IN2013 Object-Oriented Analysis and Design

"Smart Fridges" Software

Dr Peter T. Popov

Version 1.0

October 07th, 2018

Document History

1. New Document.

Introduction

There is one piece of coursework for IN2013 worth 30% of the total mark for the module. You are asked to refine a set of models by extending the analysis and then to develop a detailed design of a part of the modelled system.

Please note that although this coursework has been released at the beginning of Week 3, the coursework tests your knowledge of topics that will be taught in subsequent weeks, and you will not be able to answer all the questions until Week 4, unless you read ahead in the text book on system design and state machines.

The coursework is not due in until the end of Week 7, but you should be able to answer the first question already. Hence, you are advised to start working on the coursework as soon as possible, and not delay starting until Week 7.

The **deadline** for this coursework is **23:55 on 11th of November 2018**, and all work must be submitted electronically in accordance with the guidelines below.

Scenario: Smart Fridges software[1]

"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).

The essential functional requirements for SFXfridge subsystem are summarized as follows:

- The fridge monitors the outside temperature and adjusts the air flow to maintain the correct fridge temperature.
- If the door has been opened for more than 90 seconds, an alarm goes off to alert users.
- An LCD touch screen is provided on the front of the fridge so that the user can interact with the fridge by viewing messages, programming the operation of the fridge and extracting various reports.
- Sensors and fridge seals are monitored and in case of any fault, the user is informed via the LCD screen and via the mobile app, with an option to call in a maintenance engineer.
- The fridge is connected to the Internet via the home WiFi and can be accessed by the mobile app SFXapp. Up to 8 users can be added to the smart fridge family app group.
- The fridge has an embedded barcode scanner on the main body, mid-height, just inside from the door opening.
    - As users add new items to the fridge, they scan their barcodes.
    - Users also scan the barcodes when removing items. The default is to assume that the whole package (i.e. the entire item) will be used. The users can, however, manually override that via the LCD panel, stating how much of the package will be left, e.g. 0.5 of the package.
    - Using a barcode tagging system, the fridge keeps track of foods and their maximum storage times, and informs the user via the LCD touch screen and the mobile app of any food that has been stored for too long.
- All information about the food items is stored in a dedicated database hosted on EasyLiving's cloud servers. EasyLiving's database is populated with barcodes, with respective food item description, name and number. The database is maintained up to date by the respective food suppliers[2].
- The SFXfridge can use the inventory to determine when stocks of some items are getting low, and produce a *shopping list* to be sent to the user's favourite internet supermarket. Such lists are produced *on demand* by the user – either via LCD touch screen or from the mobile app and are assigned a unique ID by SFXfridge. The user can review and adjust the shopping list (using the LCD screen or the mobile app) and decide whether (and when) to send the shopping list to the supermarket. If the user decides to submit the shopping list to an online supermarket, SFXfridge marks the list as "submitted". The user may also decide to discard a shopping list at any time unless the shopping list has already been submitted to a supermarket. A shopping list which has not been submitted for more than 30 days will be automatically discarded by the system.
- When food items (from a submitted list) are received from the supermarket, the user marks the respective shopping list as "delivered" via LCD or SFXapp. The user scans the barcodes of all delivered items and the inventory in the fridge is updated. When all items are

[1] The case study was derived from the case study used as a coursework assignment in the module IN1005 "Software Engineering" in 2017, but is not the same.

[2] The detailed arrangements between EasyLiving and food suppliers to maintain the EasyLiving's database is outside the scope of the case study. An assumption is made, however, that should an item be searched in the EasyLiving database by its respective barcode, it will be successfully located and the database will return full details (name, number, description, barcode) about the searched item.

processed (i.e. their barcodes scanned and inventory - updated), the shopping list is "archived" on EasyLiving's cloud storage and erased from the SFXfridge' local storage.

