Seminar 1

Object-Oriented Design, IV1350

Alexander Lundqvist

Alexlu@kth.se

03-22-2021

Contents

1 Introduction

The purpose of this seminar task is to give students an introduction into an UML modeling tool of their choice and give them practical knowledge in the process of creating domain models (DM) and system sequence diagrams (SSD).

This will be realized with the task of creating a domain model in conjunction with a system sequence diagram from a requirements specification.

2 Method

For the process of creating the DM we must have suitable candidates for the classes that the domain model is going to be based on

First a noun identification was performed and the nouns that were found was put into the following table.

Sale	POS	Store	Customer
Goods	Cashier	Item	Identifier
Program	VAT	Description	Inventory
System	Price	Total	Payment
Cash	Amount	Information	Receipt
Change	Quantity	Discount	Request
Rules	Registry	Log	Date
Time	Name	Address	ID

Table 2.1: Table of potential class candidates.

The second part of the analysis was to find additional class candidates using a category list. Some of the class candidates found could be created by combining some of the candidates from the noun identification. Since the noun identification was thorough the category list became somewhat short.

Category	**Class candidates**
Transactions	
Products, services	
Roles, people, organizations	
Places	StoreAdress
Records	SalesLog
Events	DiscountRequest
Physical objects	
Devices	Printer
Descriptions	ItemDescription

Catalogs	InventorySystem
Systems	AccountingSystem, InventorySystem
Quantities, units	Currency, TotalPrice, TotalVat
Resources	

Table 2.2: List of additional potential class candidates.

The potential class candidates were then inserted into an Astah project. The next step was to determine which of the candidates that would be more suited as attributes and if any of them were unnecessary. Since we can assume that the store is only accepting the local currency, that candidate could be excluded. Goods and item also got changed to GroceryItem since it was a more fitting name. Program also got discarded as per the seminar requirements. The rest of the changes are shown in the the following picture.

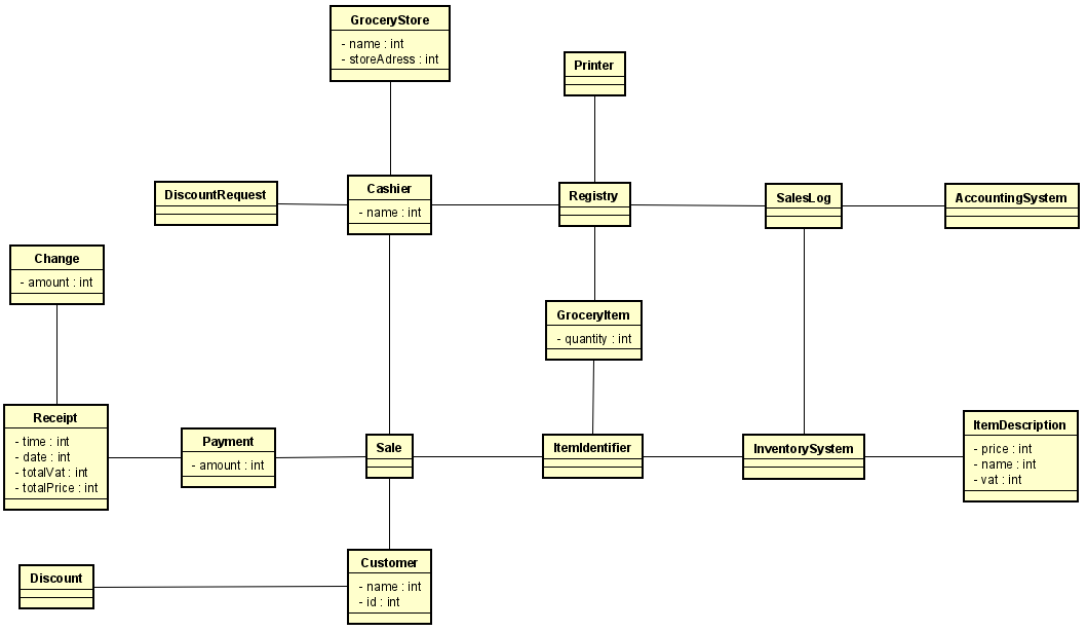

Figure 2.1: First iteration of the domain model.

