IN2013 Object-Oriented Analysis and Design

Smart Fridge
Software

Georgios Karanasios

11/11/2018

Pictures with better resolution provided after page 5 (zoom needed)!

Scenario (briefly introduction): Smart Fridges software called "Smart Fridges"

Develop a new class of fridges which in addition to the usual functionality of a fridge (such as temperature control, etc.) provides additional "smart features", such as food inventory and internet shopping. EasyLiving are developing the software for "Smart Fridges" line of products. The software system is called SmartFridgeX. The SmartFridgeX will be divided into two main subsystems, one deployed on the fridge (called SFXfridge) and the other one – a mobile app (SFXapp).

Use case realization (sequence diagram)

(If it was necessary to include the preconditions of the use case specification, please check my vpp file which has a separate sequence diagram for them with its revised analysis diagram)

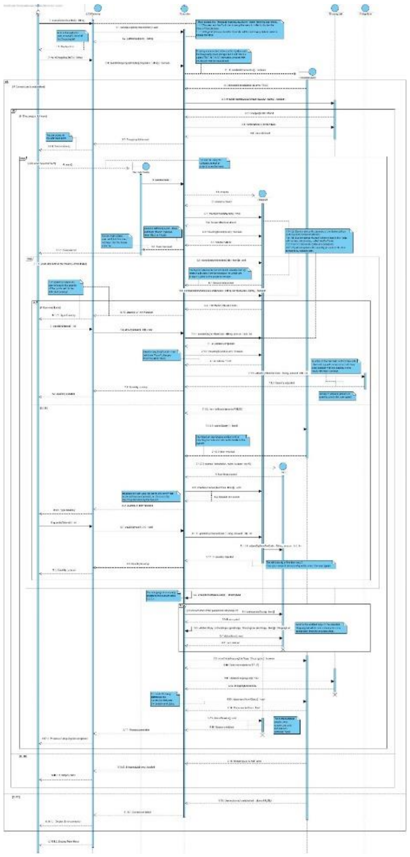
The only change which I made to the use case specification is on the Main flow (step 7-9). Now, the System creates the copy of the selected ShoppingList (object) and the copy of the items and merges them before sending to the cloud. Then, the System uploads the updated copy of the ShoppingList (included all the copied items) to the cloud. In this way, adding the copied items to the copy of ShoppingList is faster because the process does not depend on the cloud's internet but on the fridge system's speed, which is clearly faster and more efficient (steps in my sequence diagram 9.3 – 9.14).

7) The System creates a copy of the selected ShoppingList (object) on CloudStorage.
8) For each item of the processed ShoppingList:
8.1) The System sends a copy of the item to the cloud storage to be added to the copy of the ShoppingList created in step 7 above.
8.2) The System deletes the item from the memory of SFXfridge.
9) The System deletes the processed ShoppingList from the memory of SFXfridge.
10) The System disconnects from EasyLiving cloud storage.

↓

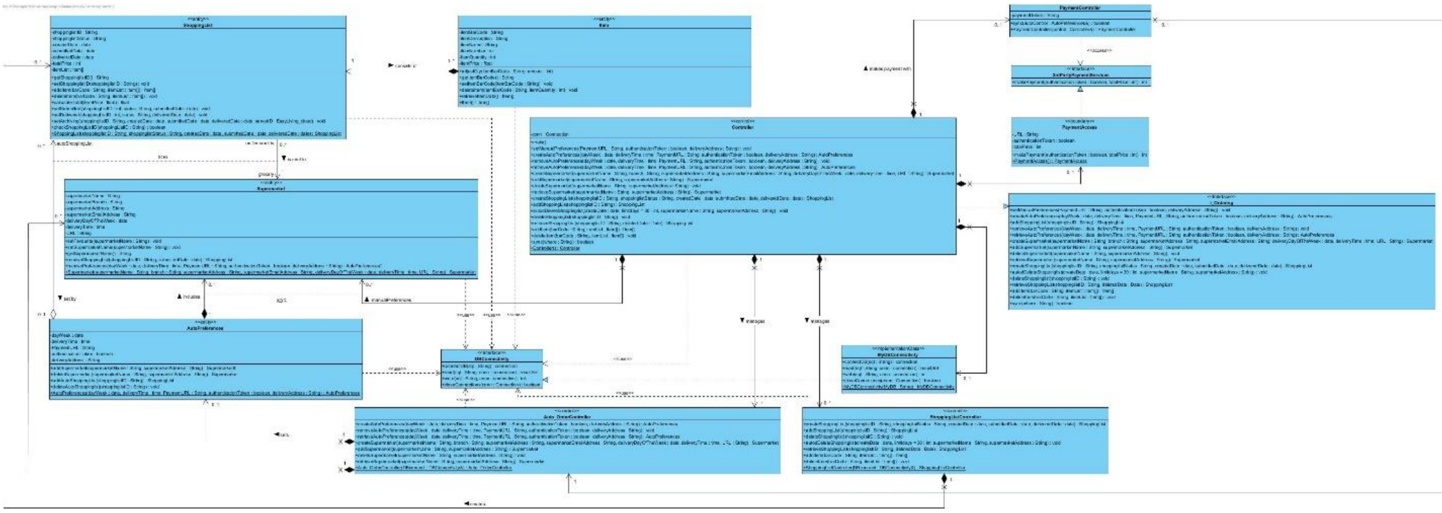
7) The System creates a copy of the selected ShoppingList (object).
8) For each item of the processed ShoppingList:
8.1) The System creates a copy of the item.
8.2) The systems add the copy of the item to the copy of the selected ShoppingList (object) created in step 7 above.
8.2) The System deletes the item from the memory of SFXfridge.
9) The system sends the updated copy of the selected ShoppingList (object) to the cloud storage.
10) The System deletes the processed ShoppingList from the memory of SFXfridge.
11) The System disconnects from EasyLiving cloud storage.