- The SmartFridgeX has the option to set automatic reordering of selected items, too. Automatic reordering only takes place if the SmartFridgeX owner has set the following details which are stored in SmartFridgeX:
    - Intervals for creating automatic reordering lists, e.g. on Thursdays.
    - The preferred supermarket details (name, address, URL for posting an online order);
    - Days and times preference for delivery slots;
    - The owner delivery address;
    - Authentication details needed to access an online payment processor (such as PayPal) and complete an online payment transaction. An account with the chosen payment processor must be set in advance.

  These details may be entered via either the LCD screen or via SFXapp.
- Temperature control details, stock items to track and low stock thresholds (for different items) are set via the mobile app (SFXapp) and stored in the SFXfridge local storage.
- Users can produce statistics and reports on the consumption of various items in the stock. These reports can be requested/inspected via the LCD panel or via SFXapp. They include:
    - List of items in the fridge at any given time;
    - Average frequency with which certain items are bought;
    - Amounts of a given item consumed in a week/month/year;
    - All items and their corresponding amounts consumed in a week/month/year;

Assignment

You are expected to refine some of the models provided in the Appendix and develop a set of new models on SFXfridge software by answering the following questions.

Question 1: Use case realization (sequence diagram)

a) Draw a sequence diagram that realizes the use case *"ProcessShoppingListDelivery"*. In your answer you should use the use case specification (and the specifications of the related use cases) including the alternative flows and the analysis class diagram, both provided in the Appendix. The diagram should cover all possible branches, loops and alternative flows that are documented in the provided use case specifications.
Make sure that your sequence diagram is consistent with the provided class diagram and the use case specifications.
Your answer to this question should also include a *revised* analysis class diagram which reflects the changes made to it, which resulted from your work on the sequence diagram (e.g. new operations defined for some of the classes, refinement of the associations between the classes, etc.). Should you decide to amend the use-case specification, the amended use-case specifications should be submitted with your answer, too.
(25 marks)

Question 2: Design class diagram

The Appendix offers an analysis class diagram, which shows how SFXfridge software is organized in 3 packages. You are expected to design in detail one of the packages, "Ordering", which includes the following classes:

- Entity classes: Supermarket, AutoPreferences, ShoppingList and Item,
- Boundary class: PaymentAccess, which allows access to a 3rd party payment clearance services, e.g. PayPal.

Design the "Ordering" subsystem of SFXfridge software by adding design details to the "Ordering" package as follows:

- Each analysis class should be refined and its attributes and operations (methods) must be fully specified.
- Constructors (one or more) should be added to each of the design classes, as necessary.
- The associations between the design classes should be refined: the simple associations in the analysis class model should become aggregations/compositions and navigability should be added.
- Design should try to achieve *loose coupling* between classes by revising the associations between classes captured in the analysis class diagram. Some of the associations shown in the analysis class diagram may be dropped. You may also discover that some new associations are needed. You may apply CRC cards and refine the responsibilities of the classes and discover the class responsibilities, which are delegated to other (helper) classes.
- Finally, in design you must add classes from the *solution domain*. You are expected to demonstrate your understanding about how design affects class diagrams by addressing the following three aspects:
  - Assume that the SFXfridge will use a local database to store the relevant data (Supermarket, AutoPreferences, ShoppingList and Item). Provide a minimalistic design of database connectivity, which includes:
    - an interface for database access that can be used to store/retrieve data from SFXfridge to the database of choice (e.g. the popular embedded SQL Lite), and
    - an implementation class that realizes the above interface;

