Problem 1: Grocery Cooperative
Domain Model, OCL invariants and functions (40%)

The application is about a grocery cooperative: a union of grocery retailers are cooperating when purchasing goods in order to get better conditions from the suppliers. The cooperative buys products, each characterized by a name and a unit size (measured in integral steps of cm^3), from suppliers. A supplier is known by its name and address. Each supplier has its own price for a given product. Goods are delivered by the supplier to one of the warehouses managed by the cooperative.

The cooperative fixes its own retail prices. A retailer, known by its name and address, orders from the cooperative the products in the quantities it needs. Retailers have accounts with the cooperative. The amount a retailer must pay is determined when the order is placed, but only when the goods were shipped and delivered to the retailer the account of the retailer is charged. The cooperative also allocates credit limits to its members. A retailer cannot overdraw this credit limit by an order. Whenever the quantity of a product in stock falls below a certain limit (as a consequence to an order by a retailer), an order to replenish the stock is sent to the supplier that offers the best price for that product.

Task 1.1

Devise a domain model that models the concepts of the grocery cooperative.

Problem 2: Ticket Vending Machine Environment Model

Informal Description:

The system for which requirements are to be gathered is an automated ticket vending machine like the ones you find for Montreal's subway system ("Le Metro") that allows users to upload tickets onto a chip card called "opus card". For simplicity reasons, we are going to focus only on the simple vending machines that only accept debit or credit card to purchase tickets (single fare, multi-fare, weekly and monthly tickets). A sketch of the input and output devices of the simple ticket vending machine is shown in figure 1. We are also going to assume that the "payment system", i.e., the software that handles the credit/debit card reader and PIN keyboard, is provided by some third-party vendor. In other words, the software we are developing does not need to communicate directly with the card reader, or have to deal with the details of how to handle credit/debit cards (entering and verifying PINs, connecting to credit and financial institutions to validate credit or debit money, etc.), but rather interacts with the payment system.

Figure 1: Simple Vending Machine

To use a ticket vending machine, the customer places the opus card into the smart card reader/encoder. The user can then consult the current tickets stored on the card, and is presented with a set of recharge options on the transaction console. The selector buttons are used to determine the desired choice. The user then interacts with the payment system to use the credit/debit card reader and PIN keyboard to pay for the selected ticket. If the payment completes successfully, the tickets are uploaded to the card and a receipt is printed. If payment was unsuccessful, the reason is displayed on the console and no tickets are issued. At any point in time before the payment is completed, the user can cancel his transaction by pressing the cancel button or simply removing his opus card from the smart card reader/encoder. Finally, during the interaction, the system beeps within 30 seconds in the case where the user does not make a selection, or forgets to remove his opus card from the smart card reader/encoder.

The use case *RechargeOpusCard* describing the complete interaction between the environment and the ticket vending machine software under development is presented below:
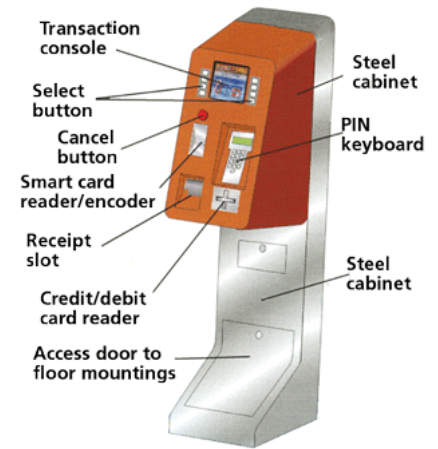
Use Case Model:

Use Case: RechargeOpusCard
Scope: TicketVendingMachine
Level: User-Goal
Intention in Context: The User wants to refill his OpusCard using a credit card.
Multiplicity: Only one User can recharge an opus card at a given time.
Primary Actor: *User*
Secondary Actors: *SmartCardRE*, *PaymentSystem*, *Printer*, *Speaker*
Main Success Scenario:
1. *User* notifies *System* that he wants to recharge his opus card.
2. *System* shows tickets that are currently on the card and recharge options to *User*.
3. *User* informs *System* of recharge choice.
   Step 4 and 5 can happen in any order.
4. *System* displays price of current choice to *User*.
5. *System* informs *PaymentSystem* of price of the ticket.
   User completes the transaction with the payment system.
6. *PaymentSystem* informs *System* of successful completion of the transaction.
7. *System* uploads tickets onto opus card using the *SmartCardRE*.
8. *System* prints receipt using *Printer*.
9. *System* asks the *User* to collect the receipt and remove opus card.

Extensions:

2-6a. *User* informs *System* that he wants to cancel the transaction. Use case ends in success.

3a. Timeout

  3a.1. *System* asks *Speaker* to beep. Use case continues at step 3.

6a. *PaymentSystem* informs *Systems* that payment was unsuccessful.

  6a.1. *System* informs *User* about failed transaction. Use case continues at step 3.

10a. Timeout

  10a.1. *System* asks *Speaker* to beep. Use case continues at step 10.

Task 2

Elaborate a *low-level* Environment Model for the TicketVendingMachine here. Low-level means that you should show that the software interacts with the hardware devices. In other words, you are not allowed to send or receive messages directly from the User. Provide type and message definitions (with parameters) on the next page.

Type and Message Definitions:

Problem 3: Parking Garage Use Case

The following is an informal description of how an automobilist interacts with a parking garage control system (PGCS) when parking his car. The function of the PGCS is to control and supervise the entries and exits into and out of a parking garage. The system ensures that the number of cars in the garage does not exceed the number of available parking spaces.

The entrance to the garage consists of a gate, a state display showing whether any parking space is available, a ticket machine with a ticket request button and a ticket printer, and an induction loop (i.e., a device that can detect the presence or absence of a vehicle). To enter the garage, the driver, receives a ticket indicating the arrival time upon his request. The gate opens after the driver takes the ticket. The driver then parks the car and leaves the parking garage. In case of problems, the PGCS notifies an attendant by means of an attendant call light.

Task 3

Elaborate the *EnterGarage* use case, which describes all the interaction steps between the system and the environment that occur when a driver enters the garage. You do not have to specify what happens when paying for parking and exiting the garage. You also do not have to specify the interactions that the attendant has with the PGCS when servicing the system or when handling problems.

You are asked to write the *EnterGarage* use case at a low-level of abstraction, clearly stating which hardware devices the software is interacting with.