A faithful realization of the use-case specification is provided below:



Design class diagram

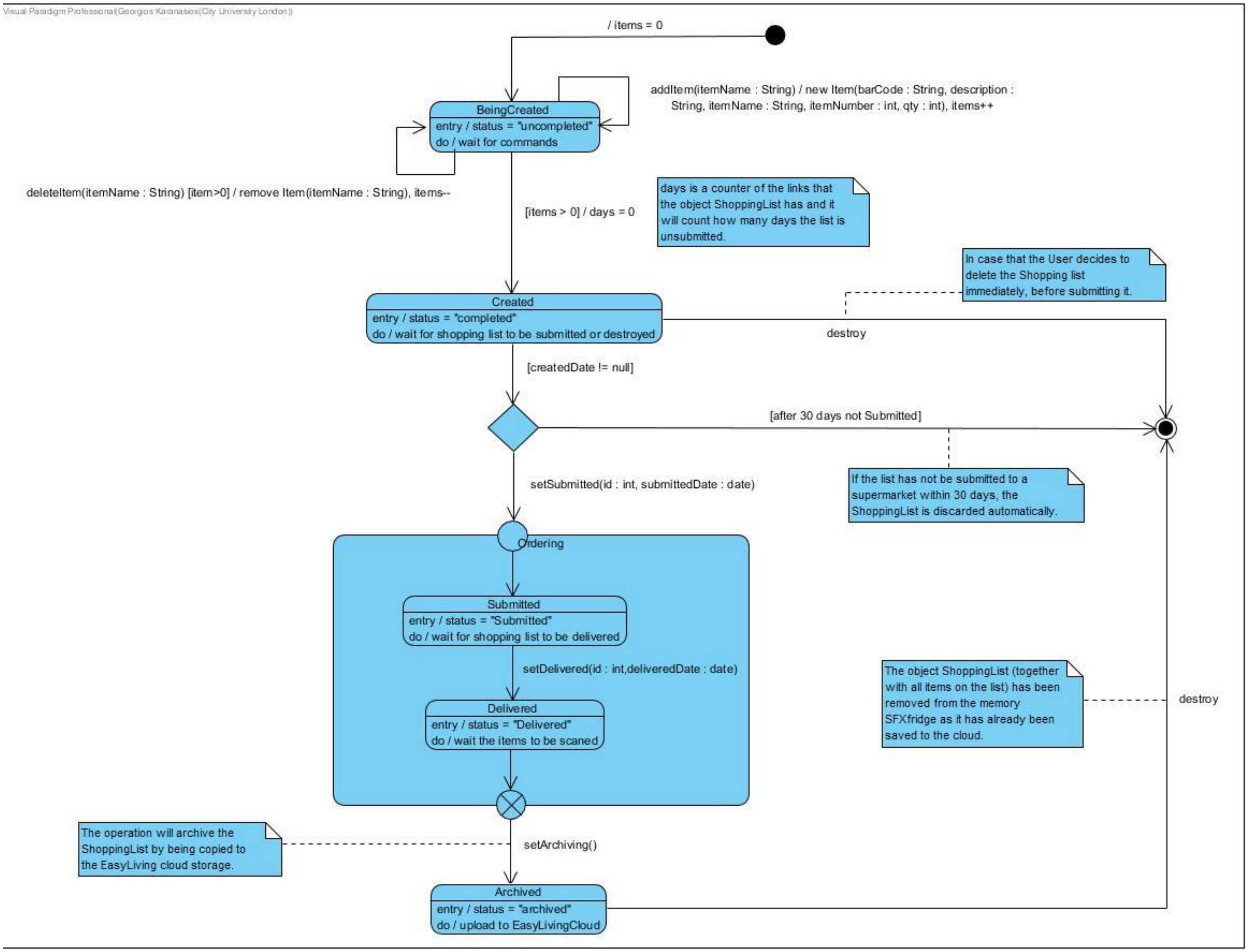
A plausible model answer is provided below:



- The diagram is self-explanatory. The classes from the analysis class diagram are refined with the design details as required:
  - Attributes and methods are fully specified.
  - Constructors are added (and specified with classifier scope, i.e. appear underscored) to all classes irrespective of their stereotype and whether they are from the problem/implementation domain (e.g. DBConnectivity).
  - The associations among the classes are also refined. In my diagram, all associations among problem domain classes are modelled as compositions (except between AutoPreferences and ShoppingList because ShoppingList could be set up by AutoPreferences but it does not have to be).
- Boundary class in the analysis class diagram is replaced with a pair interface/implementation class, as hinted in the assignment.
- The control class is splinted into several control classes to improve cohesion (AutoPreferencesController, ShoppingListController, PaymentController). The Controller is the "master" of the system (I added a main method to indicate the method that will be run first when the application is started).
- In addition, the Controller class has the responsibility of initializing all other control and implementation classes. This additional responsibility is the reason why the Controller class has composition relationships with the other control classes and with the implementation classes.
- The relationships of the interfaces with the implementation classes and with the classes using the interfaces are modelled as realization and dependence, respectively. Finally, there is an interface class called I_Ordering which allows the other subsystems to have access to the Ordering sub-system and vice versa. This interface is implemented by the Controller (hence the realization relationship).

State machine

A plausible diagram is provided below:

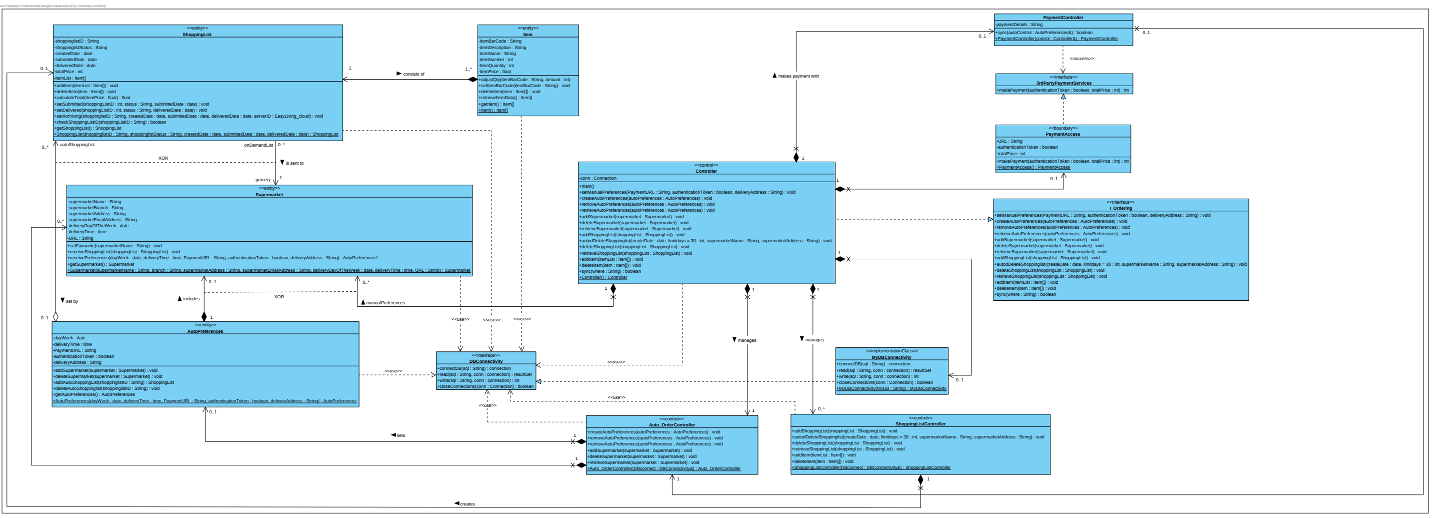


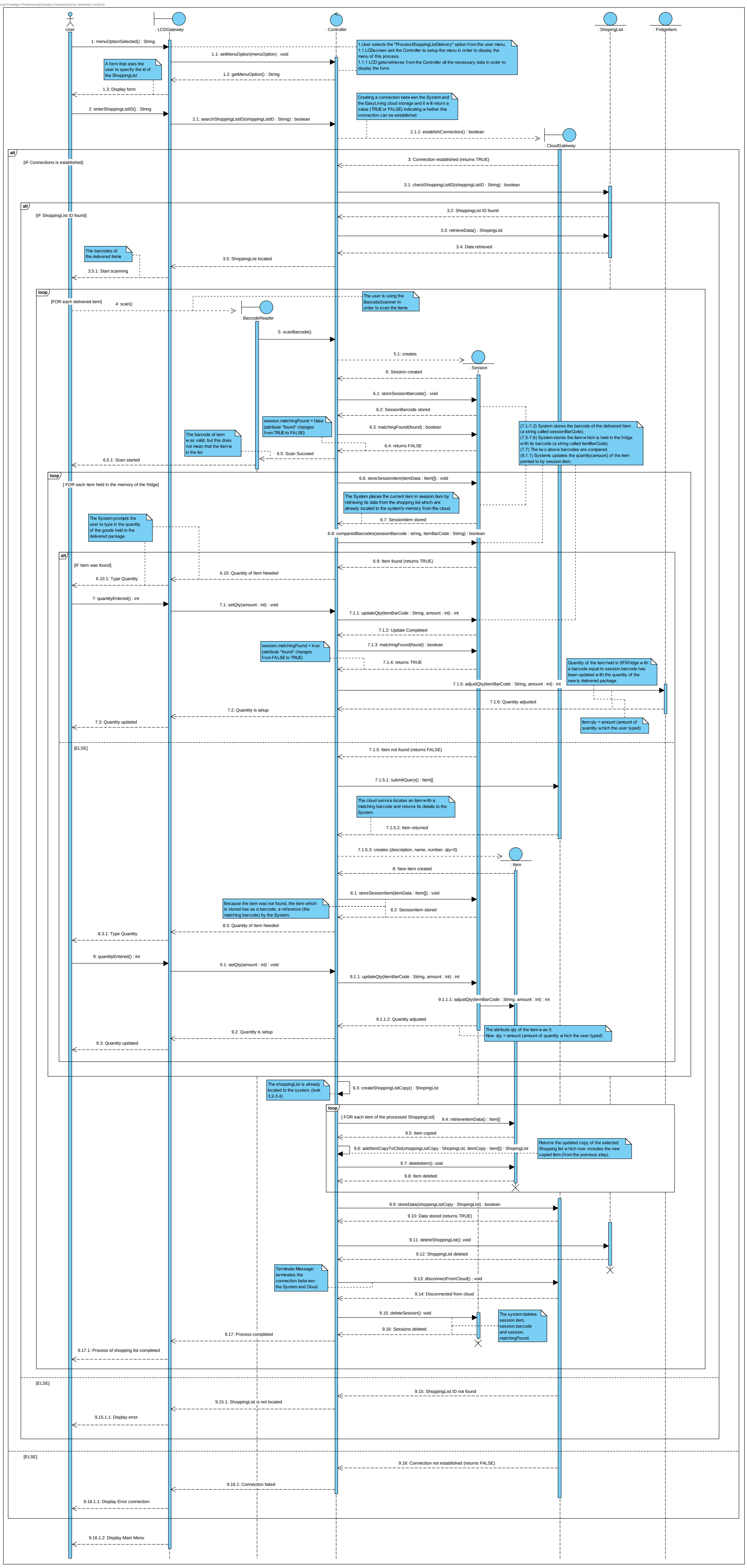
The diagram consists of <u>5 states</u>:

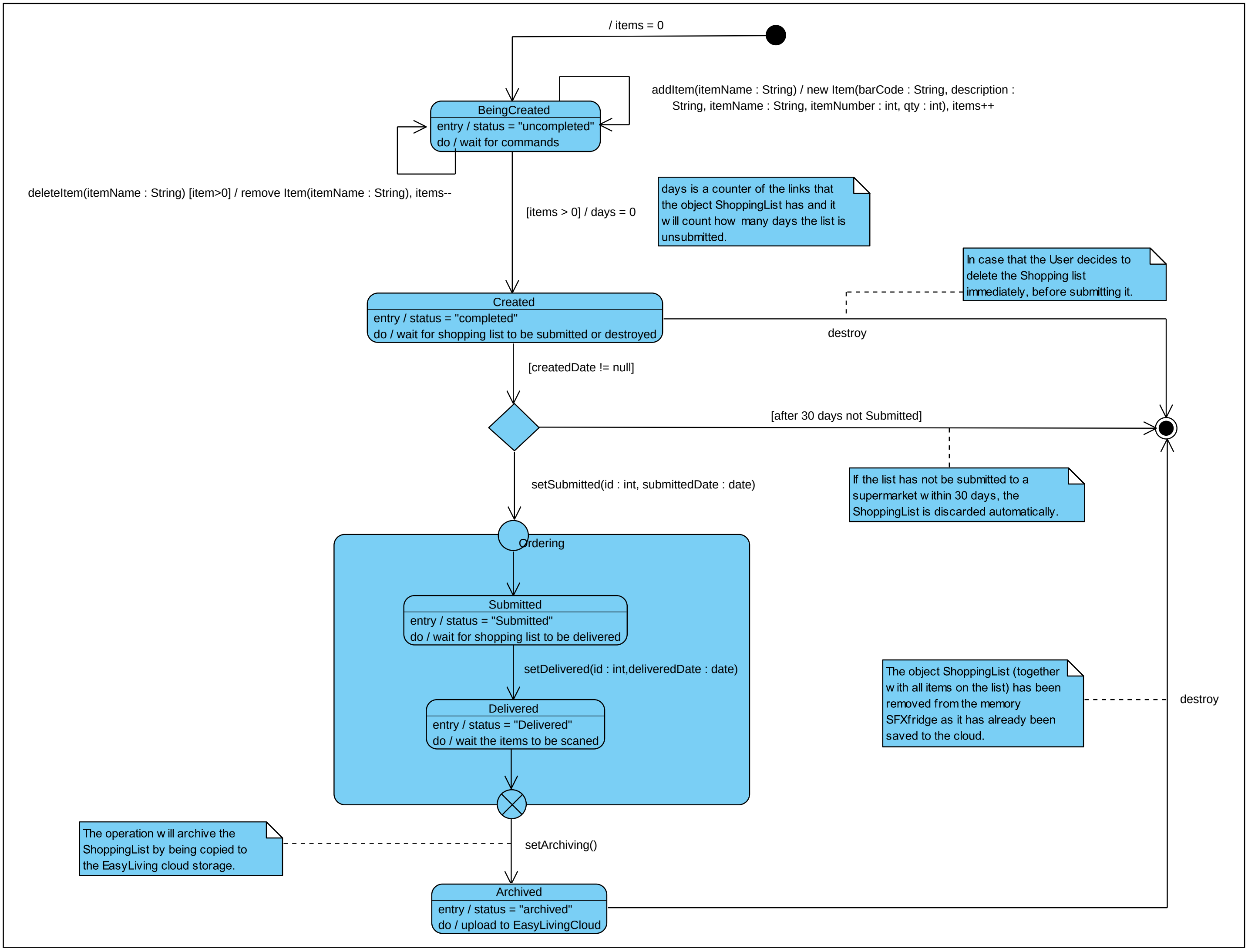
- <u>BeingCreated</u>. This is the first state where the (object-type) ShoppingList is going to be created, and the state reacts to 2 different call events – addItem() and removeItem(). These two increases or decreases the number of (objects-type) Items included in the Shopping list with some trivial checks put in place.
- <u>Created</u>. When all the Items has been added (items >0), a Shopping list is officially created, and its status has been changed to "completed" (specified in the entry action of the state).
  In this state, the object will only respond to the event setSubmitted() or destroy. Other events, including the call events addItem() and removeItem() responded to in the <u>BeingCreated</u> state, will be ignored (i.e. will have no impact on the object attributes and links).
  The transition from <u>Created</u> to <u>Submitted</u> may occur when the user set the Shopping list as "submitted". If 30 days have passed without being defined as "submitted", it is automatically

destroyed by the system and the lifecycle terminates. Also, there is the case that the user decides to delete the Shopping list immediately before submitting it, then the lifecycle terminates too.

- The other 2 states "Submitted" and "Delivered", capture two distinct states. The first is the state when the Shopping list is waiting to be delivered, (i.e until the user receives all the Items from the supermarket). The trigger here is the event setDelivered() which will move us to the next state and the status of the Shopping list will change to "delivered". Once all the Items of the Shopping list are delivered, the system is waiting the user to scan the items by using the barcode reader. When all the delivered items have been scanned, the state reacts to the call event setArchived() which archives the ShoppingList as a copy.
- Archived. This state captures the state where the ShoppingList object is archived (status changes to "archived") and the system uploads its copy (with all the delivered Items) to the EasyLiving Cloud storage. After that, the Shopping list is going to be destroyed from the memory of the fridge and the lifecycle terminates.