    You are also expected to design in detail how the data will move between entity

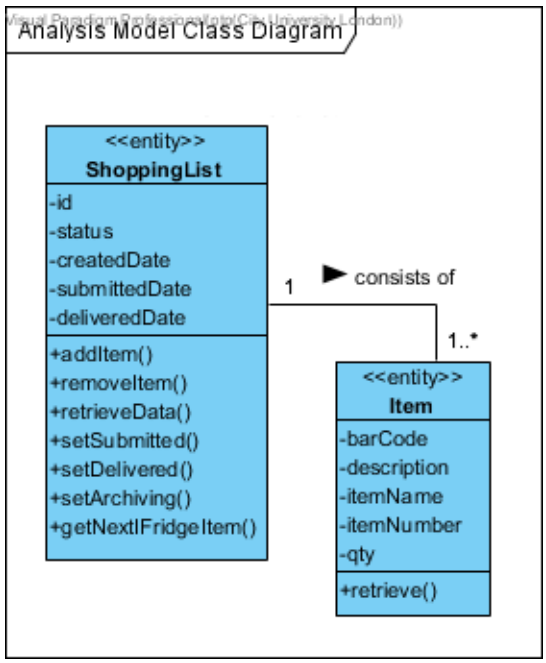
classes and database (i.e. which classes will be using/accessing database connectivity).

- Design the communication between the Ordering subsystem and the other subsystems, e.g. FridgeCore, which interact with Ordering. You may define an interface provided by the Ordering subsystems which the other sub-systems will use to access the functionality of Ordering. You may also need to define "required" interface(s), which the Ordering subsystem will use to access the other subsystems of SFXfridge software, e.g. to access the EasyLiving cloud, storage, the barcode scanner and the user interface offered in SFXfridge (LCD panel and the Mobile app).
  Hint: You do not need to provide implementation classes for the "required" interfaces as these will be provided by the other sub-systems.
- Consider using design patterns, e.g. Façade or other, suitable for the particular context to reduce the coupling between the subsystems of SXFfridge software.

(50 marks)

Question 3: State machine

Now consider a fragment of a class diagram, which models the classes ShoppingList and Item shown below.



The attributes of the class *ShoppingList* are used to store:

- "id" the unique id of a ShoppingList.
- "status" – this attribute takes values from the set {"pending", "submitted", "delivered", "archived"}.
- "createDate", "submittedDate" and "deliveredDate" – capture the dates when the status of the list has changed.

The attributes of the class Item are used to store:

- "barCode" – the unique barcode of the item.
- "description" – a brief summary of the item provided by the respective vendor.
- "itemName" – the name assigned to the item by the respective vendor.
- "itemNumber" – this is a unique alphanumeric string, assigned by the vendor, equivalent to the information captured by the barcode.
- "qty" – the current quantity of the item in the fridge (units or fractions, depending on

the nature of the item)

The classes offer a number of operations, which allow for retrieval and modification of the various object attributes at run time.

Develop a *behavioural state machine* diagram, which models the work of an instance of the class ShoppingList. The diagram should show the *states* of the instance together with the actions (entry/exit) and activities (do) associated with each of the states the instance may be in, the *transitions* between the states with their events (triggers), guards and actions. Consider the events which lead to adding a new instance of Item to the list or removing an item from a ShoppingList. Consider also the possibility for an instance of ShoppingList to be discarded automatically (after 30 days from its creation) in the case the list has not be submitted to a supermarket within 30 days. Finally, your diagram should capture the state of the ShoppingList in which the object is archived by being copied to the EasyLiving cloud storage.

> *Hint: Remember that the state of an instance is represented by the values of its attributes, the links that the instance may have with instances of other classes and the do activity associated with the state. In this case, as the fragment indicates, an instance of ShoppingList may have links with multiple instances of the class Item.*