Lastly the classes got appropriate associations except for address since it could arguably be redundant to associate it. A few modifications were made to the diagram because of layout issues and classes that were missing or redundant.

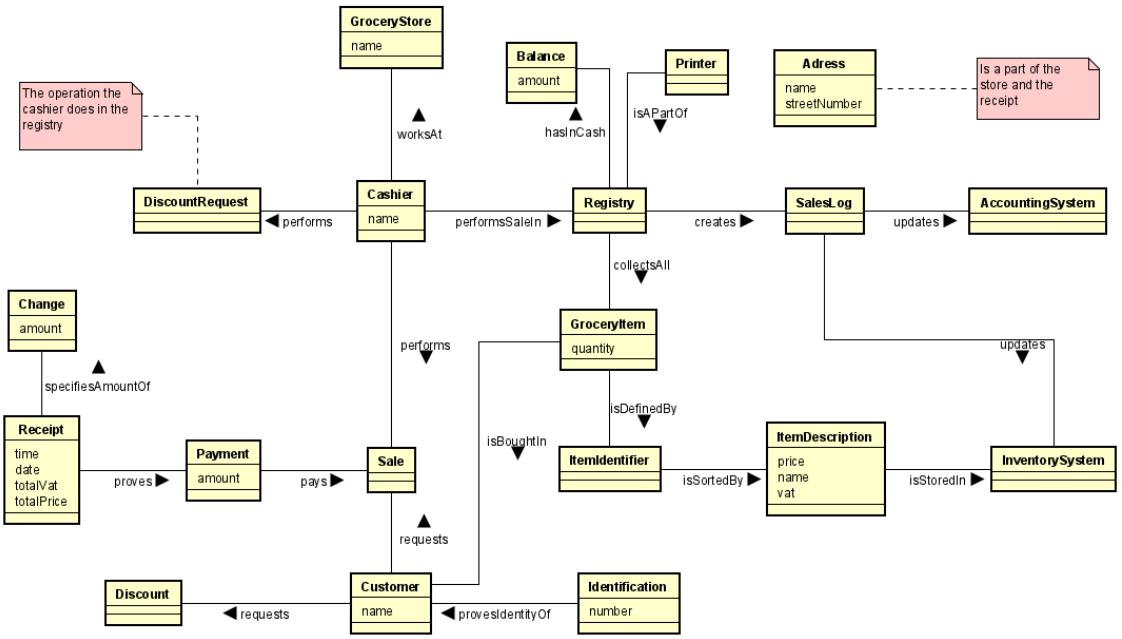

Figure 2.2: The completed domain model with associations.

When it came to the system sequence diagram the first step was to open a sequence diagram in Astah. The first step was to draw the basic lifelines for cashier and system. A lifeline for the printer was also created as it was deemed relevant.

The second step was to go through the requirements specification to figure out how the basic flow looked like. The first iteration of the SSD is described in figure 2.3.

