) : boolean
pre : true
post:
if userdatabase@pre -> exists(u:UserDatabase | u.username = username)
then result = true
else
    result = false
endif
```

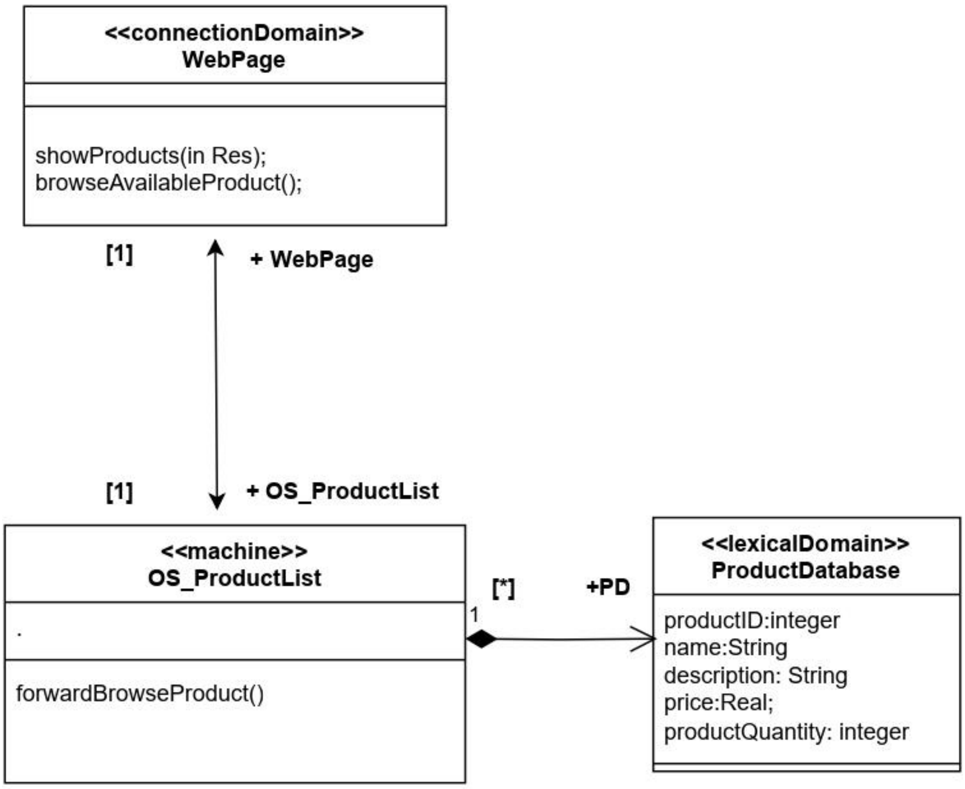
Operation Specification 3:

Name : registerUser
Description : Forward the register request to the UsersDatabase if the Userdata is valid.

```
Context OS_Register :: registerUser(email : String, username : String)
pre : true
post :
if not usernameExists(user)
then
    userdatabase -> one(u: UserDatabase |
    u.email = email
    and u.username = username)
    and
    userdatabase -> size() = userdatabase@pre -> size() + 1
    and
    PersonWebPage^showOk()
else
    PersonWebPage^showFail()
endif
```

The Operation View Products List

Class diagram:



OCL Expressions for view Products:

Operation Specification 1:

Name: browseproducts
Description: sends a browse request to the machine

Context: RegisteredUserWebPage :: browseProducts()
Pre: true
Post: OS_ProductList^forwardBrowseProduct()

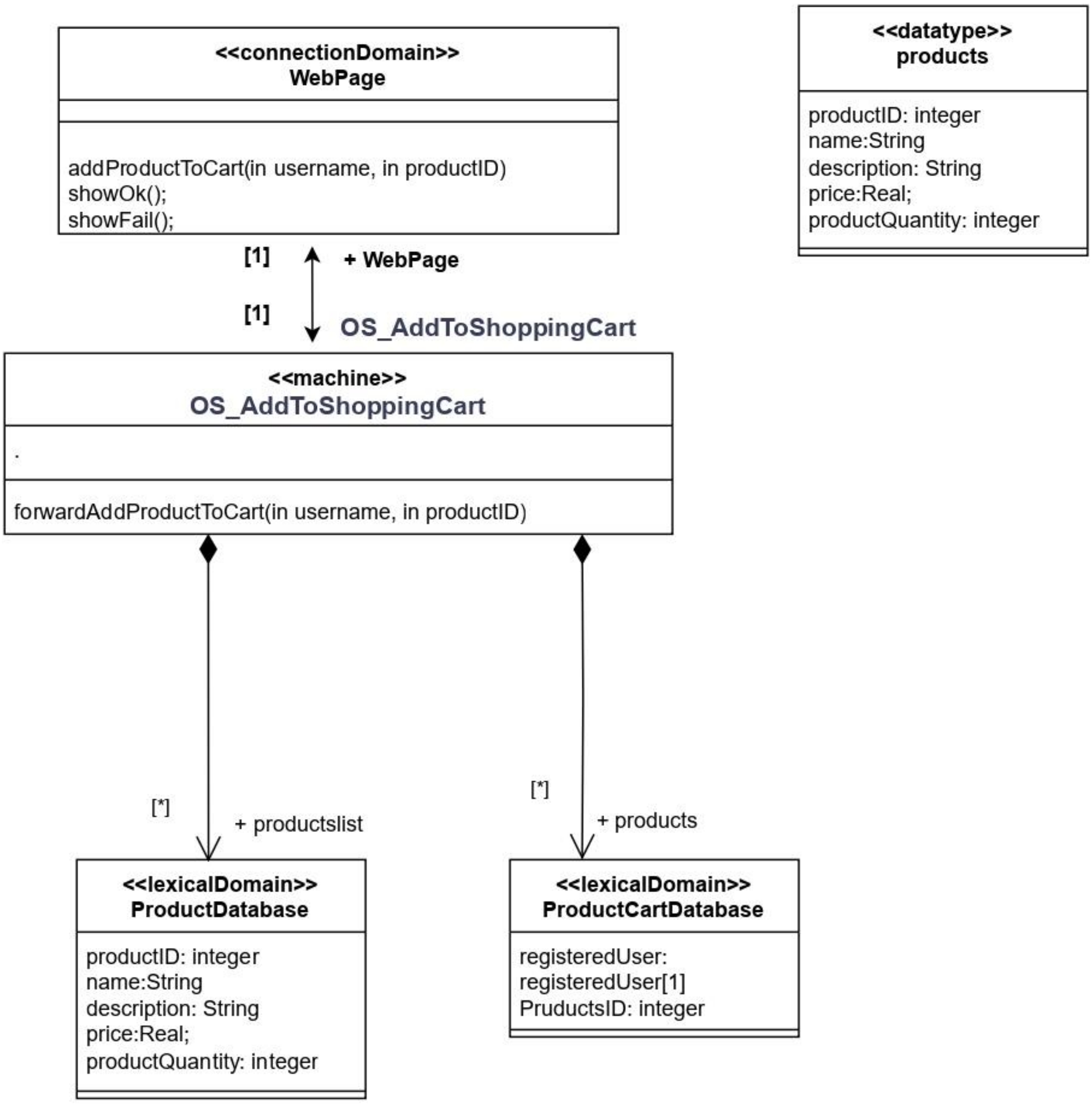
Operation Specification 2:

Name: forwardBrowseProduct
Description: forwards the browseProducts request from the machine to the database

Context: OS_ProductList :: forwardBrowseProduct() : Set(product)
Pre: true
Post: let
res = Set(ProductDatabase) = productsdatabase -> asSet()
In
RegisteredUserWebpage^showproducts(res)

The Operation Add To Shopping Cart

Class diagram:



OCL Expressions for Add to shopping cart:

Invariant:

Context ProductDatabase
addProductToProductCart

Context UserDatabaseOCL

Invariant:

```
Context UserDatabase (username and ProductID must be unique)
inv : UserDatabase.allInstances() -> isUnique(username,productID)
```

Operation Specification 1:

Name : addProductToCart
Description : send add Product To Cart request to the machine.

```
context WepPage :: addProductToCart (userName :String , productID:integer)
pre : true
post :
 OS_AddToShoppingCart ^ forwardAddProductToCart(userName :String , productID:integer)
```

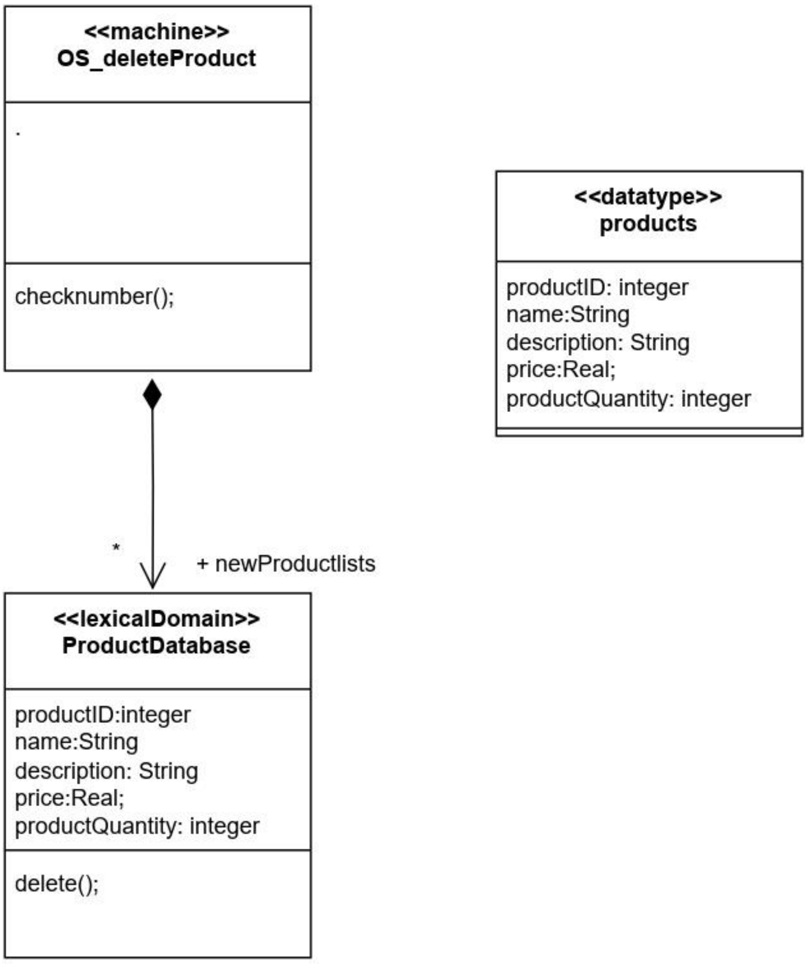
Operation Specification 2:

Name : forwardAddProductToCart
Description : add product to the products in User productslist.