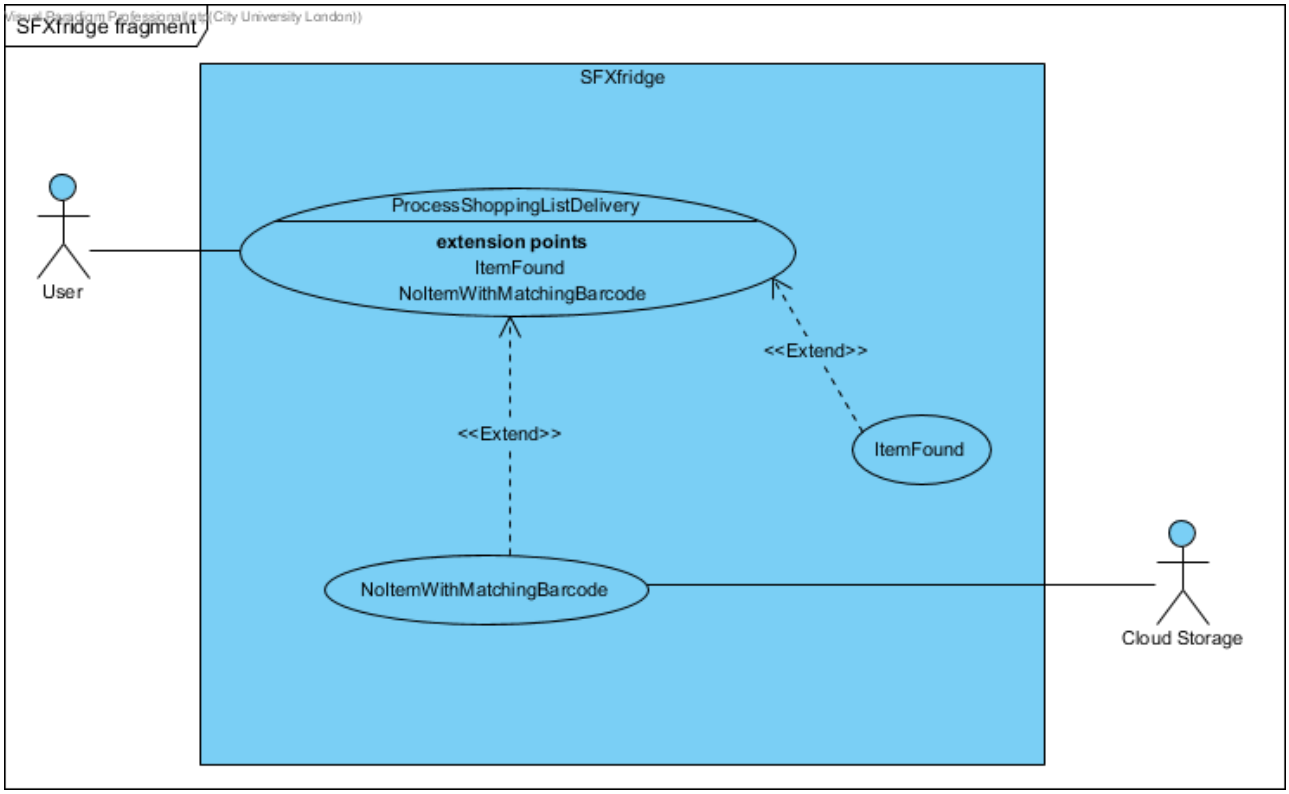
(25 marks)

Appendix

Use case model

Use case Diagram

A fragment of a use-case diagram is shown below, which captures the user interaction with SFXfridge when a Shopping List of items is delivered from a supermarket.



Specifications (of selected use cases)

The use case specifications of the use cases shown in the fragment above are presented next. In the specification I use "session" as a mechanism of passing data between use-case so that the flow of ProcessShoppingListDelivery becomes dependent on the outcomes of the extension use cases. Modelling session as a lifeline in the sequence diagram is NOT expected.

Use case ID: 1	**Use case:** ProcessShoppingListDelivery
Brief Description: This use case defines the interaction between a user and a smart fridge when a delivery has taken place from the preferred supermarket. The system matches the delivery to a shopping list (e.g. searching by the shopping list id provided by the user) held on SFXfridge. The barcode of each delivered item is then scanned and the inventory – updated. If an item cannot be located on SFXfridge by its barcode, the fridge retrieves the details of the item from the cloud storage using the barcode as a search criterion. Once all delivered items have been processed, the fridge archives the ShoppingList to the cloud storage and removes it (the list itself and the items on it) from the memory of SFXfridge.	
Primary actors: User	
Secondary actors: EasyLiving cloud storage.	
Preconditions: 1) System is operational.	

