Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Prof. Elisabetta Di Nitto, Matteo Rossi and Damian Tamburri

20133 Milano (Italia)
Piazza Leonardo da Vinci, 32
Tel. (39) 02-2399.3400
Fax (39) 02-2399.3411

Software Engineering 2 – Written Exam 2 (WE2)

January 24st, 2022

Notes

1. Remember to write your name and Id number (matricola) on each piece of paper that you hand in.
2. You may use a pencil.
3. Incomprehensible handwriting is equivalent to not providing an answer.
4. The use of any electronic apparatus (computer, cell phone, camera, etc.) is strictly forbidden, with the exception of an ebook reader.
5. The exam is composed of 2 parts, one focusing on requirements, and one focusing on design. Read carefully all points in the text!
6. **Total available time for WE2: 1h and 30 mins**

System Description: PaasPopCoin

The private security and event organisation company "HSG" from The Netherlands wants to build an application---**PaasPopCoin**---that handles the coin emission and transactions in the scope of a medium-size music festival they are organizing. The goal of the system is to allow festival-goers and operators to spend an allotted amount of money in relative safety and without the need to bring wallets and other assets around the event. The software in question needs to handle at least three scenarios:

1. Emission of coins in exchange of money through appropriate cashier desks and Automated Teller Machines (ATMs).
2. Cash-back, that is, exchange of coins with cash in the same locations (we assume that people at the festival may be willing to receive back the money corresponding to the coins they have not used).
3. Tracking of coin expenditure transactions at the various festival shops.

In the scope of the above scenarios, there are several special conditions to be considered.

First, in the scope of coin emissions, there exist four classes of coin "buyers", namely: (a) VIPs (e.g., event artists, shop-owners) who receive a 30% discount on the coins they buy; (b) event organisation people (e.g., security guards, event managers) who receive a 50% discount; (c) event ticket holders class A, who receive a 20% discount; (d) finally, regular ticket holders who receive no discount. When buying coins, users first need to authenticate themselves by inserting their own ID card in the ATM or by giving it to the cashier; this allows the system to determine the class to which each coin buyer belongs. After authentication, buyers get the coins upon inserting into the ATM or giving to the cashier the corresponding amount of money.

Second, also in the context of cash-back users need to authenticate with their ID card to make sure the appropriate amount of money is given back, considering their role and privileges.

Third, during the event, every shop clerk keeps track through the *PaasPopCoin* system of the sales of products and the coins received.

PaasPopCoin relies on a third-party analytics service to periodically check whether the festival is earning money or not (cost-benefit analysis). Such check is performed with respect to costs of products being sold during the event, as well as the overhead to cover all event organisation and management expenses. Note that reasoning and evaluating these aspects is beyond the scope of the exercise.

Part 1 Requirements (7 points)

RASD_Q1 (2 points)
With reference to the Jackson-Zave distinction between the world and the machine, identify the relevant world phenomena for *PaasPopCoin*, including the ones shared with the machine, providing a short description if necessary. For shared phenomena specify whether they are controlled by the world or the machine. Focus on phenomena that are relevant to describe the requirements of the system.

RASD_Q2 (3 points)
Describe through a UML Class Diagram the main elements of the *PaasPopCoin* domain.

RASD_Q3-Q4 (2 points)
Define a UML Use Case Diagram describing the relevant actors and use cases for *PaasPopCoin*.
You can provide a brief explanation of the Use Case Diagram, especially if the names of the use cases are not self-explanatory.

Part 2 Design (7 points)

DD_Q1 (2 points)
Assuming you need to implement system *PaasPopCoin* analyzed above, identify the most relevant components and interfaces and describe them through UML Component or Class Diagrams.

DD_Q2 (3 points)
Provide a brief description of each component identified in DD_Q1 and list the operations provided by the corresponding interfaces.
For each operation, you do not need to precisely specify its parameters; however, you should give each operation a meaningful enough name to understand what it does; you can also briefly describe what information operations use/produce.

DD_Q3 (2 points)
Describe through a UML Sequence Diagram the interaction that occurs between the system components when the user asks for cash-back through a cashier.