<<entity>>
ShoppingList
-shoppinglistID : String
-shoppinglistStatus : String
-createdDate : date
-submittedDate : date
-deliveredDate : date
-totalPrice : int
-itemList : Item[]
+addItem(itemList : Item[]) : void
+deleteItem(item : Item[]) : void
+calculateTotal(itemPrice : float) : float
+setSubmitted(shoppingListID : int, status : String, submittedDate : date) : void
+setDelivered(shoppingListID : int, status : String, deliveredDate : date) : void
+setArchiving(shoppinglistID : String, createdDate : date, submittedDate : date, deliveredDate : date, serverID : EasyLiving_cloud) : void
+checkShoppingListID(shoppingListID : String) : boolean
+getShoppingList() : ShoppingList
+ShoppingList(shoppinglistID : String, shoppinglistStatus : String, createdDate : date, submittedDate : date, deliveredDate : date) : ShoppingList
<<entity>>
Item
-itemBarCode : String
-itemDescription : String
-itemName : String
-itemNumber : int
-ItemQuantity : int
-itemPrice : float
+adjustQty(itemBarCode : String, amount : int)
+setItemBarCode(itemBarCode : String) : void
+deleteItem(item : Item[]) : void
+retrieveItemData() : Item[]
+getItem() : Item[]
+Item() : Item[]
consists of
1
1..*
0..1
0..*
autoShoppingList
onDemandList
0..*
XOR
is sent to
grocery
1
<<entity>>
Supermarket
-supermarketName : String
-supermarketBranch : String
-supermarketAddress : String
-supermarketEmailAddress : String
-deliveryDayOfTheWeek : date
-deliveryTime : time
-URL : String
+setFavourite(supermarketName : String) : void
+receiveShoppingList(shoppingList : ShoppingList) : void
+receivePreferences(dayWeek : date, deliveryTime : time, PaymentURL : String, authenticationToken : boolean, deliveryAddress : String) : AutoPreferences*
+getSupermarket() : Supermarket
+Supermarket(supermarketName : String, branch : String, supermarketAddress : String, supermarketEmailAddress : String, deliveryDayOfTheWeek : date, deliveryTime : time, URL : String) : Supermarket
0..*
0..1
0..*
XOR
set by
includes
manualPreferences
0..1
1
<<entity>>
AutoPreferences
-dayWeek : date
-deliveryTime : time
-PaymentURL : String
-authenticationToken : boolean
-deliveryAddress : String
+addSupermarket(supermarket : Supermarket) : void
+deleteSupermarket(supermarket : Supermarket) : void
+addAutoShoppingList(shoppinglistID : String) : ShoppingList
+deleteAutoShoppinglist(shoppinglistID : String) : void
+getAutoPreferences() : AutoPreferences
+AutoPreferences(dayWeek : date, deliveryTime : time, PaymentURL : String, authenticationToken : boolean, deliveryAddress : String) : AutoPreferences
0..1
sets
<<use>>
<<use>>
<<use>>
<<use>>
<<Interface>>
DBConnectivity
+connectDB(sql : String) : connection
+read(sql : String, conn : connection) : resultSet
+write(sql : String, conn : connection) : int
+closeConnections(conn : Connection) : boolean
<<use>>
<<use>>
<<use>>
makes payment with
1
<<control>>
Controller
-conn : Connection
+main()
+setManualPreferences(PaymentURL : String, authenticationToken : boolean, deliveryAddress : String) : void
+createAutoPreferences(autoPreferences : AutoPreferences) : void
+removeAutoPreferences(autoPreferences : AutoPreferences) : void
+retrieveAutoPreferences(autoPreferences : AutoPreferences) : void