```
context OS_AddToShoppingCart::forwardAddProductToCart(userName:String , productID:Integer)
pre: productslist->one(p : ProductDatabase | p.productID = productID)
post:
if products@pre->select(pc : ProductCartDatabase |
pc.productID = productID and pc.userName = userName)->size() < 5
then
products -> exists(pc: ProductCartDatabase | pc.userName = userName
and pc.productID = productID)
and products -> size() = products@pre->size() + 1
and WebPage^showOk()
else
WebPage^showFail()
endIf
```

The Operation Delete Product

Class diagram:



OCL Expressions for Delete product:

```
context OS_deleteProduct :: checknumber ()
pre : true
post :
let productsToBeDeleted : Set(product) =
select(newproduct.productQuantity <= 0)–>asSet()
```

Life Cycle

$LC_{Customer}$: (registeredCustomer | ViewProducts)

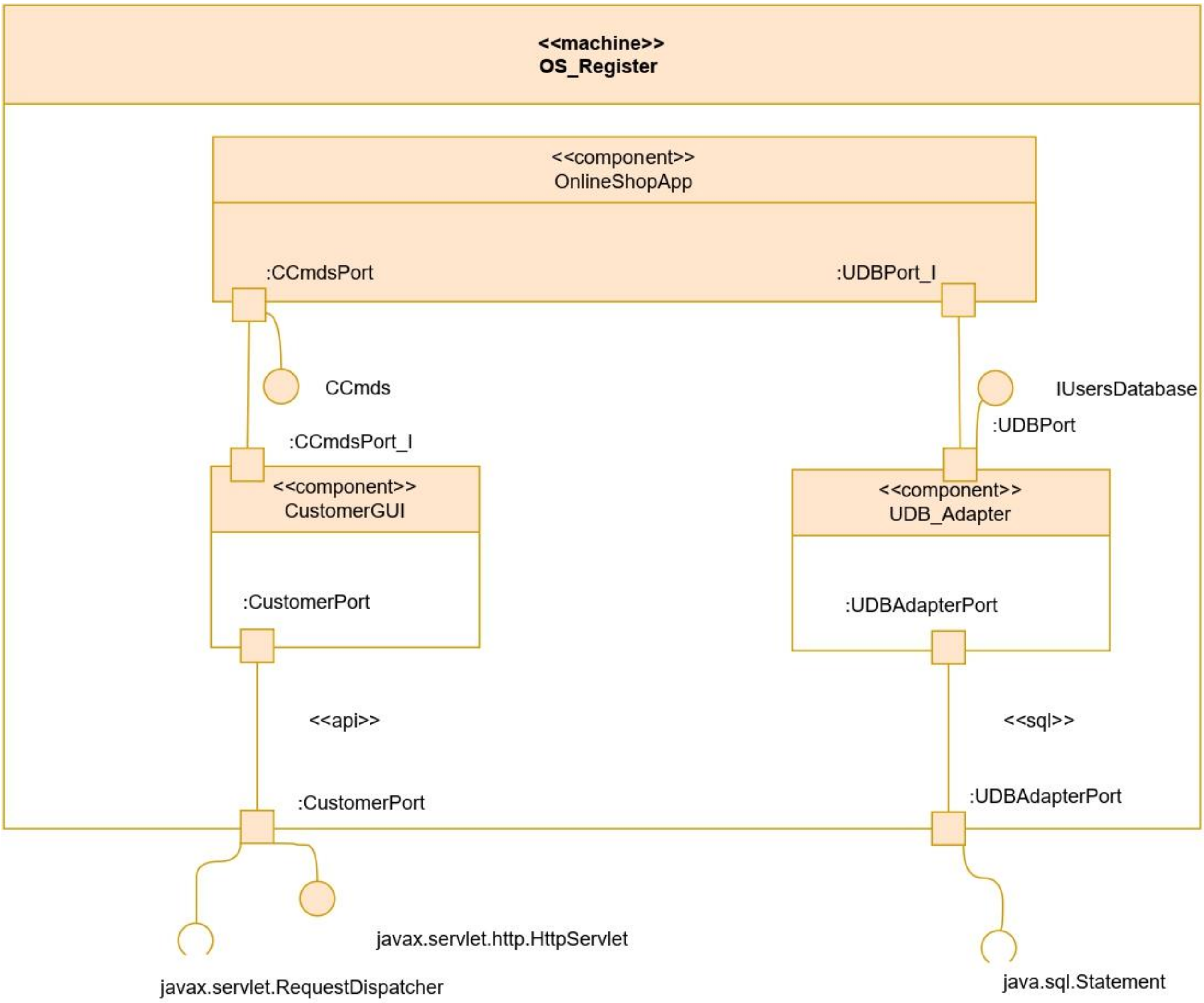
$LC_{\mathrm{R}egistered\mathrm{Customer}}$: viewProducts^{+};[AddToShoppingCart]*

$LC_{\mathrm{Customer\ OnlineMarketAppr}}$: $(||_{\mathrm{i=1}}^{n} LC_{\mathrm{R}egistered\mathrm{Customeri}})$ || $(||_{\mathrm{j=1}}^{m} LC_{Customer_{j}})$
||DeleteProducts*

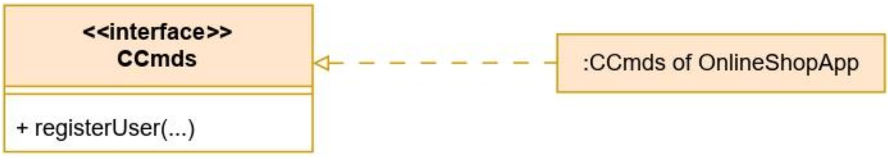
D1

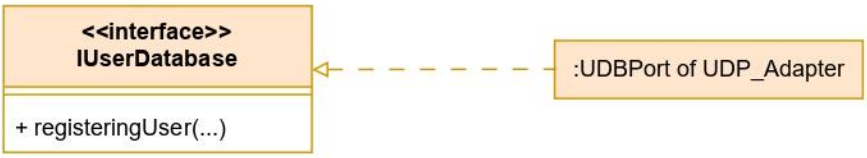
CustomerRegistration

Architecture:

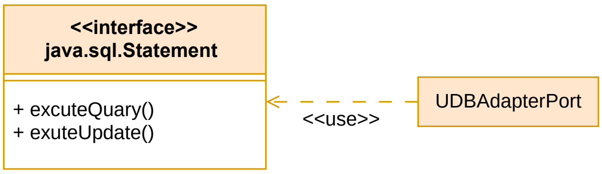


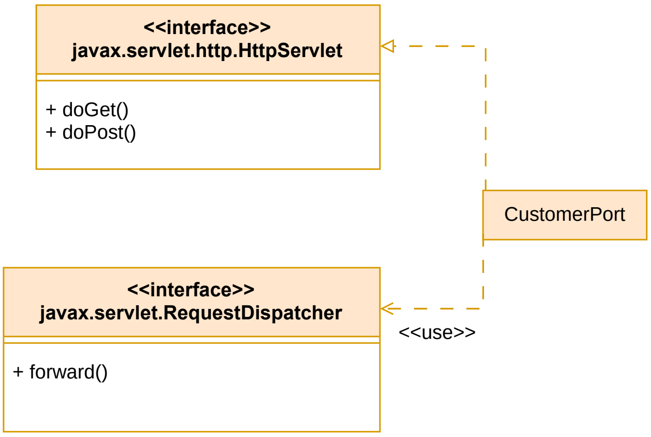
Internal Interfaces in:

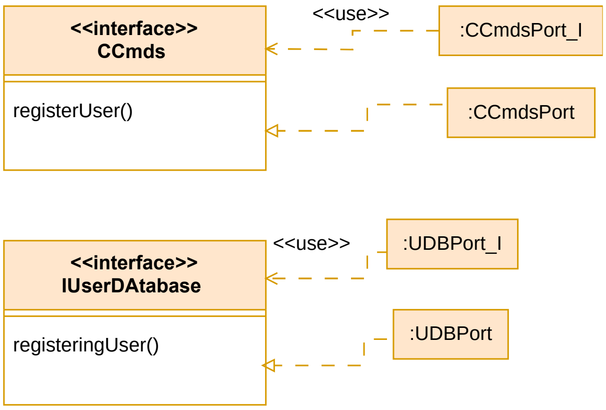




Port types and interface relations for:

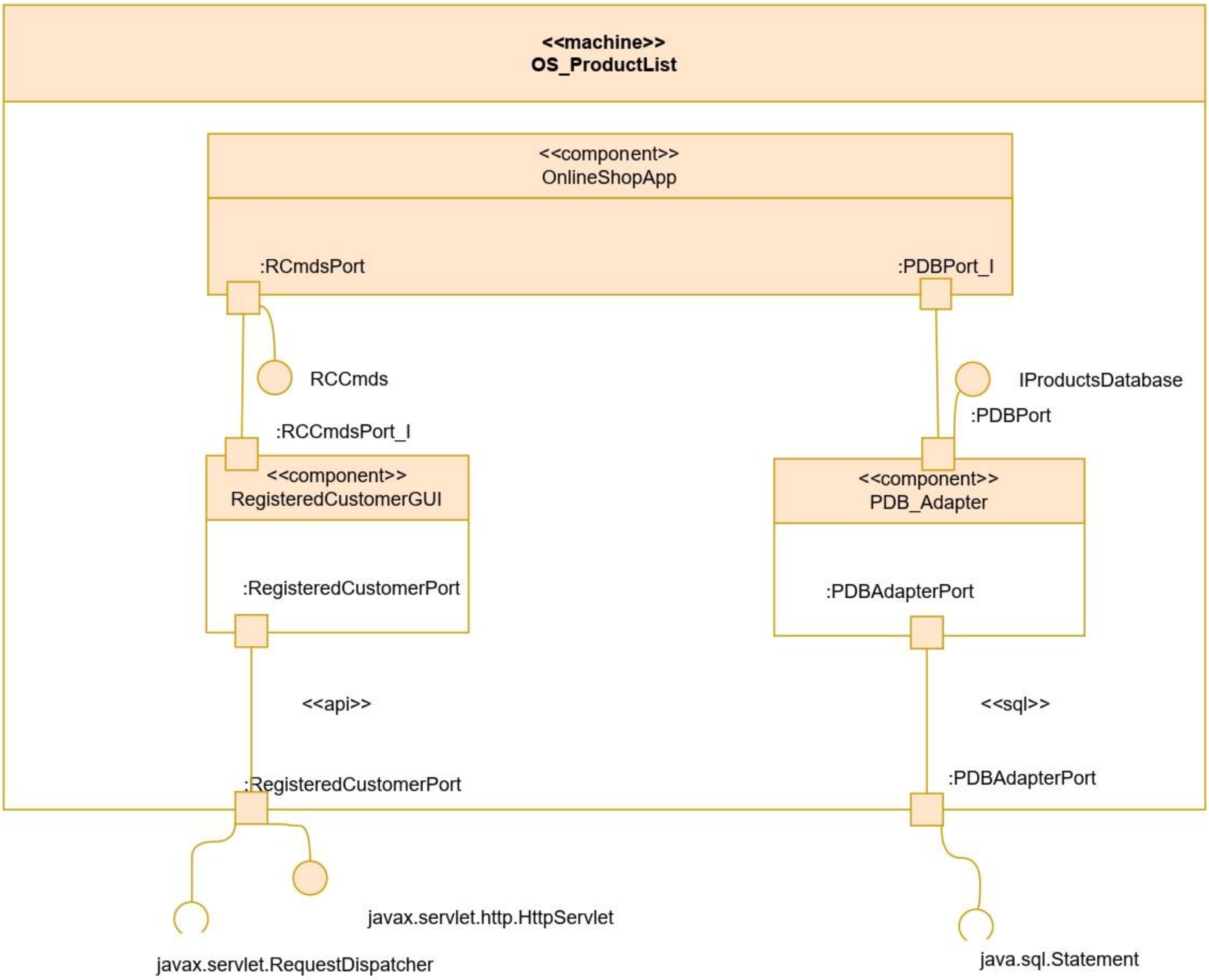




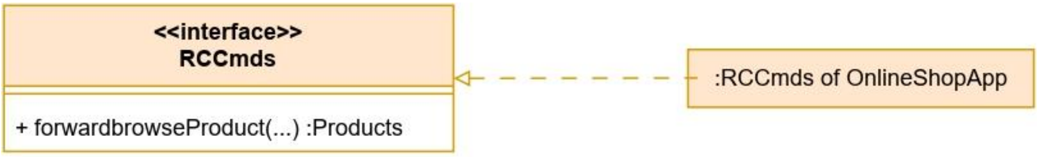


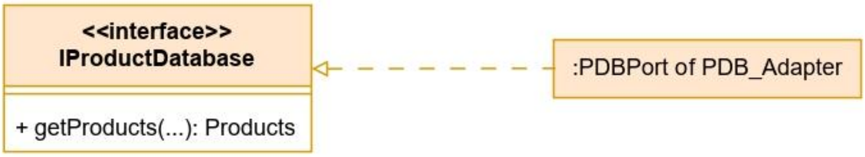
ViewProductList

Architecture:

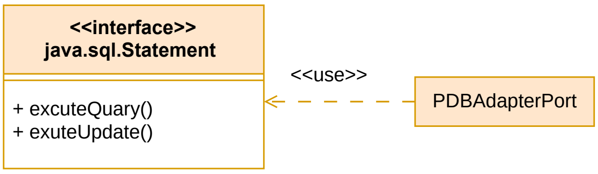


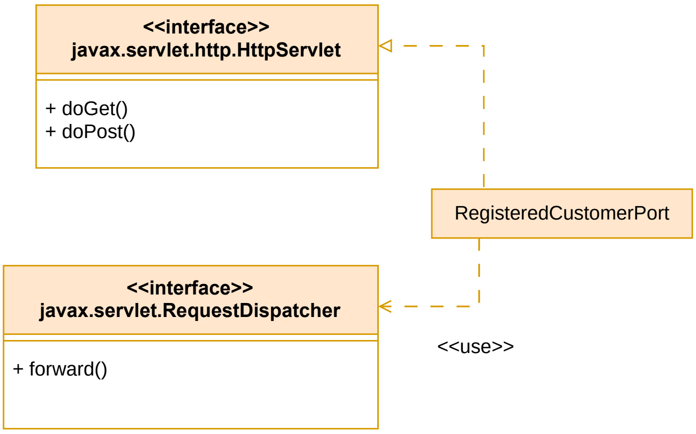
Internal Interfaces in:

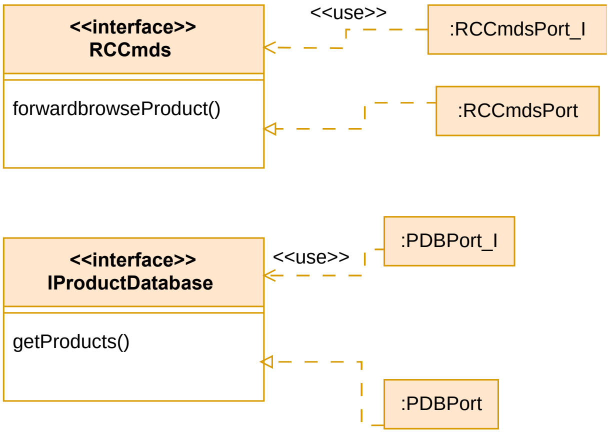




Port types and interface relations for:

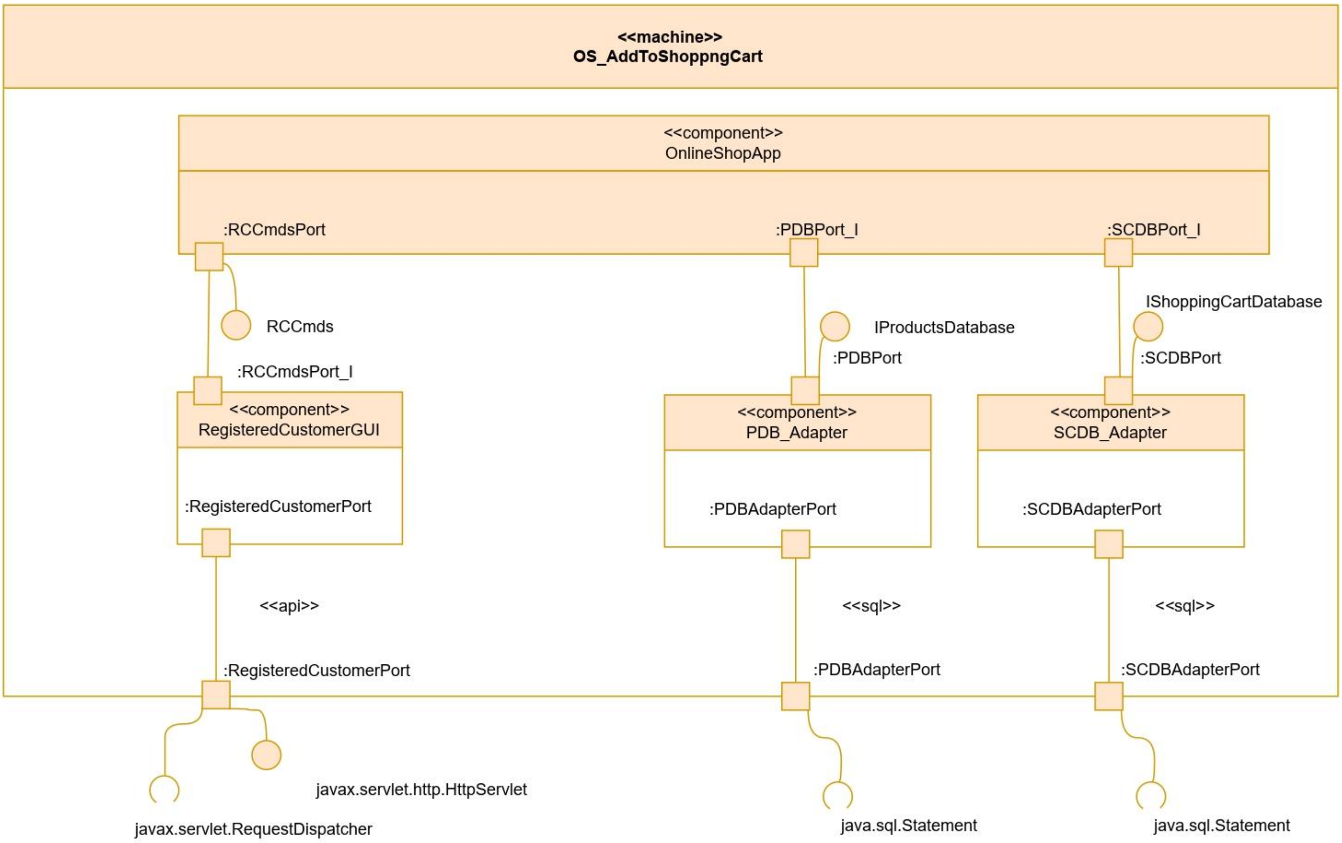




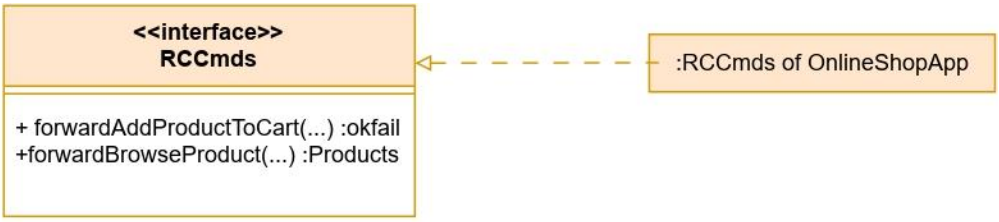


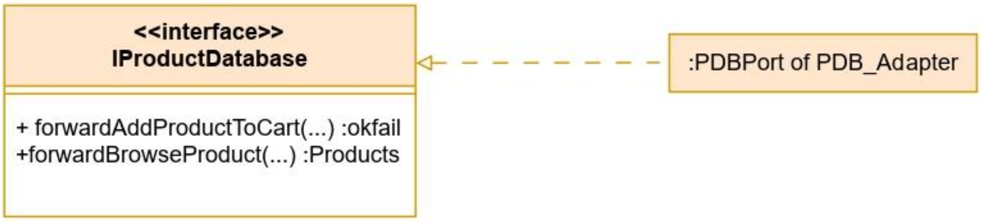
AddToShoppingCart

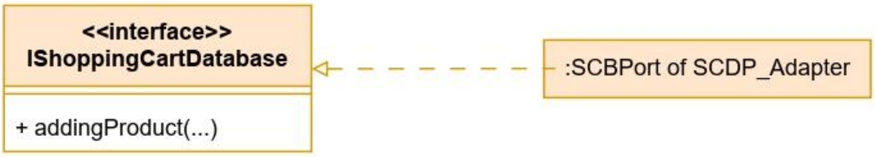
Architecture:



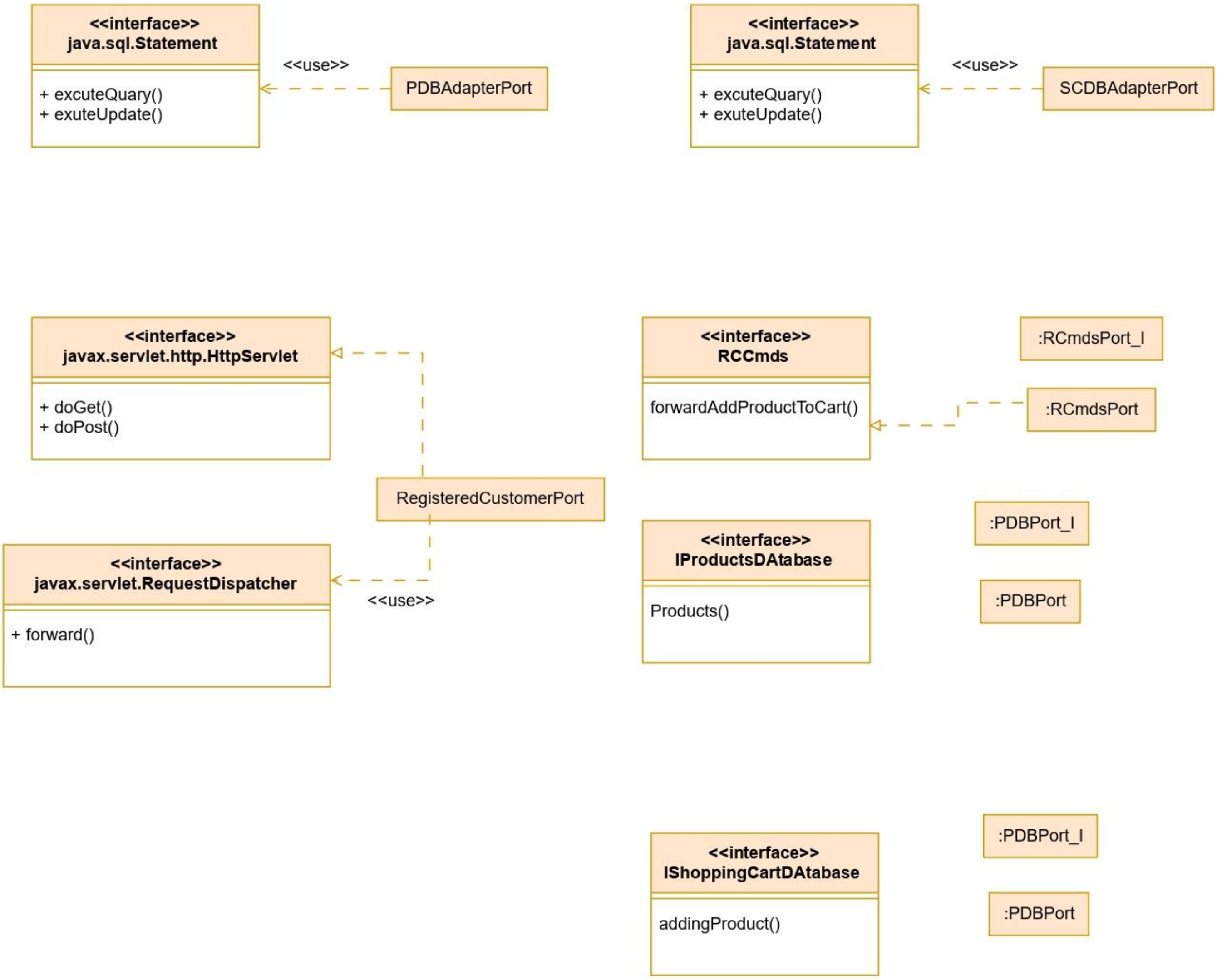
Internal Interfaces in:


<<interface>>
RCCmds
+ forwardAddProductToCart(...) :okfail
+forwardBrowseProduct(...) :Products
:RCCmds of OnlineShopApp


<<interface>>
IProductDatabase
+ forwardAddProductToCart(...) :okfail
+forwardBrowseProduct(...) :Products
:PDBPort of PDB_Adapter

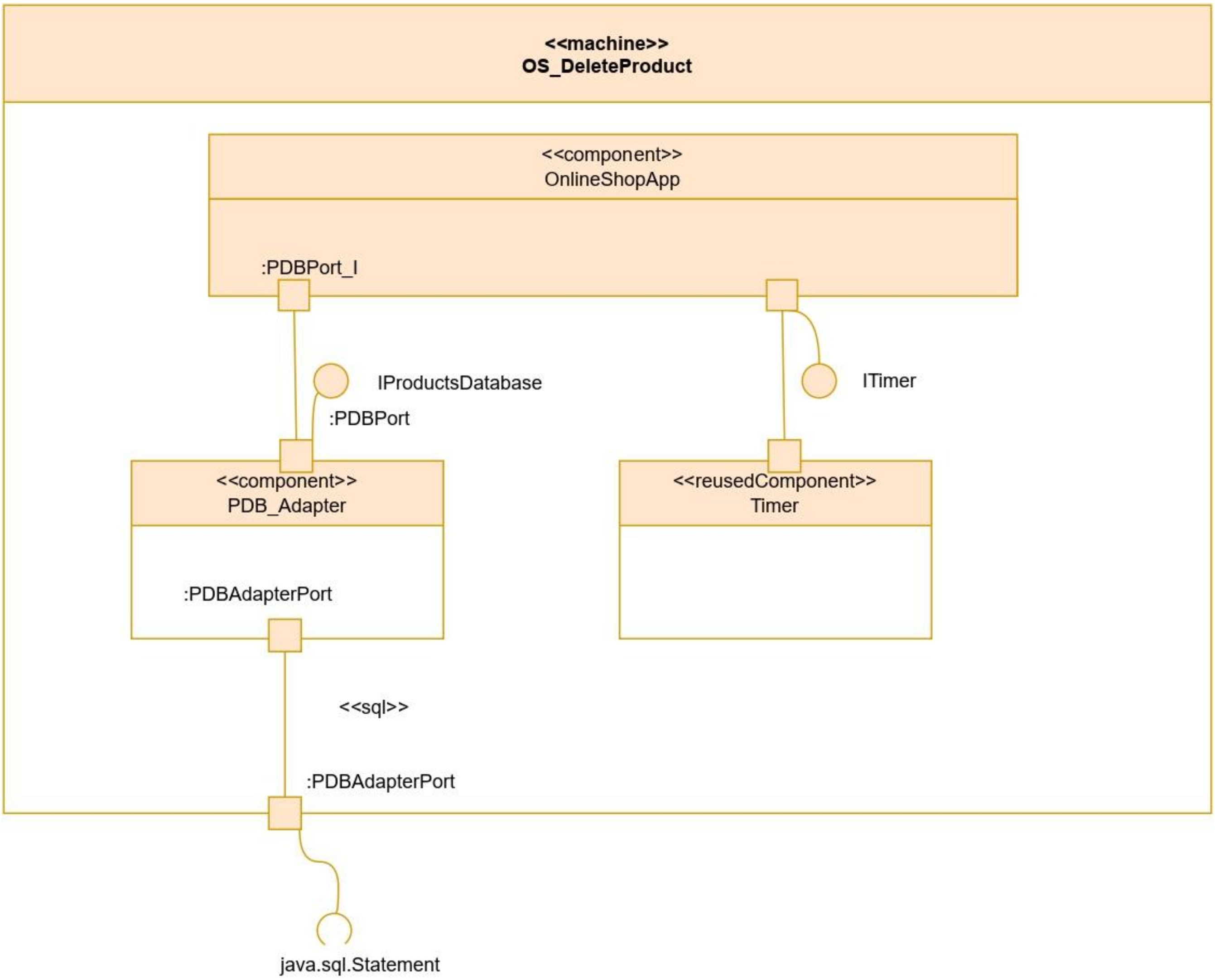


Port types and interface relations for:

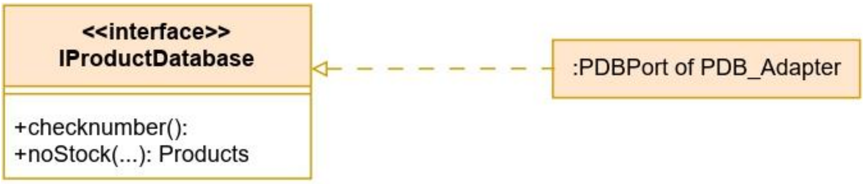


DeleteProduct

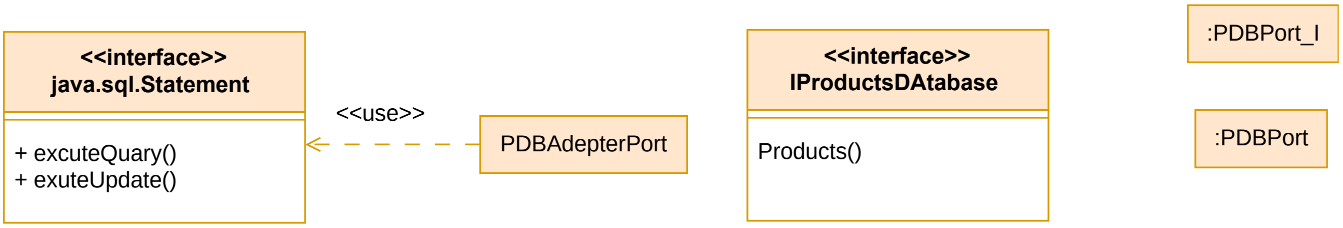
Architecture:



Internal Interfaces in:

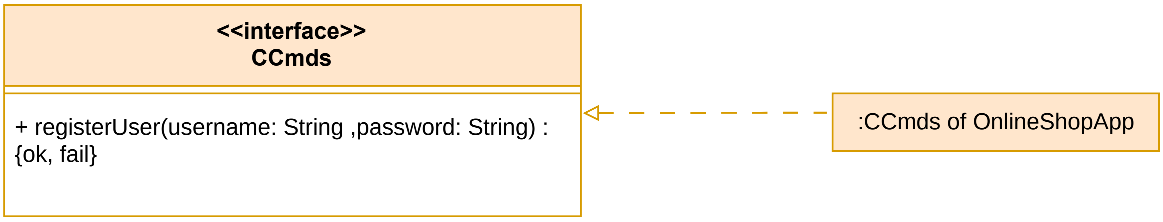


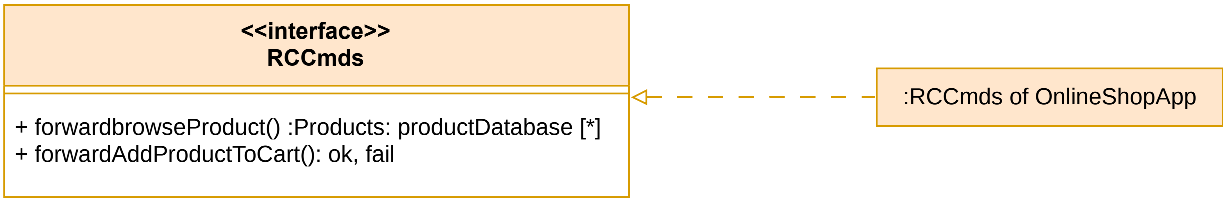
Port types and interface relations for:

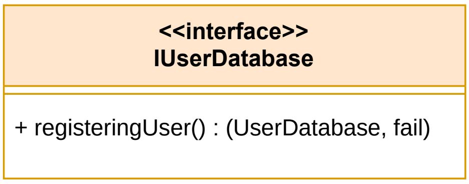


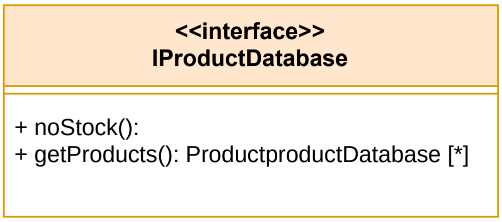
Refinment

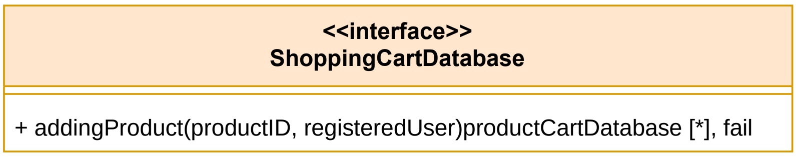
Refining app_if


<<interface>>
CCmds
+ registerUser(username: String ,password: String) : {ok, fail}
:CCmds of OnlineShopApp


<<interface>>
RCCmds
+ forwardbrowseProduct() :Products: productDatabase [*]
+ forwardAddProductToCart(): ok, fail
:RCCmds of OnlineShopApp






<<interface>>
ShoppingCartDatabase
+ addingProduct(productID, registeredUser)productCartDatabase [*], fail

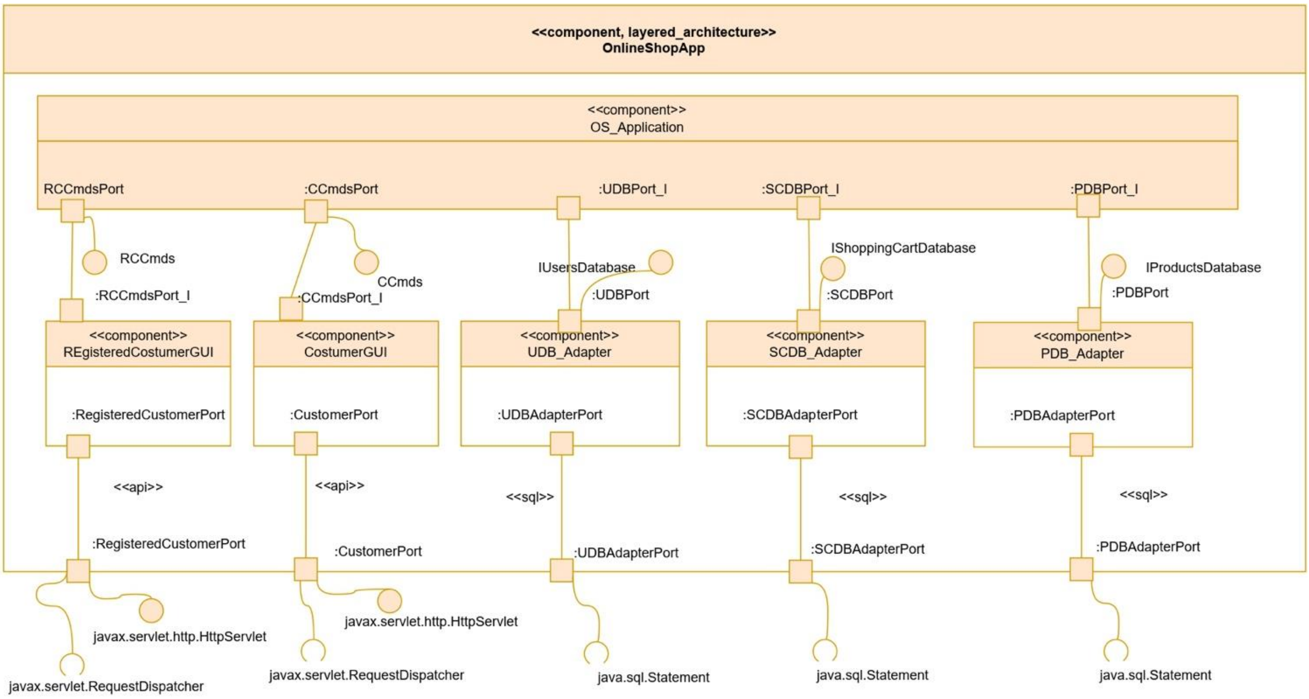
Refining tech_if

Considered interface insubproblem architecture	Technical interface
<<api>> javax.servlet.http.HttpServlet Register	<<api>> AT!{doGet,doPost}
<<api>> javax.servlet.http.HttpServlet ProductList	<<api>> AT!{doGet,doPost}