2) A ShoppingList, held in the memory of the fridge, has been submitted to a supermarket for delivery.
3) The User has logged in from the LCD panel of SFXfridge and has chosen from the menu the option “Process Shopping List Delivery”.

Main flow:

1) The use case starts when the User selects the "ProcessShoppingListDelivery" option from the user menu.
2) The System shows a form on LCD screen and prompts the user to specify the id of the ShoppingList, which the user wants to process.
3) The user enters a ShoppingList id.
4) The System establishes a connection with the EasyLiving cloud storage.
5) The System locates the Shopping List and prompts the user via the LCD screen to start scanning the barcodes of the delivered items.
6) For each delivered item:
   6.1. The user scans the barcode of the item.
   6.2. The System stores the barcode in session.barcode
   6.3. The system sets session.matchingFound = false
   6.4. For each item held in the memory of the fridge:
      6.4.1. The System places the current item in session.item
      6.4.2. The System compares the barcode of the item pointed to by session.item with the barcode held in session.barcode
      extension point : ItemFound
      6.4.3. Continue (the loop)

   extension point: NoItemWithMatchingBarcode
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.
11) The system deletes session.item, session.barcode and session.matchingFound.
12) The system informs the user that processing the selected shopping list is complete.

Postconditions:

1) The content of the fridge has been updated.
2) A copy of the ShoppingList has been stored in the CloudStorage
3) The chosen ShoppingList (together with all items on the list) has been removed from the memory SFXfridge.
4) session.item, session.barcode and session.matchingFound have been erased from SFXfridge.

Alternative flows:
ListNotFound
NoCommunication

Use case ID: 1.1	**Use case:** ProcessShoppingListDelivery : ListNotFound
Brief Description: ShopingList with provided id not found.	
Primary actors: User	
Secondary actors: EasyLiving cloud storage.	

Preconditions:

1) System is operational. The User has logged in via the LCD panel, has started to process a Shopping List Delivery and typed in a Shopping List id.

MainFlow:

5.1) The flow starts at step 5 of the main flow when the search for a ShoppingList with a given id fails to locate a ShoppingList with a matching id.
5.2) The System informs the User that a Shopping list with an id equal to the one provided by the user cannot be found on the SFXfridge.
5.3) The user acknowledges the notification.
5.4) The System shows the main menu on LCD panel.

Postconditions:

None.

Use case ID: 1.2	**Use case:** ProcessShoppingListDelivery : NoCommunication

Brief Description: No communication with Cloud Storage can be established.

Preconditions:

1) System is operational. The User has logged in via the LCD screen and has started a Shopping List Delivery processing.

Main flow:

4.1) The flow starts at step 4 of the main flow.
4.2) The system informs the User that no connection with EasyLiving cloud storage can be established.
4.3) The user acknowledges the notification.
4.4) The System shows the main menu on LCD screen.

Postconditions:

None.

Use case ID: 2	**Use case:** ItemFound

Brief Description: The system finds an item on SFXfridge with barcode matching the scanned barcode. The system ask the User to type in the amount in the delivered package and updates the quantity of the item held in SFXfridge.

Primary actors: User

Secondary actors: None.

Preconditions:

1) System is operational. The User has logged in via the LCD screen, has started a Shopping List delivery processing.
2) A ShoppingList with a matching id has been identified.
3) A barcode is held in session.barcode.
4) A reference to an item on SFXfridge with matching barcode is held in session.item.

Main flow:

1) The System prompts the user to type in the quantity of the goods held in the delivered package.
2) The User types in the quantity.
3) The System updates the quantity of the item pointed to by session.item with the amount provided by the User.
4) The System sets session.matchingFound = true.