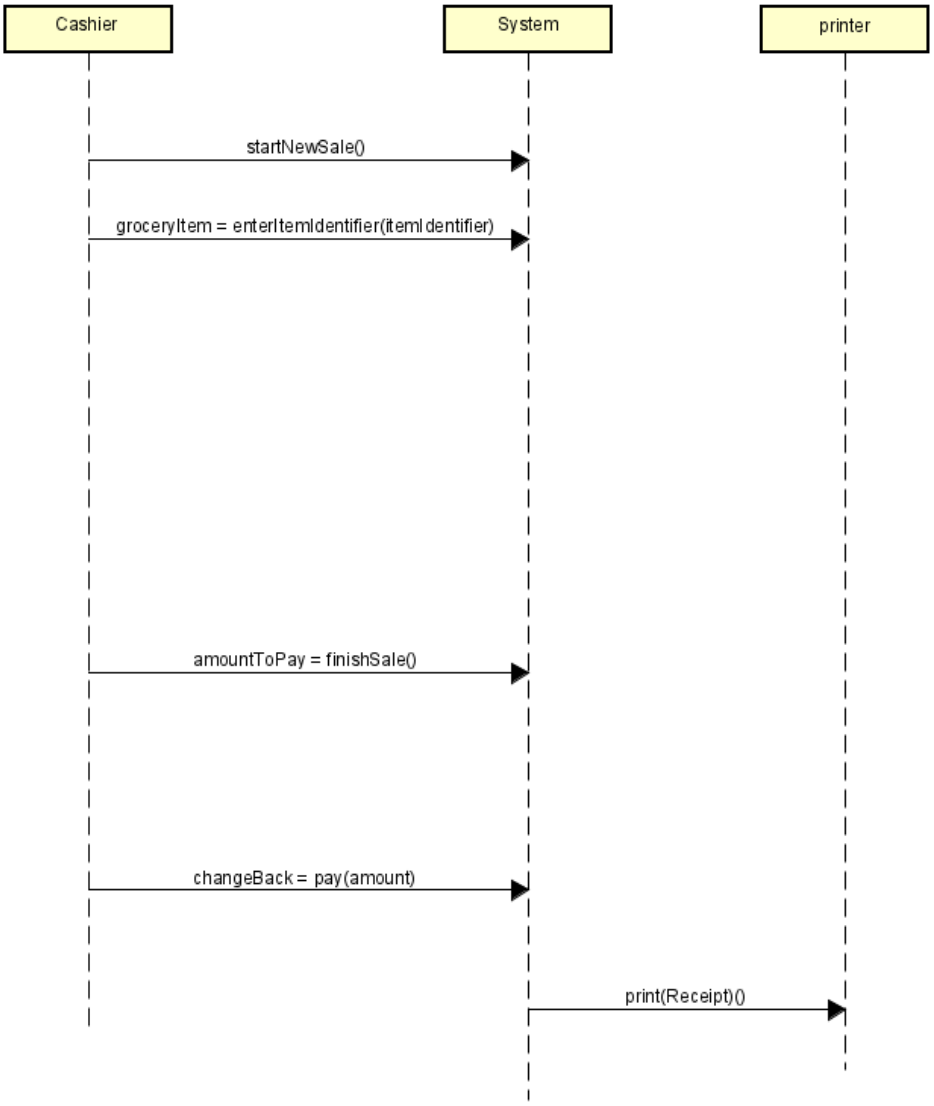

Figure 2.3: First iteration of the system sequence diagram.

The next part was to consider the alternative flows. To start with, a structure for the potential interactions was made. Through some thought and trial and error, five segments were created. The basic process was that the cashier just scanned items until there were no left. The cashier also got the option to enter a desired quantity of the current item scanned.

The alternative flow where no item with the specified identifier was found was put into an own segment where the program would give an error to the cashier. The alternative flow where the item already existed in the sale was also put in an own segment.

The last two segments were used for the end part of the sale as the cashier could either apply a discount on the sale or just finish it. Some other changes were made to the SSD to give it a better representation of the sale. The resulting SSD is shown in result.

3 Result

The final DM. The DM shows the classes produced in the analysis with their respective attributes and associations. Both address and discount request got notes to clarify the reasoning behind them.