<<api>> javax.servlet.http.HttpServlet addToShoppingcart	<<api>> AT!{doGet,doPost}
<<api>> javax.servlet.RequestDispatcher in Register	<<api>> OS!{forward}
<<api>> javax.servlet.RequestDispatcher in ProductList	<<api>> OS!{forward}
<<api>> javax.servlet.RequestDispatcher in addToShoppingcart	<<api>> OS!{forward}
<<sql>> java.sql.Statement in Register	<<call_return.sql>> OS!{executeQuery,executeUpdate}
<<sql>> java.sql.Statement in ProductList	<<call_return.sql>> OS!{executeQuery,executeUpdate}
<<sql>> java.sql.Statement in addToShoppingcart	<<call_return.sql>> OS!{executeQuery,executeUpdate}
<<sql>> java.sql.Statement in deleteProduct	<<call_return.sql>> OS!{executeQuery,executeUpdate}

Refining adapter_if

There are no HAL components in the subproblem architechtures. Hence, there are no adapter if' interface classes that need to be refined.

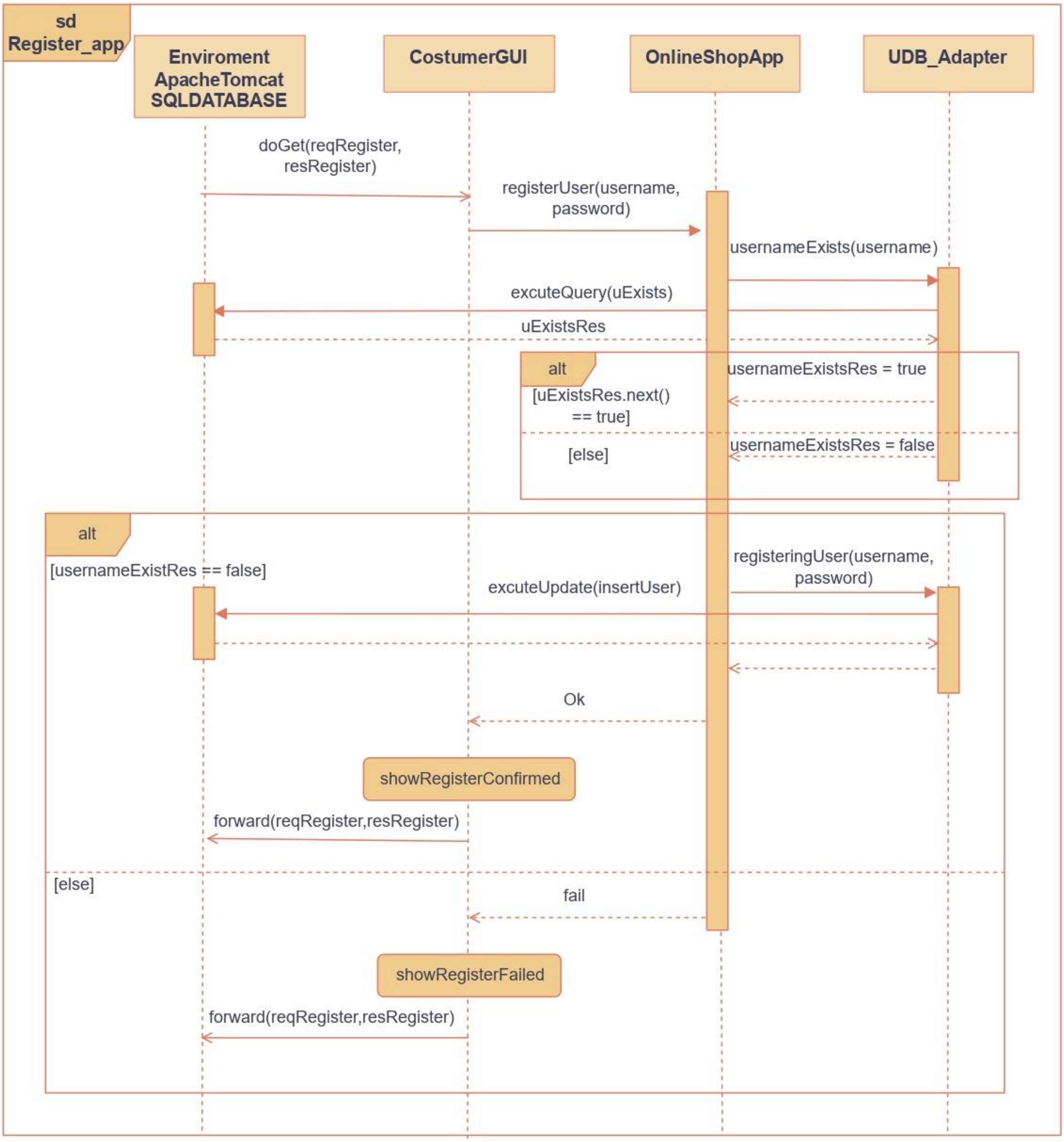
Merged architecture



D2

CustomerRegistration

Inter-component diagram


sd Register_app
Enviroment ApacheTomcat SQLDATABASE
CostumerGUI
OnlineShopApp
UDB_Adapter
doGet(reqRegister, resRegister)
registerUser(username, password)
usernameExists(username)
excuteQuery(uExists)
uExistsRes
alt
[uExistsRes.next() == true]
usernameExistsRes = true
[else]
usernameExistsRes = false
alt
[usernameExistRes == false]
registeringUser(username, password)
excuteUpdate(insertUser)
Ok
showRegisterConfirmed
forward(reqRegister,resRegister)
[else]
fail
showRegisterFailed
forward(reqRegister,resRegister)

SQL

insertUser
INSERT INTO UserDatabase (username, password) VALUES (“username”, "password")

uExists
SELECT * FROM UserDatabase WHERE username = “username”

Remarks

reqRegister represent HTTPServletRequest