Postconditions: 1) 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. 2) session.matchingFound set to “true”.
Alternative flows: None.

Use case ID: 3	**Use case:** NoItemWithMatchingBarcode
Brief Description: This use case specifies system behavior in case a scanned item cannot be located in SFXfridge. The system connects with the cloud storage and retrieves from there the details of the item with a barcode equal to session.barcode and stores in it the details from the delivery.	
Primary actors: User	
Secondary actors: EasyLiving cloud storage.	
Preconditions: 1) System is operational. The User has logged in via LCD screen and has started a Shopping List Delivery processing. 2) A ShoppingList with a matching id has been identified. 3) session.matchingFound = false. 4) The item barcode is held in session.barcode.	
Main flow: 1) The System submits a query to EasyLiving cloud storage for locating an item with a barcode matching the barcode held in session.barcode. 2) The cloud service locates an item with a matching barcode and returns its details to the System. 3) The System create a new item using the details (description, name, number) provided in the response of the cloud storage. The attribute “qty” of the newly created item is set to 0. 4) The System stores in session.item a reference to the newly created item. 5) The System asks the User to type in the quantity of the item (with a barcode matching the barcode of the last scanned item). 6) The user types in the amount. 7) The system updates the quantity of the item pointed to by session.item.	
Postconditions: 1) A new item is created matching the barcode held in session.barcode. 2) A reference to this newly created item is held in session.item.	
Alternative flows: None.	