Solutions

RASD_Q1

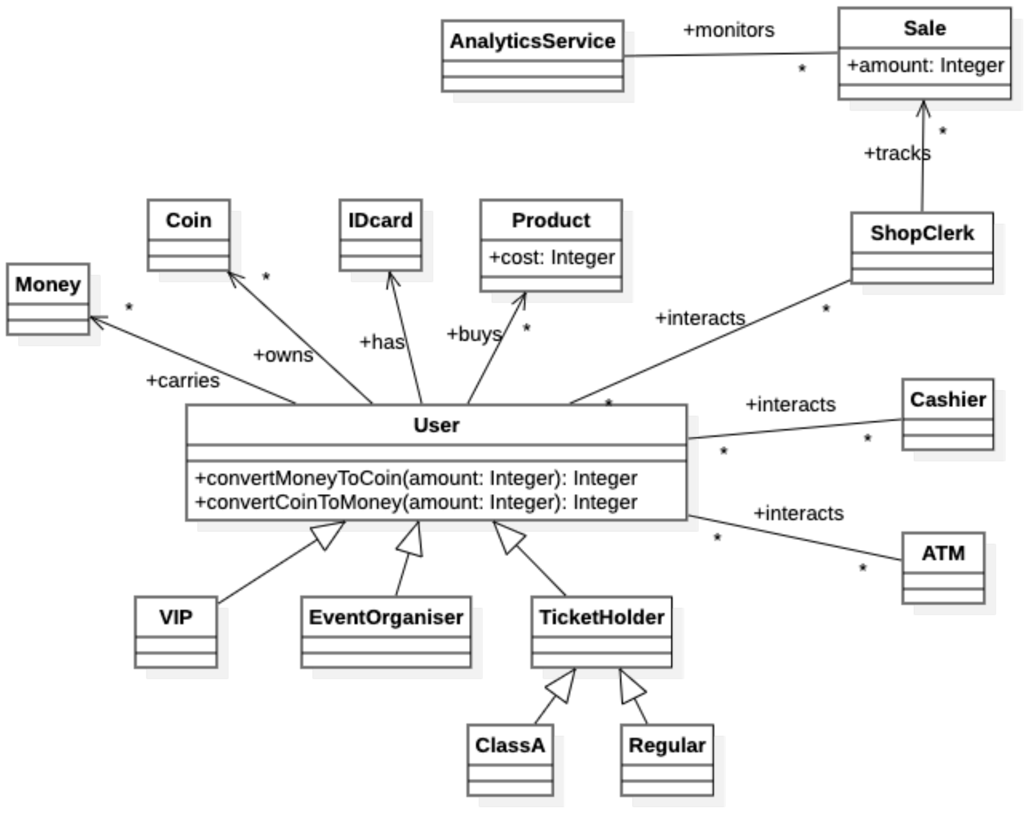
World phenomena:
User buys Class A ticket
User buys regular ticket
VIP is contracted for event
Event organization is started and contractors registered
Event starts
User gives money to cashier (to be converted in coins)
User gives coins to cashier (to be converted in money)
User gives ID card to cashier
User buys some product at festival
The external analytics service checks the success of an event

Shared phenomena, world-controlled
User inserts money into an ATM machine
ID Card is inserted into ATM
User inserts coins into an ATM
Cashier inserts in the system an ID card number
Cashier inserts in the system the amount of money handed by a certain user
Cashier inserts in the system the amount of coins returned by a certain user
Store clerk inputs in system the amount of coins spent by user in shop

Shared phenomena machine-controlled
The system checks ID card and inserted amount of money and enables coin exchange
The system checks ID card and inserted number of coins and enables cash-back
The system sends data about purchases to the external analytics service