resRegister represent HttpServletResponse

The state predicate showRegisterConfirmed represents that the the confirmation is shown.

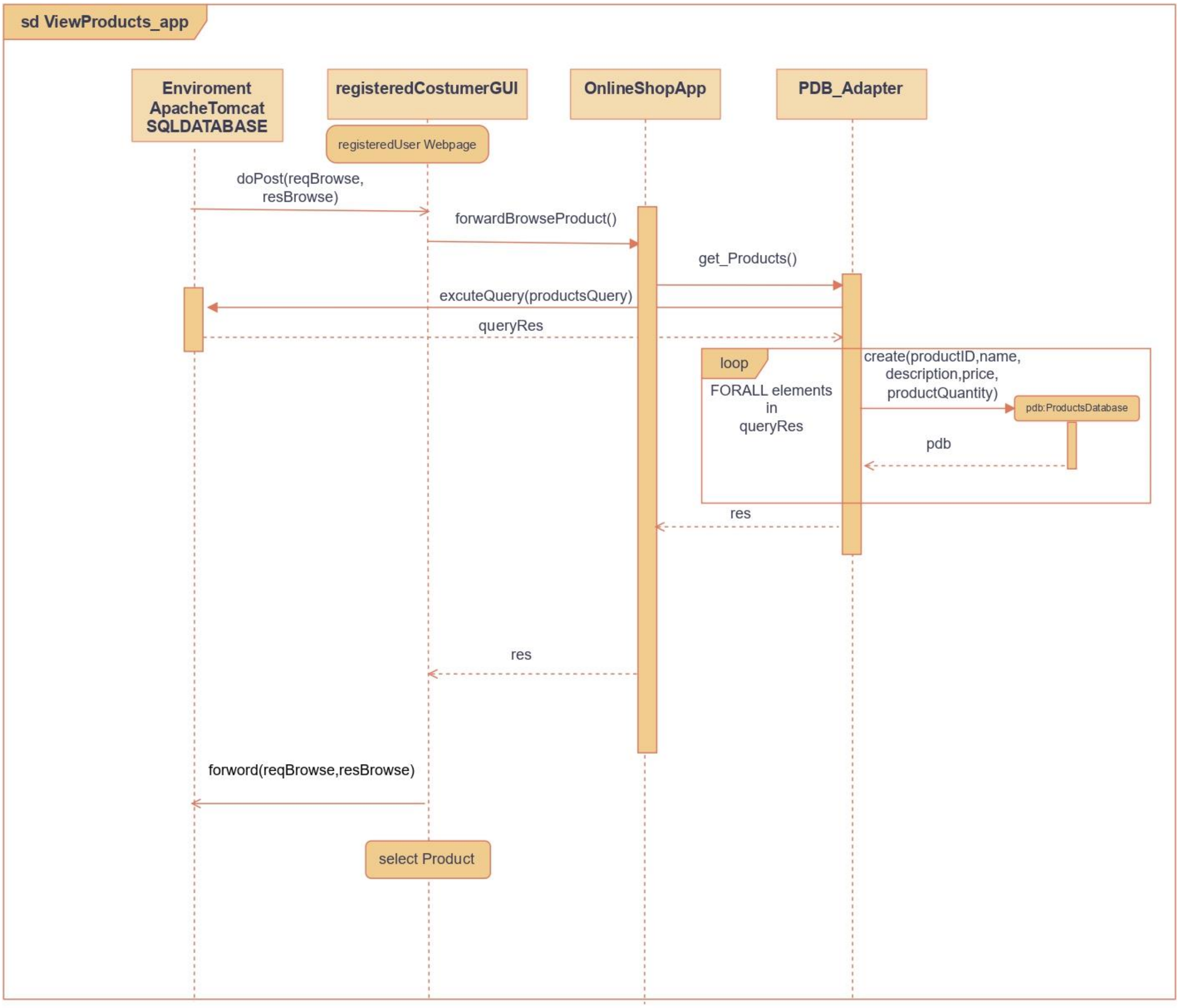
The state predicate showRegisterFailed represents that the the error message is shown.

forward(...) sends the request and response back to the

server to generate the HTML webpage.

ViewProductList

Inter-component diagram



sd ViewProducts_app
Enviroment ApacheTomcat SQLDATABASE
registeredCostumerGUI
OnlineShopApp
PDB_Adapter
registeredUser Webpage
doPost(reqBrowse, resBrowse)
forwardBrowseProduct()
get_Products()
excuteQuery(productsQuery)
queryRes
loop
FORALL elements in queryRes
create(productID,name, description,price, productQuantity)
pdb:ProductsDatabase
pdb
res
res
forword(reqBrowse,resBrowse)
select Product

productQuery

SELECT *

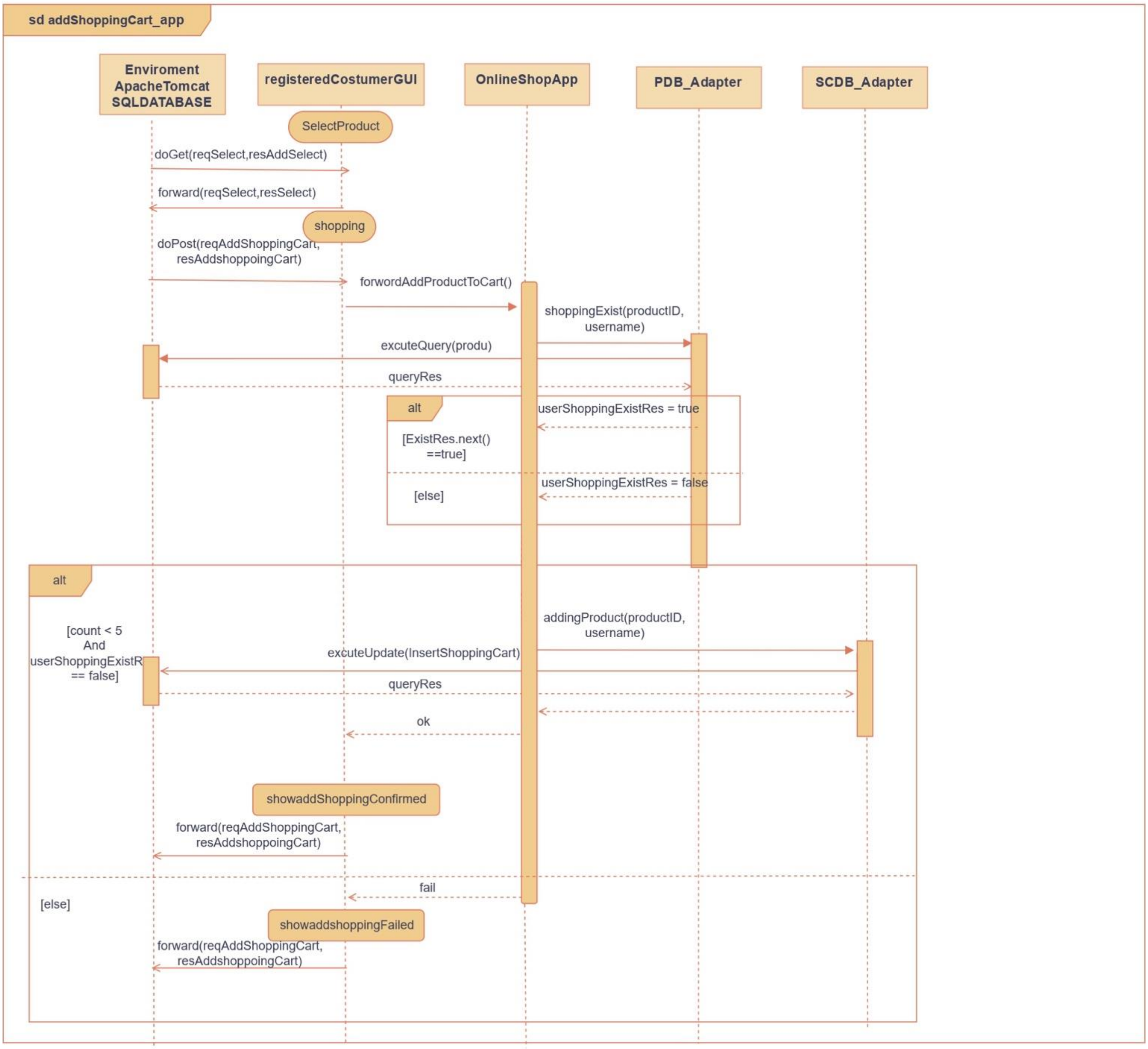
FROM ProductDatabase

ORDER BY price DESC;

AddToShoppingCart

Inter-component diagram

update this Diagramm


sd addShoppingCart_app
Enviroment ApacheTomcat SQLDATABASE
registeredCostumerGUI
OnlineShopApp
PDB_Adapter
SCDB_Adapter
SelectProduct
doGet(reqSelect,resAddSelect)
forward(reqSelect,resSelect)
shopping
doPost(reqAddShoppingCart, resAddshoppoingCart)
forwordAddProductToCart()
shoppingExist(productID, username)
excuteQuery(produ)
queryRes
alt
userShoppingExistRes = true
[ExistRes.next() ==true]
userShoppingExistRes = false
[else]
alt
addingProduct(productID, username)
[count < 5 And userShoppingExistR == false]
excuteUpdate(InsertShoppingCart)
queryRes
ok
showaddShoppingConfirmed
forward(reqAddShoppingCart, resAddshoppoingCart)
fail
[else]
showaddshoppingFailed
forward(reqAddShoppingCart, resAddshoppoingCart)

produ