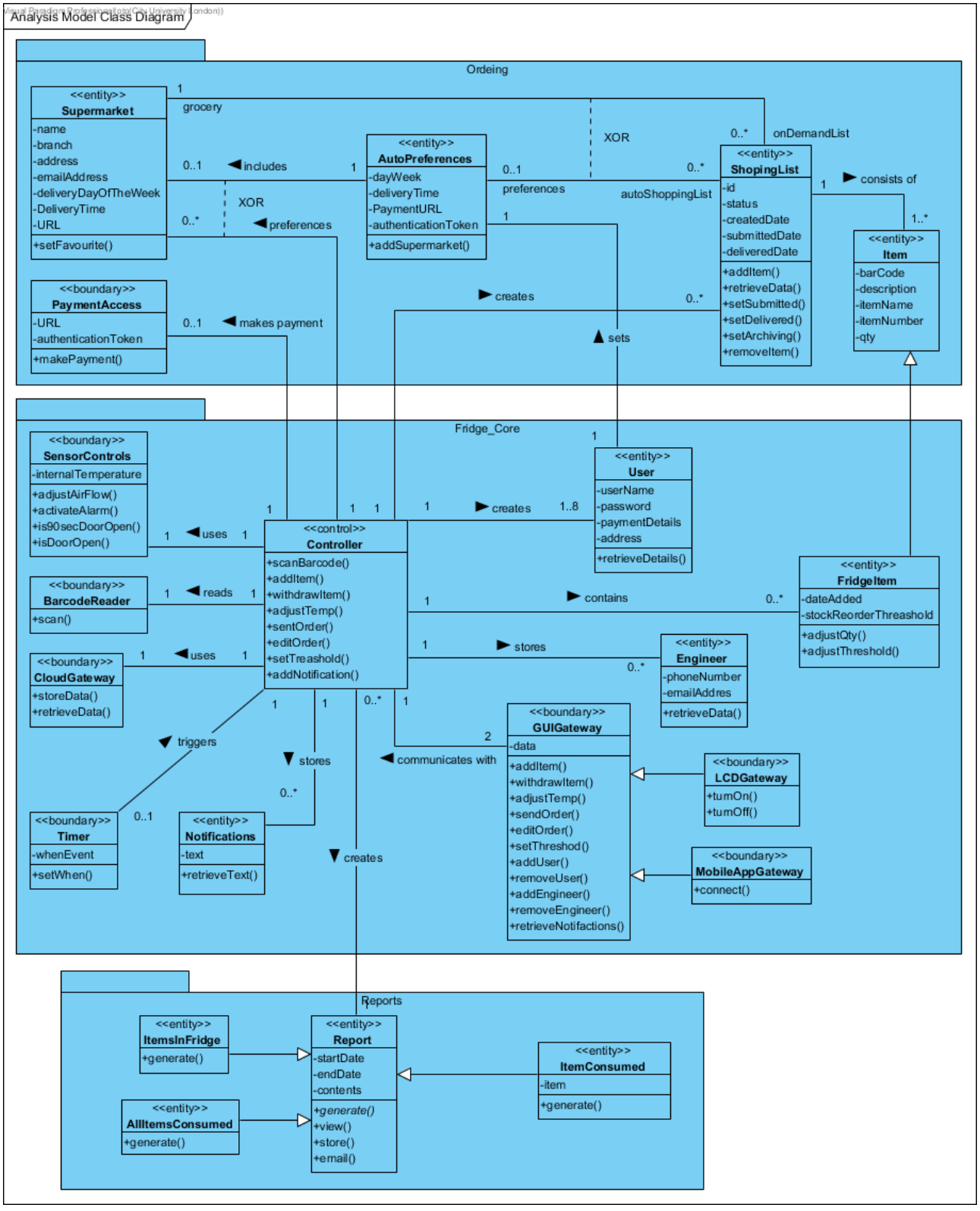
Analysis class diagram

The initial analysis class diagram is provided below. The analysis class model is also available on Moodle as a Visual Paradigm project.
The diagram includes:

- a number of boundary classes to allow interaction of SFXfridge with a number of actors, as defined in the case study;
- A single control class, Controller, is included which coordinates the interaction of the

other classes.

- A number of entity classes captures the main abstractions from the problem domain.



For Q1 you use the entire diagram – most of the lifelines belong to Ordering package, but some – to other packages (e.g. the boundary classes LCDGateway, CloudGateway and BarcodeReader and the Controller class).

For Q2 you only work with package Ordering.

Submission guidelines

1) Submissions can **only** be made **electronically** via **Moodle**, using the coursework submission area for the IN2013 module.

2) The **deadline** for submission is **23.55 on Sunday, the 11th of November 2018**.

3) Moodle will adhere to the cut-off date/time and automatically prevent you from attempting to submit your work after the deadline.

4) I suggest that you do not even attempt to work right up to the deadline and instead recommend that you get your submission in well before the cut-off time. The last thing you need is the stress and worry of watching the clock tick and then encountering a problem that delays you. It can and does happen!

Late submission policy

In accordance with the usual policy on coursework submission, **late submission will receive 0%** unless there are **extenuating circumstances with supporting evidence**, which must be notified to the Course Officer in **advance of the deadline**.

Feedback

We will aim to get coursework marked and returned to you by the end of Week 10. General feedback will be provided during the revision lecture in Week 10, and you will also receive individual feedback and a provisional mark via the grade book in Moodle.

Format

All UML diagrams must be created using a UML tool and exported as images for subsequent inclusion in the submission document. Diagrams drawn without a tool will be penalized by ***deducting 20%*** from the awarded mark.

Your coursework must be submitted as a **single** PDF file. Export your diagrams from the UML tool you have used as images, and then assemble all your answers, text and diagrams, in a single word processor file and convert to PDF. Make sure you diagrams are eligible.

You may also submit your entire project if you have used Visual Paradigm for the diagrams.

UML Diagrams

UML tools are available on PCs in the laboratories, and you can also download a copy for your personal machine (Windows, Mac, or Linux) by following the instructions on Moodle.

Note that you may come across variations in UML syntax on websites and within certain textbooks. This is of no consequence for the purposes of this coursework.