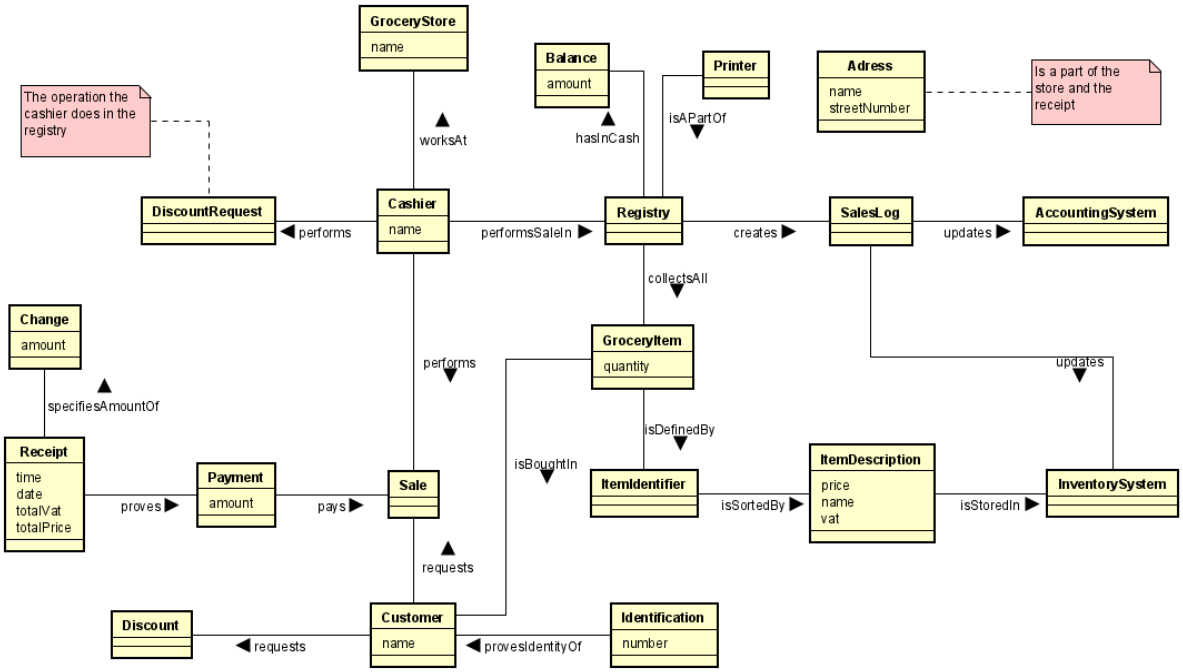

Figure 3.1: Final domain model.

The final SSD is described by figure 3.2. Here one can see how the basic flow is structured in combination with the alternative flow. The essential lifelines are the cashier and system, as they perform the most actions. The three notes were added as to explain more what the actions actually do.