```
SELECT p.*
FROM product p
JOIN shopping s ON p.productID = s.productID
WHERE s.username = 'username';
```

InsertShoppingCart

insert new row in Shopping Cart

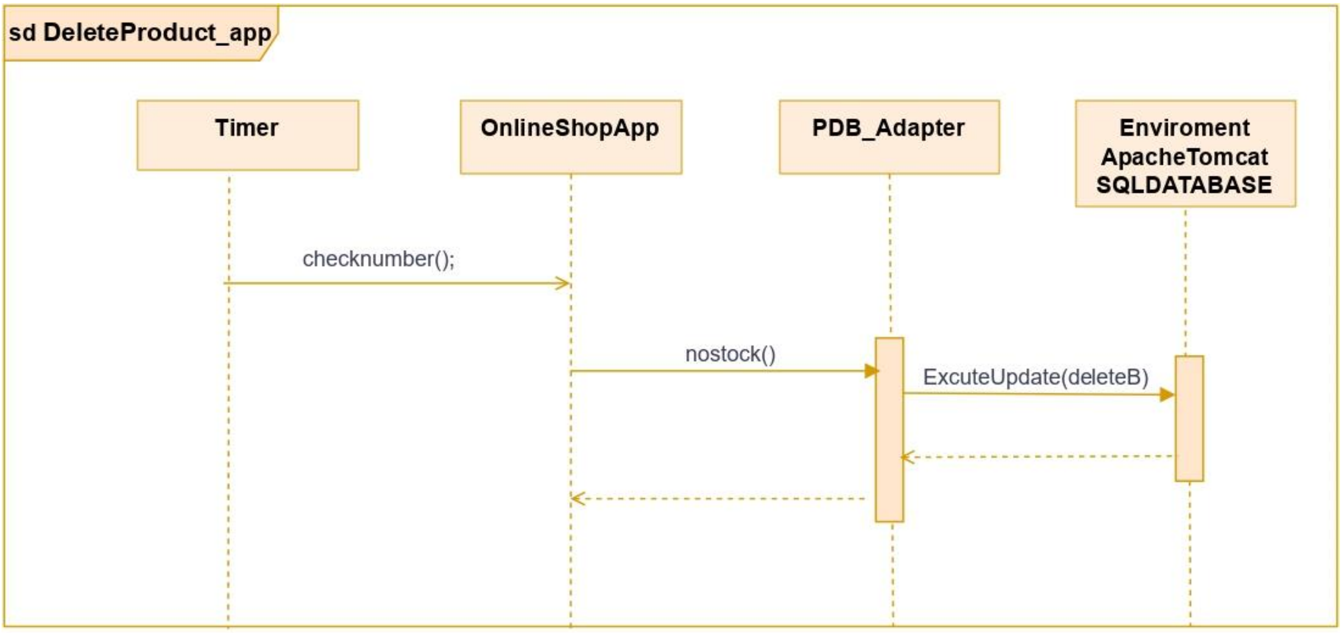
```
INSERT INTO shopping (username, productID)
VALUES ('username', " productID ");
```

trigger for count shopping and max product <= 5