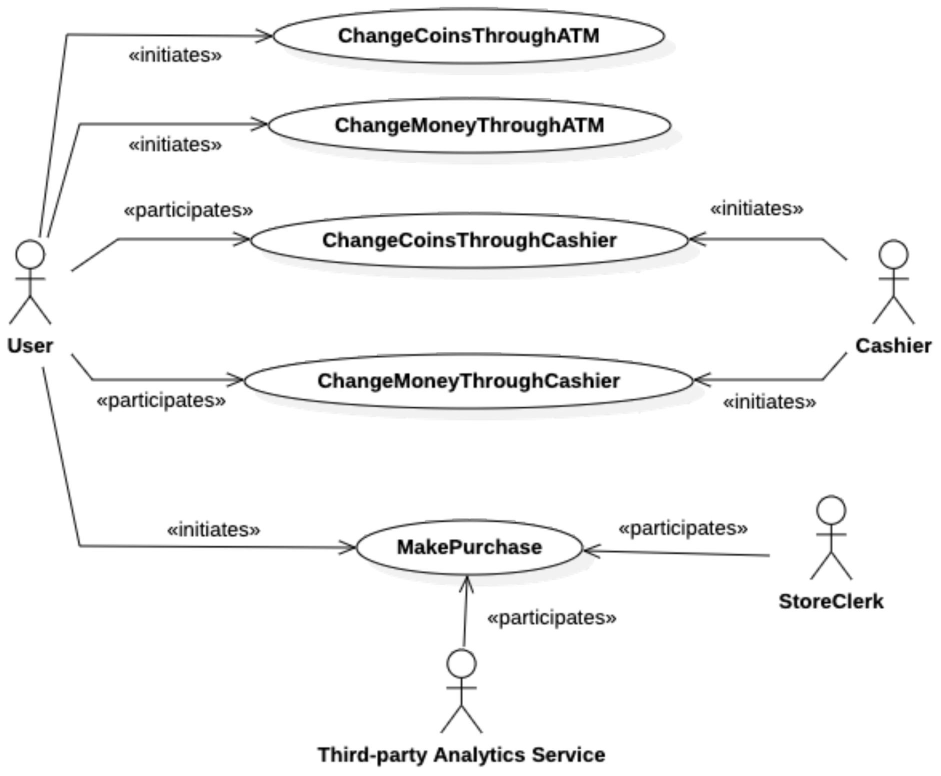
RASD_Q2

A possible domain model for the PaasPopCoin system is the following:



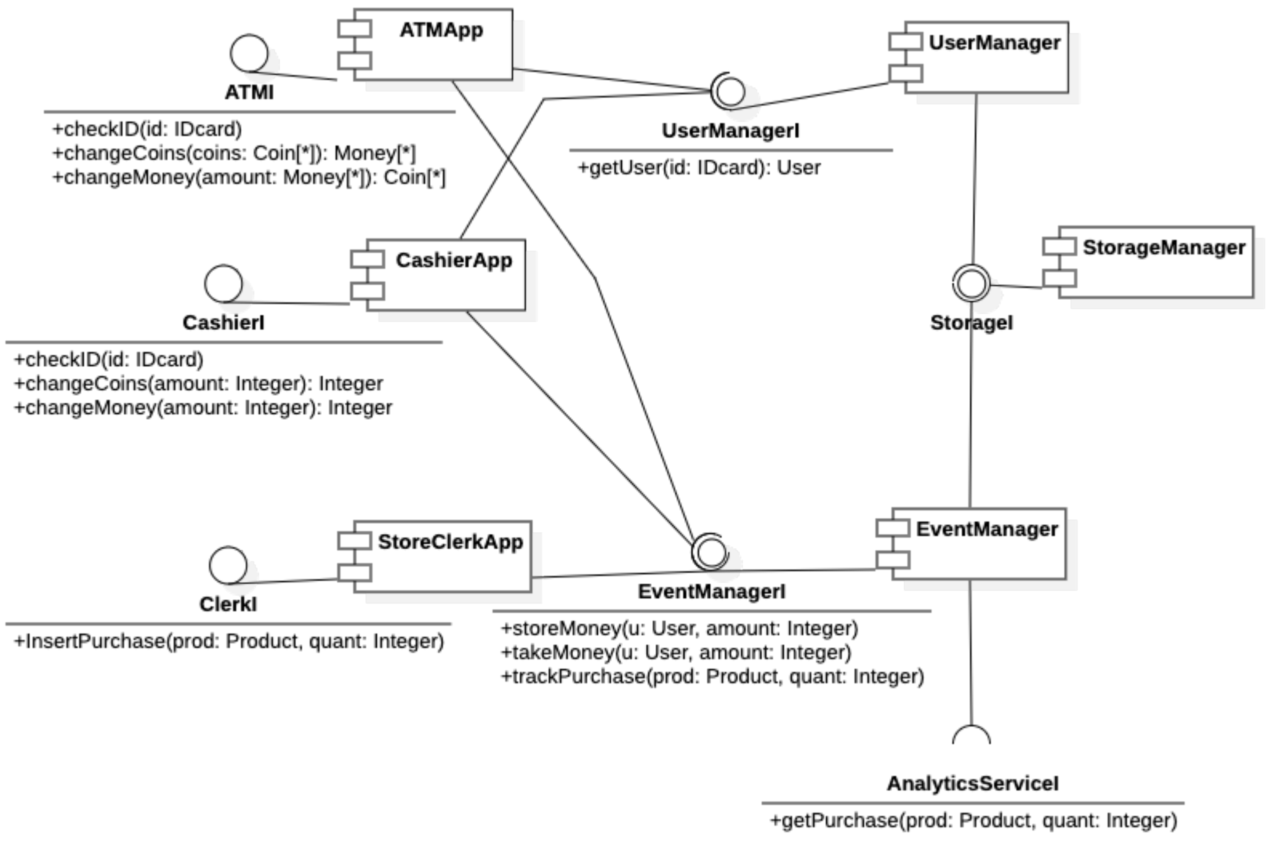
Notice that *ShopClerk* and *Cashier* could be considered as specializations of *EventOrganizer*.