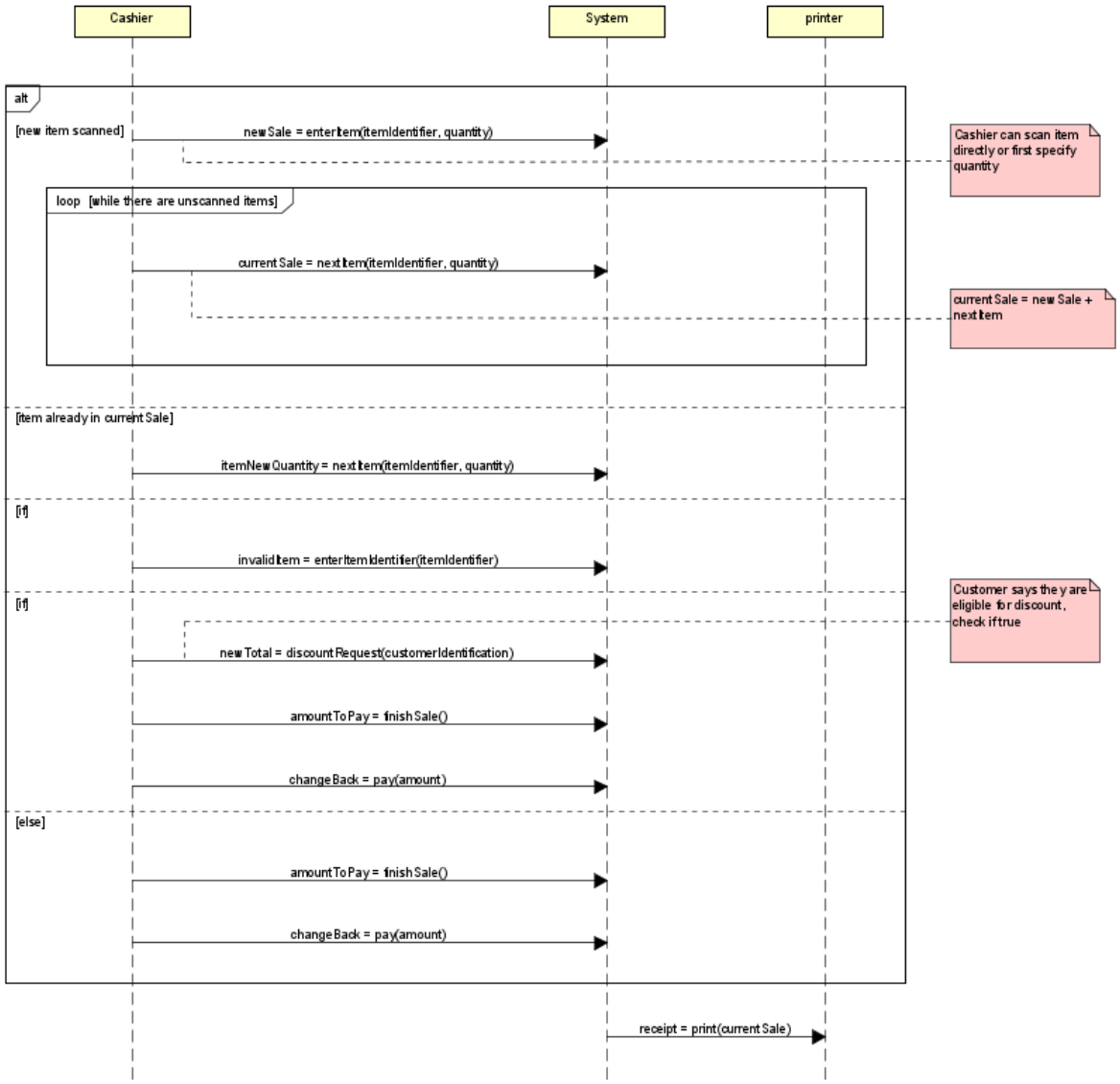

Figure 3.2: Final system sequence diagram.

4 Discussion

The above tasks is concluded and discussed in the following chapter.

The immediate thoughts about the domain model and the SSD is that they both seem to be correct from a pure technical standpoint. Both the DM and SSD follow the naming convention stated in the course literature. The model can be said to be a non-naïve and to most degree non-programmatic. There are however classes which are named AccountingSystem for instance, but my reasoning for these that the "system" could be simply a multidimensional array for instance, which indeed is a real world object as opposed to an abstract system. Furthermore, there aren't any "spider-in-the-web"-classes as far as I can see, even though both the cashier and registry have more than average relations.

We now look to the theoretical aspects of the DM and SSD. Regarding the DM, it can be discussed endlessly about what a correct DM should look like, as there aren't very many rights or wrongs regarding this subject. We can however debate whether the DM has sufficiently many classes or not, and to my understanding this model has enough classes to encapsulate the requirements specification. If there are any missing classes, I can not say as this is the first time for me doing such a model. For me, the model is explanatory enough, and since I made the model according to the method described in both the lectures and the course literature, I would argue that the choices of candidates, attributes and associations are satisfactory for the task specified.

I made the choice of not implementing any form of cardinality between the classes, mostly because I didn't find them necessary but also due to time constraints. Examples of where to implement such cases could've been between the Registry and GroceryItem.

As for the SSD, the choices of actors were based on the idea that it is only the cashier that uses the POS. As such, no internal system is described except for the system as it is necessary to describe the interactions. The third part of the SSD is the printer. As for more actors, it could be argued that I could've added more, such as display maybe or the accounting system. But I digress. For this task I made the choice of just including three actors, as I decided they were the most relevant. As for the operations I made the choice of not explicitly use operations and return values and instead used the more programmatic approach of assigning the return value directly to the operation. I made this choice as I found it more readable, but this can be argued to be the wrong approach. As an example, the operations between the printer and the system would be that the system calls for a receipt to be printed, but cash registers nowadays also displays the digital version of the receipt. Therefore the printer would print the receipt, and when that happens the system would display the digital receipt at the same time.

At the time of writing, I realize that the result chapter could've been deeper and explained more, but as there are images of the end results, I thought they we're self-explanatory. Any changes to the respective model before the seminar will be accounted for and explained during the seminar.