```
CREATE TRIGGER enforce_max_products
BEFORE INSERT ON shopping

BEGIN
   SELECT COUNT(*)
   INTO product_count
   FROM shopping
   WHERE username = NEW.username And productID = NEW.productID;

   IF product_count >= 5 THEN
      RAISE EXCEPTION 'Cannot insert more than 5 products per user.';
   END IF;

   RETURN NEW;
END
```

DeleteProduct

Inter-component diagram



```
DELETE FROM shopping
WHERE productID IN (
    SELECT productID
    FROM product
    WHERE ProductQuantity = 0
);

DELETE FROM product
WHERE ProductQuantity = 0;
```

we have 3 Tabels (user , Product, shopping(shoppingCart))

```sql
CREATE TABLE product (
    productID INT PRIMARY KEY,
    name VARCHAR(255),
    description VARCHAR(255),
    price DECIMAL(10, 2),
    productQuantity INT
);


CREATE TABLE user (
    username VARCHAR(255) PRIMARY KEY,
    password VARCHAR(255)
);


CREATE TABLE shopping (
    username VARCHAR(255),
    productID INT,
    FOREIGN KEY (username) REFERENCES user(username),
    FOREIGN KEY (productID) REFERENCES product(productID)
);
```

D3

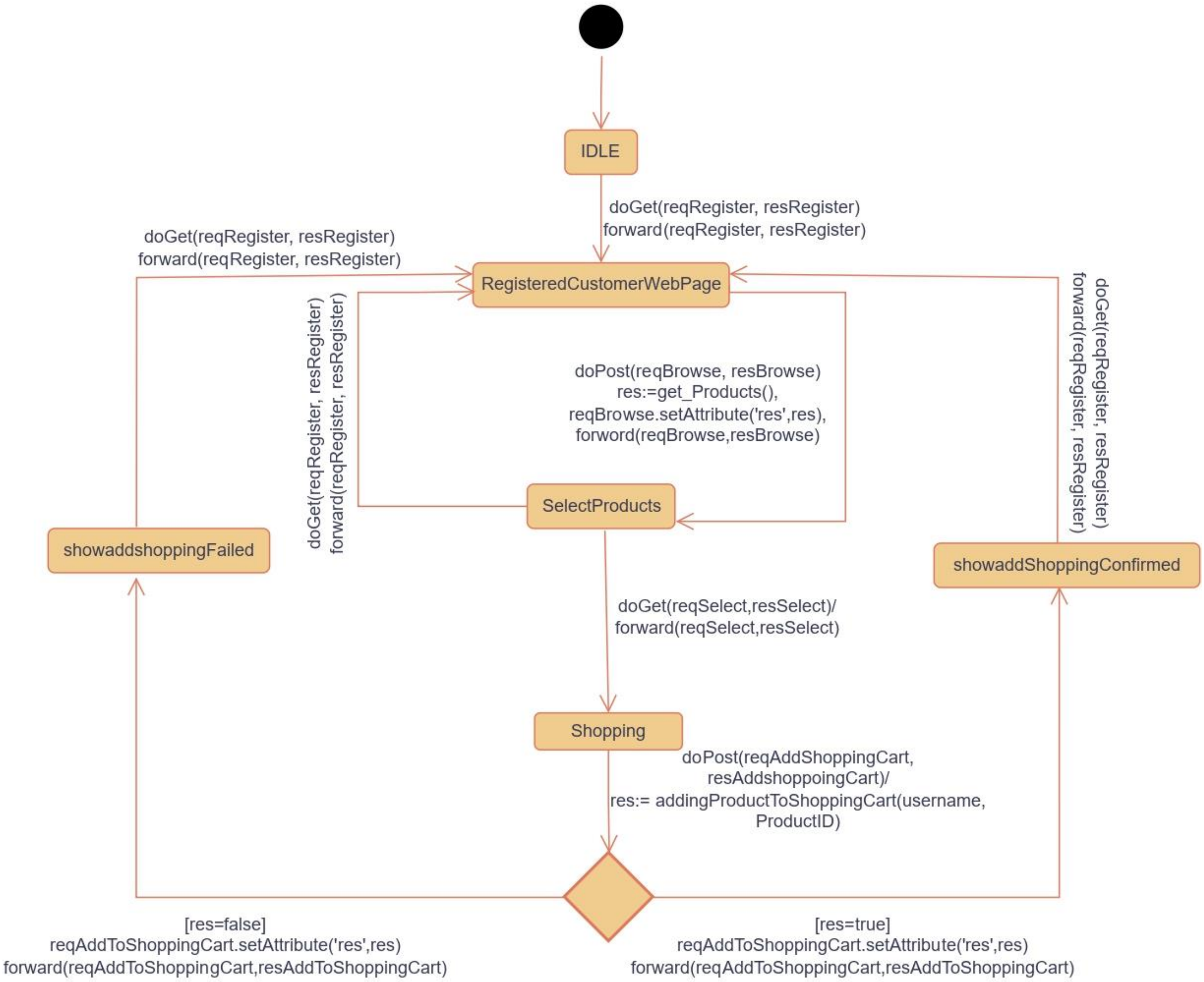
No further decomposition of the components defined in Step D1 is needed, because the components in our project are described in details in a very comprehensive way. The components are small enough that we will not need to define a set of smaller sub-components that perform the functionality of the D1 components.

D4

Check whether a state machine is necessary:

1. The component RegisteredCostumerGUI: no refinement exists in Step D3: Intra-Component Interaction; continue with looking at Step D2: Inter-Component Interaction. There are more than two states. Therefore, a state machine is required.
2. The component CostumerGUI: no refinement exists in Step D3: Intra-Component Interaction; continue with looking at Step D2: Inter-Component Interaction. There are exactly two states. Therefore, a state machine is not required.
3. The component PDB_Adapter: no refinement exists in Step D3: Intra-Component Interaction; continue with looking at Step D2: Inter-Component Interaction. There are less than two states. Therefore, a state machine is not required.
4. The component SC_Adapter: no refinement exists in Step D3: Intra-Component Interaction; continue with looking at Step D2: Inter-Component Interaction. There are les than two states. Therefore, a state machine is not required.
5. The component UDB_Adapter: no refinement exists in Step D3: Intra-Component Interaction; continue with looking at Step D2: Inter-Component Interaction. There are less than two states. Therefore, a state machine is not required.

State machine


IDLE
doGet(reqRegister, resRegister)
forward(reqRegister, resRegister)
doGet(reqRegister, resRegister)
forward(reqRegister, resRegister)
RegisteredCustomerWebPage
doGet(reqRegister, resRegister)
forward(reqRegister, resRegister)
doGet(reqRegister, resRegister)
forward(reqRegister, resRegister)
doPost(reqBrowse, resBrowse)
res:=get_Products(),
reqBrowse.setAttribute('res',res),
forword(reqBrowse,resBrowse)
SelectProducts
showaddshoppingFailed
showaddShoppingConfirmed
doGet(reqSelect,resSelect)/
forward(reqSelect,resSelect)
Shopping
doPost(reqAddShoppingCart,
resAddshoppoingCart)/
res:= addingProductToShoppingCart(username,
ProductID)
[res=false]
reqAddToShoppingCart.setAttribute('res',res)
forward(reqAddToShoppingCart,resAddToShoppingCart)
[res=true]
reqAddToShoppingCart.setAttribute('res',res)
forward(reqAddToShoppingCart,resAddToShoppingCart)

Remarks

We add the following additional transitions to model the complete behavior of the web application:

- doGet(reqDefault,resDefault) is a trigger that represents the initial request for the webpage when entering the URL.
- forward(reqDefault,resDefault) is the corresponding action to generate the starting page.
- SetAttribute(“name” , value) is a method to use results in a generated webpage.

Glossary

Name	Type	Description	Source
A			
AT		Abbreveiation of ApacheTomcat	TCD
Api	Technical phenomena	Application program interface	TCD
ApachteTomcat	ConnectionDomain	An Open Source JSP and Servlet Container	TCD
AccessAllProducts	Phenomenon	Software user to access all products	Contextdiagram
AddingProduct	Phenomenon	machine requests adding a movie to DB	Contextdiagram sdaddShoppingCart_app sd ViewProducts_app
AddingProduct ToShoppingCart	Phenomen	Software user for adding products	Contextdiagram
AddToShoppingCart	Phenomen	Internal software user	Problemdiagram
AvailableProduct	Phenomenon	Software user for Available Product	Contextdiagram
AvaliableProduct	Phenomenon	Software user for Available Product	Problemdiagram
B			
BuyProducts	Phenomenon	Software user for buying products	Contextdiagram

Name	Type	Description	Source
Browse	Phenomenon	Software user for browsing	Contextdiagram
browse AvailableProduct	Phenomenon	To browse AvailableProduct	sd ViewProducts
BrowseAvailableProduct	Phenomenon	Software user to browse available product	Problemdiagram
C			
Call_return,sql	Technical phenomen on	Interface when using database on the same machince as the programm	TCD
CCmds	interface	used to trigger the operations "CustomerRegistration"	subArchRegister, globalArch
Create()	Auxiliary function	Software user to create products	sd ViewProducts_app
CreateNewProduct	Phenomenon	Software user to create products	Contextdiagram
Customer	Biddable Domain	User of the software	Contextdiagram
CustomerWebPage	Connection domain, Class	Web page that controls registration	pdRegister, sdRegister, CR ClassDiagram
CurrentContents OfShoppingCart	Phenomenon	Software user for current contents of shoppingcart	Problemdiagram
checkNumber()	Auxiliary function	Provides the number of the product	DP ClassDiagram, sd DeleteProduct, sd DeleteProduct_app
CustomerGUI	Component	Web interface for customers	subArchRegister, globalArch
ComponentTest	Class	Test class for the components	Class model

Name	Type	Description	Source
checkDeleteApps	Method	A function that god fired each 1 sec to delete not available products	Class Timercheck
D			
doGet	Phenomenon	To get information from server	TCD
doPost	Phenomenon	To send information to the server	TCD
doGet()	message, state	To send information to the server	sdRegister_app, sdaddShoppingCart_app, sdViewProduct_app, State Machine
doPost()	message, state	To send information to the server	sdRegister_app, sdaddShoppingCart_app, sdViewProduct_app, State Machine
Description	Attribute	Represents description of the product	Class model
delete()	Auxiliary function	Delete the product	Class model
DeleteB	Table name	SQL table name for deleting	SQL
E			
Employee	Biddable Domain	Internal software user	Contextdiagram
EnvironmentApachteTomcat	ConnectionDomain	An Open Source JSP and Servlet Container	sdRegister_app, sdaddShoppingCart_app, sdViewProduct_app sdDeleteProduct_app
ExecuteUpdate	Phenomenon	To send Information to the DataBase	TCD, sdAddShoppingCartt_app,

Name	Type	Description	Source
			sdDeleteProduct_app
ExecuteQuery	Phenomenon	To get information from the Database	TCD, sdRegister_app, sdaddShoppingCart_app, sdViewProduct_app
F			
FailInfoRepresentation	Phenomenon	shows user that registration failed	sdRegisterUser, sdAddToShoppingCart
FeedbackE	Phenomenon	Software user for feedback	Contextdiagram
FeedbackRC	Phenomenon	Software user for feedback	Contextdiagram
Feedback	Phenomenon	Software user for feedback	Problemdiagram
forward()	message, state	sends the request and response back to the server to generate the HTML webpage	sdaddShoppingCart_app State Machine
ForwardBrowseProduct	Phenomenon	To forward the browsing from connection domain to machine	sd ViewProducts, sd ViewProducts_app, sdAddShoppingCartt_app
ForwardAddProductToCart	Phenomenon	OS_AddToShoppingCArt operation for adding to shopping cart	SC ClassDiagram sdViewProducts_app
G			
getProducts	Phenomenon	To get products	sd ViewProducts sd ViewProducts_app sdaddShoppingCart_app

Name	Type	Description	Source
getProductByID	Phenomenon	To get product by a specific Id	Class model
GUI	technical phenomen on	Graphical user interfase	TCD
GUISystemTest	Class	Class for the system test	Class model
H			
http	technical phenomen on	Hypertext Transfer Protocol	TCD
I			
IDLE	State	Indicates that the server waits for incoming requests	State Machine RegisteredCostumer GUI
InsertShoppingCart	Table name	SQL table name for inserting products in shopping cart	SQL
IProductDatabase	interface	used to trigger the operation “ProductList”	subArchProductList, subArchAddToShoppingCart, subArchDeleteProduct, globalArch
IShoppingCartDatabase	interface	used to trigger the operation “ShoppingCart”	subArchAddToShoppingCart, globalArch
IUserDatabase	interface	used to trigger the operation “CustomerRegistration”	subArchRegister, globalArch
J			
java.sql.Statement	interface	Java API to communicate with Databases	subArchRegister, subArchProductList, subArchAddToShoppingCart, subArchDeleteProduct, globalArch

Name	Type	Description	Source
javax.servlet.http.HttpServlet	interface	Java API from Apache Tomcat server for web communication	subArchRegister, subArchProductList, subArchAddToShoppingCart, globalArch