RASD_Q3



Notice that the use cases for the conversions money→coins and coins→money are separate depending on whether the exchange occurs through the ATM or with the cashier, because the interactions in the 2 cases are different (indeed, in the second case it is the cashier who interacts with the system, rather than the customer). In addition, in this design the idea is that the external analytics service monitors purchases (as highlighted in the domain model), which means that it receives information when purchases are made, hence it participates in the *MakePurchase* use case. Other solutions and designs are possible.

DD_Q1


ATMApp
ATMI
+checkID(id: IDcard)
+changeCoins(coins: Coin[*]): Money[*]
+changeMoney(amount: Money[*]): Coin[*]
UserManager
UserManagerI
+getUser(id: IDcard): User
CashierApp
CashierI
+checkID(id: IDcard)
+changeCoins(amount: Integer): Integer
+changeMoney(amount: Integer): Integer
StorageManager
StorageI
StoreClerkApp
ClerkI
+InsertPurchase(prod: Product, quant: Integer)
EventManager
EventManagerI
+storeMoney(u: User, amount: Integer)
+takeMoney(u: User, amount: Integer)
+trackPurchase(prod: Product, quant: Integer)
AnalyticsServiceI
+getPurchase(prod: Product, quant: Integer)

DD_Q2

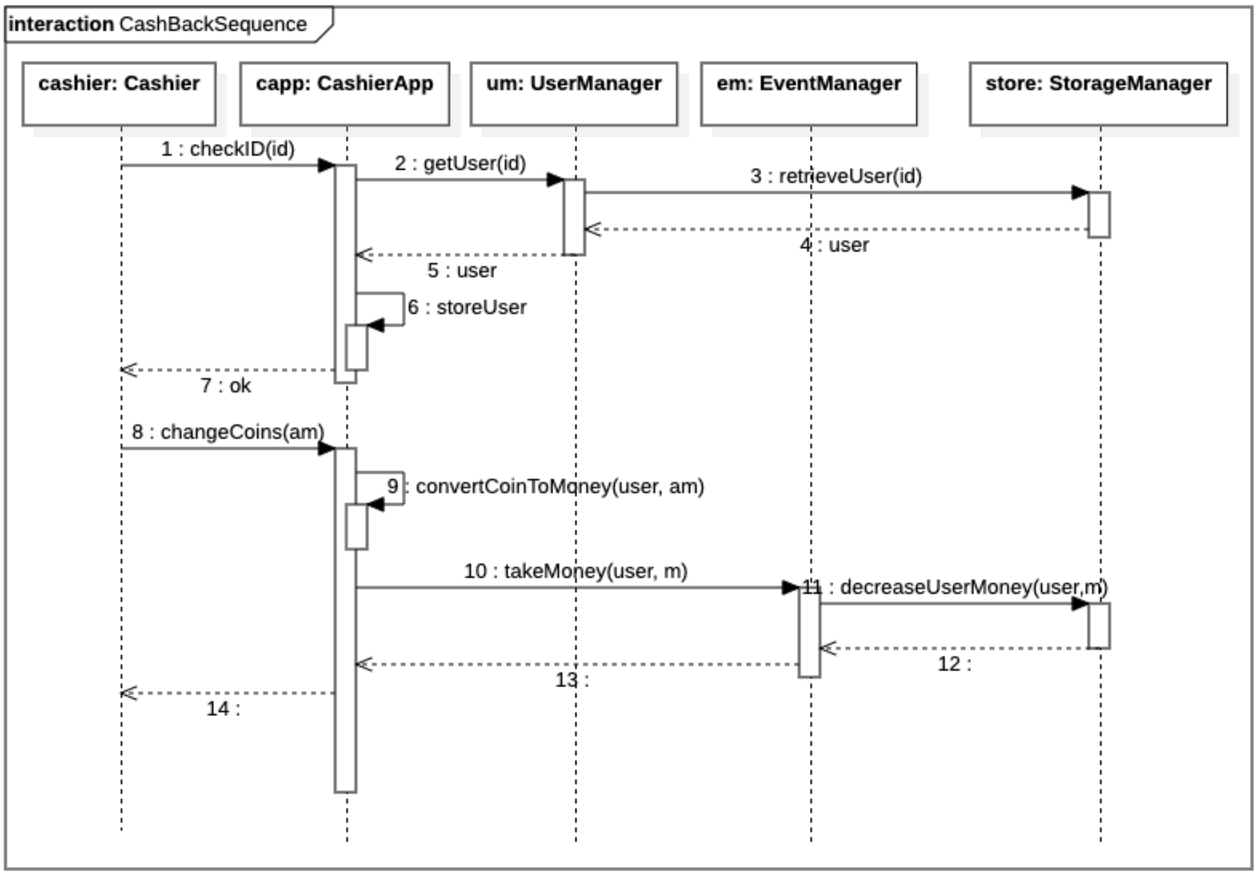
There are 3 different modules for the front-end of the applications: one to handle the interaction of the customer with the ATM (ATMApp), one to handle the interaction of the cashier with the system (*CashierApp*), and one to handle the interaction of the store clerk with the system (*StoreClerkApp*). The first two modules (*ATMApp* and *CashierApp*) provide similar functions, but they are separate modules, because they are realized (and deployed) differently, given the different users.