+addSupermarket(supermarket : Supermarket) : void
+deleteSupermarket(supermarket : Supermarket) : void
+retrieveSupermarket(supermarket : Supermarket) : void
+addShoppingList(shoppingList : ShoppingList) : void
+autodDeleteShoppinglist(createDate : date, limitdays = 30 : int, supermarketName : String, supermarketAddress : String) : void
+deleteShoppingList(shoppingList : ShoppingList) : void
+retrieveShoppingList(shoppingList : ShoppingList) : void
+addItem(itemList : Item[]) : void
+deleteItem(item : Item[]) : void
+sync(where : String) : boolean
+Controller() : Controller
1
1
1
1
1
manages
manages
PaymentController
-paymentDetails : String
+sync(autoControl : AutoPreferences&) : boolean
+PaymentController(control : Controller&) : PaymentController
0..1
0..1
<<access>>
<<Interface>>
3rdPartyPaymentServices
+makePayment(authenticationToken : boolean, totalPrice : int) : int
<<boundary>>
PaymentAccess
-URL : String
-authenticationToken : boolean
-totalPrice : int
+makePayment(authenticationToken : boolean, totalPrice : int) : int
+PaymentAccess() : PaymentAccess
0..1
<<Interface>>
I_Ordering
+setManualPreferences(PaymentURL : String, authenticationToken : boolean, deliveryAddress : String) : void
+createAutoPreferences(autoPreferences : AutoPreferences) : void
+removeAutoPreferences(autoPreferences : AutoPreferences) : void
+retrieveAutoPreferences(autoPreferences : AutoPreferences) : void
+addSupermarket(supermarket : Supermarket) : void
+deleteSupermarket(supermarket : Supermarket) : void
+retrieveSupermarket(supermarket : Supermarket) : void
+addShoppingList(shoppingList : ShoppingList) : void
+autodDeleteShoppinglist(createDate : date, limitdays = 30 : int, supermarketName : String, supermarketAddress : String) : void
+deleteShoppingList(shoppingList : ShoppingList) : void
+retrieveShoppingList(shoppingList : ShoppingList) : void
+addItem(itemList : Item[]) : void
+deleteItem(item : Item[]) : void
+sync(where : String) : boolean
<<implementationClass>>
MyDBConnectivity
+connectDB(sql : String) : connection
+read(sql : String, conn : connection) : resultSet
+write(sql : String, conn : connection) : int
+closeConnections(conn : Connection) : boolean
+MyDBConnectivity(MyDB : String) : MyDBConnectivity
0..1
1
0..*
1
1
<<control>>
Auto_OrderController
+createAutoPreferences(autoPreferences : AutoPreferences) : void
+removeAutoPreferences(autoPreferences : AutoPreferences) : void
+retrieveAutoPreferences(autoPreferences : AutoPreferences) : void
+addSupermarket(supermarket : Supermarket) : void
+deleteSupermarket(supermarket : Supermarket) : void
+retrieveSupermarket(supermarket : Supermarket) : void
+Auto_OrderController(DBconnect : DBConnectivity&) : Auto_OrderController
1
<<control>>
ShoppingListController
+addShoppingList(shoppingList : ShoppingList) : void
+autodDeleteShoppinglist(createDate : date, limitdays = 30 : int, supermarketName : String, supermarketAddress : String) : void
+deleteShoppingList(shoppingList : ShoppingList) : void
+retrieveShoppingList(shoppingList : ShoppingList) : void
+addItem(itemList : Item[]) : void
+deleteItem(item : Item[]) : void
+ShoppingListController(DBconnect : DBConnectivity&) : ShoppingListController
1
creates


