Formal modeling of the Kid2Kid Information System in VDM++

Formal Methods in Software Engineering
Master in Informatics and Computing Engineering

João Esteves - up201505145
Renato Campos - up201504942

January 6, 2019

1 Informal system description and list of requirements

1.1 Informal system description

This system models the Kid2Kid store franchise from an Information System perspective. Kid2Kid is made up of physical stores which buy and sell children items such as clothing from its clients.

In this system, an user can log itself as an Admin or as a local store's cashier. The Admin can manage the system's registered clients, stores and transactions, including but not limited to renaming clients, adding new store cashiers and editing a store's location. A store cashier is limited to the regular operations of a single store: buying and selling items, as well as seeing the products currently available and past transactions.

1.2 List of requirements

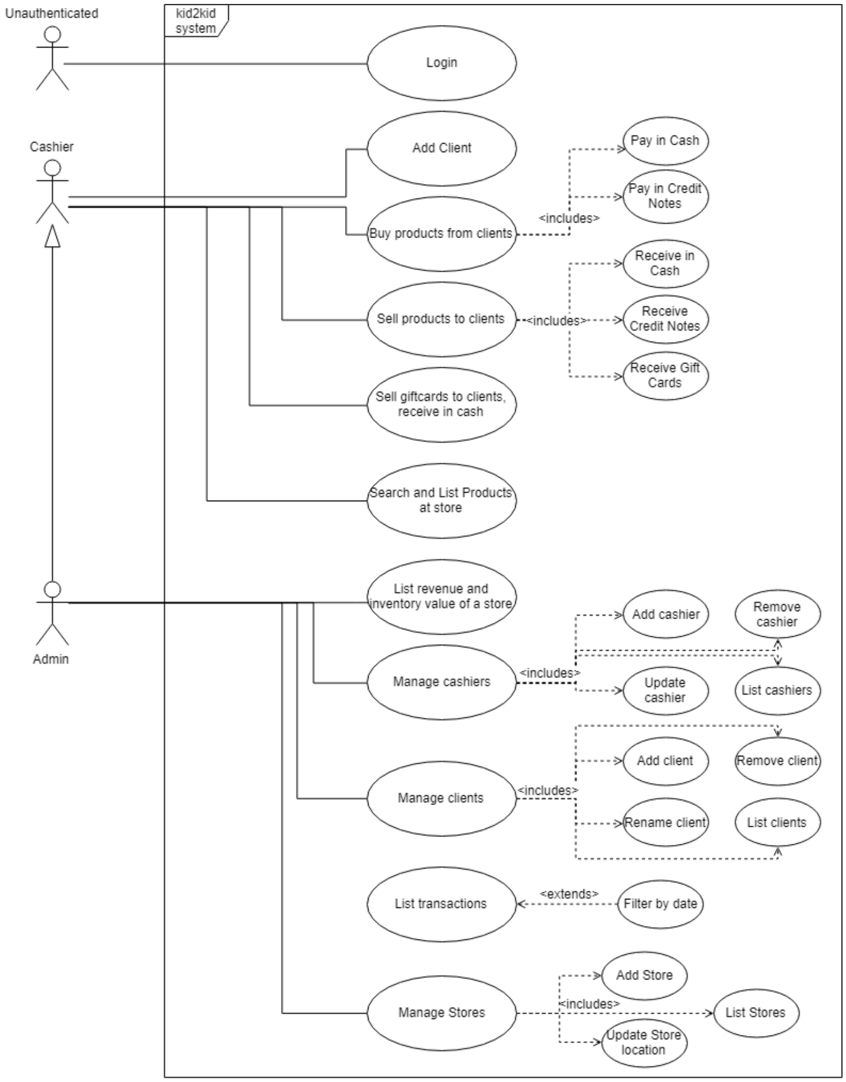
ID	Priority	Description
R1	Mandatory	An anonymous user should provide his account name to authenticate in the system.
R2	Mandatory	An admin can list all the clients.
R3	Mandatory	An admin can add a client.
R4	Mandatory	An admin can rename a client.
R5	Mandatory	An admin can remove a client.
R6	Mandatory	An admin can see the details of a client (name, id, products bought, products sold and gift-cards bought).
R7	Mandatory	An admin can list the transactions made in all the stores.
R8	Optional	An admin can filter transactions between 2 dates.
R9	Mandatory	An admin can list all the stores.
R10	Mandatory	An admin can add a new store.
R11	Mandatory	An admin can list the revenue and inventory value of all stores.
R12	Mandatory	An admin can manage cashiers (list, add, remove, and update).
R13	Mandatory	The cashier should be able to buy products from clients for the store with their category, age and usage state.
R14	Mandatory	The cashier should be able to sell products to clients and receive in cash, credit notes or gift cards.
R15	Mandatory	The cashier should be able to sell gift cards to clients.
R16	Mandatory	When a cashier buys a product to a client, the latter can be paid in cash or credit notes.
R17	Mandatory	A cashier should be able to add a new client to the system.
R18	Mandatory	A cashier should be able to search for products by category, age and usage state
R19	Mandatory	A cashier should be able to list all products in its store.
R20	Mandatory	An admin can do all the operations that a cashier can do.
R21	Mandatory	A cashier can edit the description of an existing product.
R22	Mandatory	A cashier can edit the sell price of an existing product.