Module *UserManager* handles the checks of the user ids. Notice that user information is available in the system, but this information is not created by PaasPopCoin. Indeed, users do not register to the PaasPopCoin system, they buy tickets to the festival, but this is not a function that is covered by the PaasPopCoin (it occurs through some other system). So, we assume that customers are already known by the system, and *UserManager* only needs to provide a mechanism to verify the user (notice that class *User*, as shown in the Class Diagram, provides the function to compute the conversions).

Modules *ATMApp* and *CashierApp* compute the conversions (through class *User* mentioned above); in the first case, the module returns the right amount of coins, whereas in the second case the module returns the number of coins that the cashier should give back to the customer. Both modules use component *EventManager*, which is in charge of keeping track of the various transactions that occur at the festival (coin/money exchanges, and purchases). The relevant data (users, transactions, etc.) are stored in a suitable storage, which is accessed through the *StorageManager* module. The *StorageManager* module provides all necessary functions to create, modify, delete the data (these functions are not detailed in the diagram for simplicity, there are several functions for each type of information handled by the *StorageManager*). The *EventManager* sends the information about purchases also to the external analytics service, which is assumed to provide an interface (*AnalyticsServiceI*) for this purpose.

The lists of operations are directly provided in the diagram (except for the *StorageManager*, as explained above)

DD_Q3



Notice that the interaction occurs through the Cashier, who is the actor that actually uses the system. The interaction is made of 2 parts: the authentication part (in which the cashier checks through the system the id of the customer), and the conversion part. Notice that in the second part the system returns the amount of money to be handed to the customer, but the actual transfer of money from the cashier to the customer is not represented in this diagram, because it occurs outside of the purview of the system.