User
: LCDGateway
: Controller
: ShopingList
: FridgeItem
1: menuOptionSelected() : String
1.User selects the "ProcessShoppingListDelivery" option from the user menu. 1.1 LCDscreen ask the Controller to setup the menu in order to display the menu of this process. 1.1.1 LCD gets/retrieves from the Controller all the necessary data in order to display the form.
1.1: setMenuOption(menuOption) : void
A form that asks the user to specify the id of the ShoppingList
1.2: getMenuOption() : String
1.3: Display form
2: enterShoppingListID() : String
Creating a connection between the System and the EasyLiving cloud storage and it will return a value (TRUE or FALSE) indicating whether this connection can be established.
2.1: searchShoppingListID(shoppingListID : String) : boolean
2.1.1: establishConnection() : boolean
: CloudGateway
alt
[IF Connections is established]
3: Connection established (returns TRUE)
3.1: checkShoppingListID(shoppingListID : String) : boolean
alt
[IF ShoppingList ID found]
3.2: ShoppingList ID found
3.3: retrieveData() : ShopingList
3.4: Data retrieved
The barcodes of the delivered items
3.5: ShoppingList located
3.5.1: Start scanning
loop
[FOR each delivered item]
4: scan()
The user is using the BarcodeScanner in order to scan the items
: BarcodeReader
5: scanBarcode()
5.1: creates
: Session
6: Session created
6.1: storeSessionBarcode() : void
6.2: SessionBarcode stored
session.matchingFound = false (attribute "found" changes from TRUE to FALSE)
6.3: matchingFound(found) : boolean
(7.1-7.2) System stores the barcode of the delivered Item (a string called sessionBarCode). (7.5-7.6) System stores the item which is held in the fridge with its barcode (a string called itemBarCode). (7.7) The two above barcodes are compared. (8.1.1) Systems updates the quantity(amount) of the item pointed to by session.item.
The barcode of item was valid, but this does not mean that the item is in the list
6.4: returns FALSE
6.5: Scan Succeed
6.5.1: Scan started
loop
[FOR each item held in the memory of the fridge]
6.6: storeSessionItem(itemData : Item[]) : void
The System places the current item in session.item by retrieving its data from the shopping list which are already located to the system's memory from the cloud.
The System prompts the user to type in the quantity of the goods held in the delivered package
6.7: SessionItem stored
6.8: comparedBarcodes(sessionBarcode : string, itemBarCode : String) : boolean
alt
[IF Item was found]
6.9: Item found (returns TRUE)
6.10: Quantity of Item Needed
6.10.1: Type Quantity
7: quantityEntered() : int
7.1: setQty(amount : int) : void
7.1.1: updateQty(itemBarCode : String, amount : int) : int
7.1.2: Update Completed
session.matchingFound = true (attribute "found" changes from FALSE to TRUE)
7.1.3: matchingFound(found) : boolean
7.1.4: returns TRUE
Quantity of the item held in SFXfridge with a barcode equal to session.barcode has been updated with the quantity of the newly delivered package.
7.1.5: adjustQty(itemBarCode : String, amount : int) : int
7.1.6: Quantity adjusted
7.2: Quantity is setup
7.3: Quantity updated
Item qty = amount (amount of quantity which the user typed)
[ELSE]
7.1.5: Item not found (returns FALSE)
7.1.5.1: submitQuery() : Item[]
The cloud service locates an item with a matching barcode and returns its details to the System.
7.1.5.2: Item returned
7.1.5.3: creates (description, name, number, qty=0)
: Item
8: New Item created
8.1: storeSessionItem(itemData : Item[]) : void
Because the item was not found, the item which is stored has as a barcode, a reference (the matching barcode) by the System.
8.2: SessionItem stored
8.3: Quantity of Item Needed
8.3.1: Type Quantity
9: quantityEntered() : int
9.1: setQty(amount : int) : void
9.1.1: updateQty(itemBarCode : String, amount : int) : int
9.1.1.1: adjustQty(itemBarCode : String, amount : int) : int
9.1.1.2: Quantity adjusted
The attribute qty of the item was 0. Now qty = amount (amount of quantity which the user typed)
9.2: Quantity is setup
9.3: Quantity updated
The shoppingList is already located to the system. (look 3.2-3.4)
9.3: createShoppingListCopy() : ShopingList
loop
[FOR each item of the processed ShoppingList]
9.4: retrieveItemData() : Item[]
9.5: Item copied
9.6: addItemCopyToClist(shoppingListCopy : ShopingList, itemCopy : Item[]) : ShopingList
Returns the updated copy of the selected Shopping list which now includes the new copied Item (from the previous step).
9.7: deleteItem(): void
9.8: Item deleted
9.9: storeData(shoppingListCopy : ShopingList) : boolean
9.10: Data stored (returns TRUE)
9.11: deleteShoppingList(): void
9.12: ShoppingList deleted
Terminate Message: terminates the connection between the System and Cloud.
9.13: disconnectFromCloud() : void
9.14: Disconnected from cloud
9.15: deleteSession(): void
The system deletes session.item, session.barcode and session. matchingFound.
9.16: Sessions deleted
9.17: Process completed
9.17.1: Process of shopping list completed
[ELSE]
9.15: ShoppingList ID not found
9.15.1: ShoppingList is not located
9.15.1.1: Display error
[ELSE]
9.16: Connection not established (returns FALSE)
9.16.1: Connection failed
9.16.1.1: Display Error connection
9.16.1.2: Display Main Menu


/ items = 0
addItem(itemName : String) / new Item(barCode : String, description : String, itemName : String, itemNumber : int, qty : int), items++
BeingCreated
entry / status = "uncompleted"
do / wait for commands
deleteItem(itemName : String) [item>0] / remove Item(itemName : String), items--
[items > 0] / days = 0
days is a counter of the links that the object ShoppingList has and it will count how many days the list is unsubmitted.
In case that the User decides to delete the Shopping list immediately, before submitting it.
Created
entry / status = "completed"
do / wait for shopping list to be submitted or destroyed
destroy
[createdDate != null]
[after 30 days not Submitted]
setSubmitted(id : int, submittedDate : date)
If the list has not be submitted to a supermarket within 30 days, the ShoppingList is discarded automatically.
Ordering
Submitted
entry / status = "Submitted"
do / wait for shopping list to be delivered
setDelivered(id : int,deliveredDate : date)
Delivered
entry / status = "Delivered"
do / wait the items to be scaned
The object ShoppingList (together with all items on the list) has been removed from the memory SFXfridge as it has already been saved to the cloud.
destroy
The operation will archive the ShoppingList by being copied to the EasyLiving cloud storage.
setArchiving()
Archived
entry / status = "archived"
do / upload to EasyLivingCloud