1.3 Business Rules

ID	Description
B1	Each product is unique.
B2	A client interacts with a store through the cashier of that store.
B3	A store might have multiple cashiers.
B4	The client list is common to all stores.
B5	A gift card bought in 1 store can be used in another store.
B6	The sell price of a product must be 30% higher than the buy price.
B7	The credit note value should be 20% higher than the cash value when a client sells products to the store.
B8	There are multiple Kid2Kid stores each with its own products.
B9	Credit notes are directly connected to a store and can only be used in that store.
B10	To simplify, the price to be paid for each product is calculated as 10*PS. PS is product state: new = 1.0, low use = 0.8, high use = 0.5.
B11	A gift card can only be used once.

2 Visual UML model

2.1 Use Cases Model



Major use cases:

Scenario	**Sell products to clients in cash**
Description	Store sells a product in cash.
Pre-conditions	• Logged in user has cashier permissions • Client exists in the system • Cashier that authorizes the transaction exists in the system • Product exists in the store
Post-conditions	• Product is not available in the store • Product is saved as sold in the store • Product is saved as bought by the client • A sale transaction is created

Scenario	**Sell products to clients in credit notes**
Description	Clients can pay using credit notes bought in the same store
Pre-conditions	• Logged in user has cashier permissions • Client exists in the system • Cashier that authorizes the transaction exists in the system • Product exists in the store • Client has enough credit notes to buy the product
Post-conditions	• Product is not available in the store • Product is saved as sold in the store • CreditNotes of the client have decreased by the same amount as the sell price of the product. • Product is saved as bought by the client • A sale transaction is created

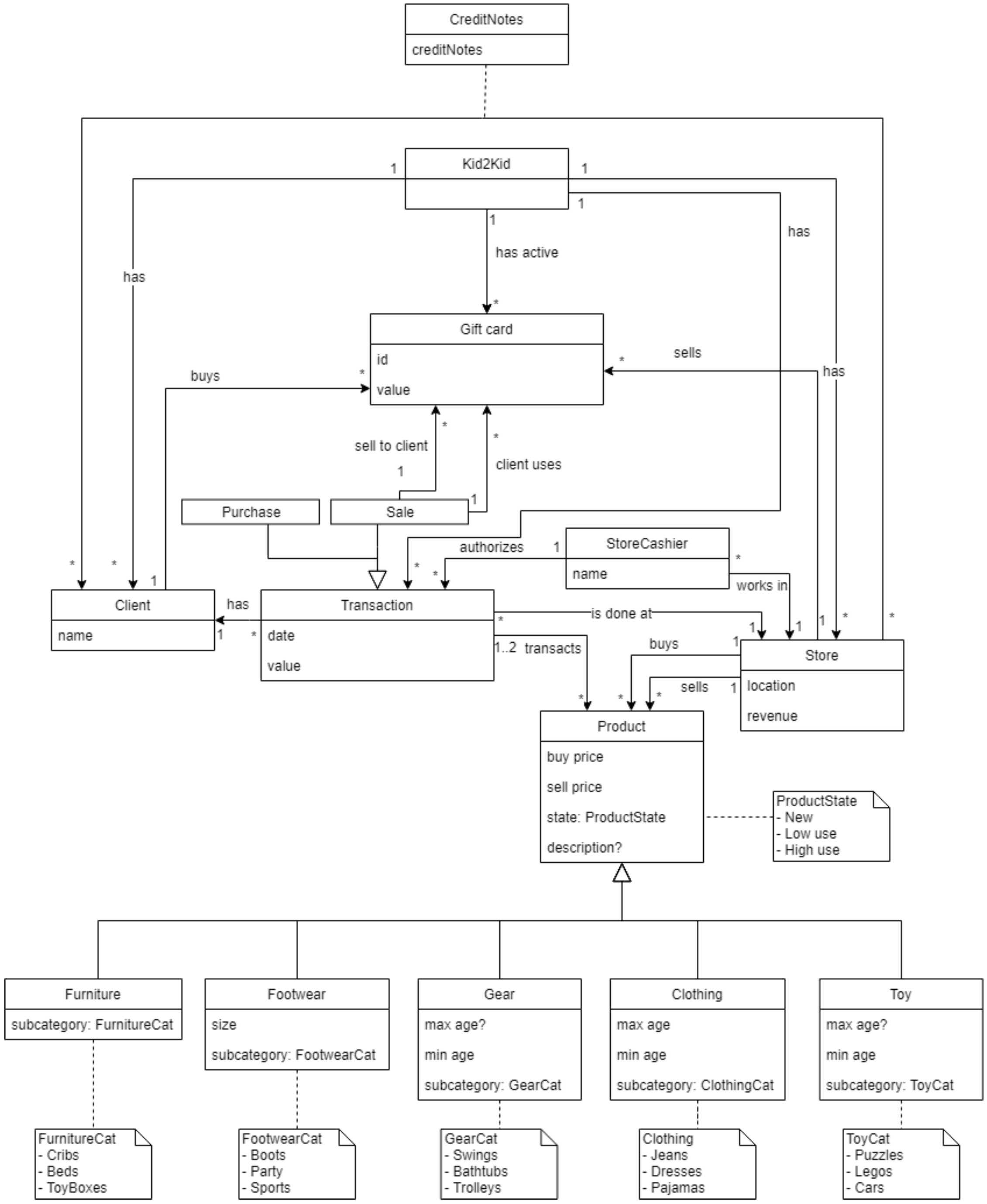
Scenario	**Sell products to clients in gift cards**
Description	Clients can pay using giftcards bought by anyone
Pre-conditions	• Logged in user has cashier permissions • Client exists in the system • Cashier that authorizes the transaction exists in the system • Product exists in the store • GiftCards to be used are active • The value of all giftcards is bigger than the value of the product being bought
Post-conditions	• Product is not available in the store • Product is saved as sold in the store • GiftCards used are set as inactive • Product is saved as bought by the client • A sale transaction is created