javax.servlet.RequestDispatcher	interface	Java API from Apache Tomcat server for web communication	subArchRegister, subArchProductList, subArchAddToShoppingCart, globalArch
K			
L			
LogIn	Phenomenon	Software user to log in	Contextdiagram
LogInInformation	Phenomenon	Software user for login information	Contextdiagram
LogOut	Phenomenon	Software user to log out	Contextdiagram
$\boldsymbol{LC}_{\mathbf{Person}}$	Life-cycle	Life-cycle for one person	LC
$\boldsymbol{LC}_{\mathbf{Customer}}$	Life-cycle	Life-cycle for one RegisteredUser	LC
$\boldsymbol{LC}_{\mathbf{Customer\ OnlineMarketAppr}}$	Life-cycle	Life-cycle for the whole machine	LC
M			
N			
Name	Attribute	Represents name of the product	Class model
NewProductlists	Class	Represents modified productlist	Class model

Name	Type	Description	Source
nostock()	Auxiliary function	Provides the number of the product in DB	DP ClassDiagram, sd DeleteProduct, sd DeleteProduct_app
O			
OnlineShopApp	Machine	Software to be developed	Contextdiagram, TechnicalContextDiagram
OnlineShopApp	component	responsible for the communication between the machine and all other components.	subArchRegister, subArchProductList, subArchAddToShoppingCart, subArchDeleteProduct, globalArch
okRepresentation	Phenomanen	shows User that Registration has failed	sd Register, sdAddToShoppingCart
OS_Register	Machine	Software to be developed	Problemdiagram
OS_ProductList	Machine	Software to be developed	Problemdiagram
OS_AddToShoppingCart	Machine	Software to be developed	Problemdiagram
OS_DeleteProduct	Machine	Software to be developed	Problemdiagram
P			
ProductDatabase	Lexical Domain	User of the software	Contextdiagram
PutProductsUp ToFiveUnits	Phenomenon	User of the software	Contextdiagram
PutProductsUp ToFiveUnits	Phenomenon	User of the software	Problemdiagram
Products	Table name	SQL table name for ViewProducts	SQL
ProductDatabase	lexicaDomain, DesignedDomain	Software user of the product database	Problemdiagram

Name	Type	Description	Source
ProductDetials	Phenomenon	Software user for products detials	Problemdiagram
ProductList	Phenomenon	Software user for products list	Problemdiagram
ProductsQuery	Table name	SQL table name for Products	SQL
ProductsRepresentation	Phenomenon	To represent products	sd ViewProducts
Password	Attribute	Represents password of the username	Class model
ProductID	Attribute	Represents ID of the product	Class model
Price	Attribute	Represents price of the product	Class model
ProductQuantity	Attribute	Represents quantity of the product	Class model
Products	Class	Represents products added to shopping cart	Class model
ProductsList	Class	Represents viewed products list	Class model
PDB_Adapter	Component	Adapter for using the ProductDataBase	subArchProductList, subArchAddToShoppingCart, subArchDeleteProduct, globalArch
Q			
R			
RCCmds	interface	used to trigger the operations “CustomerRegistration”, “ProductList”, ” ShoppingCart”	subArchProductList, subArchAddToShoppingCart, globalArch
Register	Phenomenon	Software user to register	Contextdiagram, Problemdiagram, sdRegister
Registered Customer	Biddable Domain	User of the software	Contextdiagram

Name	Type	Description	Source
RegisteredCustomerWebPage	Connection domain, Class, state	Web page that controls overviewing the products , Webpage that user can use to add Products to the SC	pdViewProducts, pd AddToShoppingCart , sdViewProducts, sdAddToShoppingCart, ViewProducts ClassDiagram AddToShoppingCart ClassDiagram, State Machine RegisteredCostumer GUI
RegisteringUser	Phenomenon	Software user to register	Contextdiagram, sdRegister
RegisteringUser()	auxiliary function	Registering the user in the app	sdRegister _app
RegisterCustomer	Phenomenon	Software Customer to register	Problemdiagram
RegisteredCustomer	Phenomenon	Software user for a registered customer	Problemdiagram
RegisterUser()	Auxiliary function	OS_Register operation for register	CRClassDiagram, sdRegister, sdRegister _app
RegisteredCustomerGUI	Component	Web interface for registered customers	subArchProductList, subArchAddToShoppingCart, globalArch
RemovingProduct	Phenomenon	Software user to removing products	Contextdiagram
resRegister	Table name	SQL table name for users	SQL
reqRegister	Table name	SQL table name for users	SQL
reqAddToShoppingCart .setAttributes()	Phenomenon function	Request of function	State machine
res:=addingProductToShoppingCart()	Phenomenon function	Result of function browse	State machine
Res	attribute	Result of function browse	VPClassDiagram sdViewProduct_app

Name	Type	Description	Source
S			
SCDB_Adapter	Component	Adapter for using the ShoppingCartDatabase	subArchAddToShoppingCart, Architecture, globalArch
Shopping	Table name	SQL table name for AddToShoppingCart	SQL
Shopping	State	Indicates shopping	State Machine RegisteredCostumer GUI
ShoppingCart	Lexical Domain	User of the software	Contextdiagram
shoppingExist()	auxiliary function	Checks if Shopping data exists in the Database already	sd addShoppingCart_app
SubmitOrder	Phenomenon	Software user to submit an order	Contextdiagram
SendOrder	Phenomenon	Software user to send an order	Contextdiagram
SelectProduct	Phenomenon	Software user to select products	Problemdiagram
SelectProducts	State	Indicates selecting products	State Machine RegisteredCostumer GUI
select Product List Page	state predicates	To select Product in the List Page	sd ViewProduct_app, State machine
showProducts	Phenomenon	To show Products	sd ViewProducts
SortedProductList	Phenomenon	Software user for a sorted product list	Contextdiagram
sortedProductIn DescendingPriceOrder	Phenomenon	Software user for sorting product in descending order	Problemdiagram
ShowShoppingCart	Phenomenon	Software user for showing shopping cart	Problemdiagram

Name	Type	Description	Source
showaddShoppingConfirmed	State	To show add Shopping Confirmed	State Machine
showaddShoppingFail	State	To Show add Shopping Fail	State Machine
ShowOkR	Phenomenon	Success feedback for register	CRClassDiagram
ShowFailR	Phenomenon	Success feedback for register	CRClassDiagram
ShowOkA	Phenomenon	Success feedback for adding to shopping cart	SCClassDiagram
ShowFailA	Phenomenon	Success feedback for adding to shopping cart	SCClassDiagram
showOk	Phenomen on, auxiliary function	sends showOk to Webpage	sdregister, sdAddToShoppingCart, SCClassDiagram, CRClassDiagram,
showFail	Phenomen on, auxiliary function	sends showFail to Webpage	sdregister, sdAddToShoppingCart, SCClassDiagram, CRClassDiagram,
showaddShoppingConfirmed	state predicates	Shows Shopping Confirmed	sdaddShoppingCart_app, State machine
showaddshoppingFailed	state predicates	Shows Shopping Failed	sdaddShoppingCart_app, State machine
ShowRegisterFailed	state predicates	Shows register failed	SdRegister_app
ShowRegisterConfirmed	state predicates	Shows register confirmed	SdRegister_app
SQLDatabase	Causal domain	Database to be used in the Programm	TCD
T			
Timer	Phenomen on, auxiliary function	To check the timing of deleting	subArchDelete sd DeleteProduct_app
TimerCheck	Class	Class that contains the timer logic	sdDeleteProduct_app
testAlreadyRegsiterUser()	Test function	Test the dbfacade method registeringUser() in	Class ComponentTest

Name	Type	Description	Source
		case the user already registred	
testAddProductToShoppingList	Test function	Test the GUI webpage of adding the product to the shopping cart	Class GUISystemTest
testBrowseProducts ()	Test function	Test the GUI webpage AllProductsOverview	Class GUISystemTest
testRegsiterUser()	Test function	Test the dbfacade method registeringUser()	Class ComponentTest
testOverView ()	Test function	Test the dbfacade method getProducts()	Class ComponentTest
testDeleteProductsWithQuantatityZero	Test function	Test deleting of the products that have 0 quantatiy	Class ComponentTest
testAddSameProductTwice	Test function	Test adding the products twice to the same shopping cart	Class ComponentTest
testAddProductsMorethanFiveTimes	Test function	Test adding more than 5 products to the shopping cart	Class ComponentTest
testAddProductToshoppingCart	Test function	Test adding product to the shopping cart of the user	Class ComponentTest
U			
User	Table name	SQL table name users	SQL
UserDatabase	Lexical Domain	User of the software	Contextdiagram
UserName	Attribute	Represents unique username of customer	CRClassDiagram
usernameExists()	Auxilary function	Checks if username already exists in the UsersDatabase	sdRegister_app CD_RegisterUser the Operation User Registration Operation Specification 2

Name	Type	Description	Source
usernameExistsRes	Message,variable	Response from the usernameExists	sdRegister_app
uExistsRes	Message, variable	Response from executequery	sdRegister_app
userShoppingExistRes	Message,variable	Response from shoppingExists	addShoppingCart_app,The Operation addShoppingCart, Operation Specification 2
UDB_Adapter	Component	Adapter for using the UserDataBase	subArchRegister, globalArch
ud	Parameter	AParameter that represent the entity Userdatabase	Class model
userData	Parameter	AParameter that represent the entity Userdatabase	Class model
V			
ViewProduct	Phenomenon	Software user to view product	Contextdiagram
viewAvailableProduct	Phenomenon	Software user to view available product	Problemdiagram
W			
WebPage	Connection Domain	Software user of web page	Problemdiagram