Scenario	**Sell giftcards to clients**
Description	A store can sell giftcards to clients
Pre-conditions	• Logged in user has cashier permissions • Client exists in the system • Cashier that authorizes the transaction exists in the system • Giftcard value is one of the possible values
Post-conditions	• A new giftcard is set as active • GiftCard is saved as sold in the store • GiftCard is saved as bought in the client • A sale transaction is created

Scenario	**A cashier can buy products to clients and pay in cash**
Description	Clients sell products to the store and are paid in cash
Pre-conditions	• Logged in user has cashier permissions • Client exists in the system • Cashier that authorizes the transaction exists in the system • Product is saved as sold by the client
Post-conditions	• Product bought exists in the store • Product is saved as bought in the store • A purchase transaction is created

Scenario	**A cashier can buy products to clients and pay in credit notes**
Description	Clients sell products to the store and receive credit notes at the store
Pre-conditions	• Logged in user has cashier permissions • Client exists in the system • Cashier that authorizes the transaction exists in the system • Product is saved as sold by the client
Post-conditions	• Product bought exists in the store • Product is saved as bought in the store • A purchase transaction is created • Client credit notes in the store increase by the value of the product sold * 1.2

2.2 Class Diagram



Class	Description
Client	Defines a client for the stores.
Clothing	One of the types of Products available to transact.
Date	Simple Date class with day, month and year.
Footwear	One of the types of Products available to transact.
Furniture	One of the types of Products available to transact.
Gear	One of the types of Products available to transact.
GiftCard	Gift card which can be sold to clients for one of them to use in sales later.
Kid2Kid	Core model; defines the state variables and operations available to the users.
Kid2KidTest	Defines the test/usage scenarios and test cases for the Kid2Kid system.
Product	Defines a product which can be transacted at the Kid2Kid stores.
Purchase	Purchase transaction of products from clients.
Sale	Sale transaction of products and gift cards to clients.
Store	Defines each of Kid2Kid's physical stores.
StoreCashier	Defines a cashier working at a physical store.
Toy	One of the types of Products available to transact.
Transaction	Abstract class for the purchase and sale of products and gift cards.

3 Formal VDM++ model

The model, along with its colored coverage, is annexed in the file *VDM+Coverage.pdf*.

4 Model validation (i.e., testing)

There is a class which tests the entire project, Kid2KidTest. As the user's interaction with the system passes through the class Kid2Kid, this is the most directly tested class. By fully testing this class, most features already get covered in the underlying classes.

The class Kid2KidTest is present at the end of the file *VDM+Coverage.pdf*.

5 Model verification (i.e., consistency analysis)

5.1 Example of domain verification

One of the proof obligations generated by Overture is:

No.	PO Name	Type
183	Store‘getCreditNotesOfClient(nat)	legal map application

The code under analysis (with the relevant map application underlined) is:

```
public getCreditNotesOfClient: nat ==> real
getCreditNotesOfClient(clientId) ==
    if clientId in set dom clientsCreditNotes then
        (return clientsCreditNotes(clientId))
    else (
            clientsCreditNotes := clientsCreditNotes ++ {clientId |-> 0};
            return 0
        )
    pre true
    post RESULT = clientsCreditNotes(clientId);
```

In this case the proof is trivial because the quantification 'clientId in set dom clientsCreditNotes' in the 'if' condition assures that the map is accessed only inside its domain.

5.2 Example of invariant verification

Another proof obligation generated by Overture is:

No.	**PO Name**	**Type**
155	Product'setPrices()	state invariant holds

The code under analysis is:

```
-- Algorithm that sets the buy price of the Product
protected setPrices: () ==> ()
setPrices() ==
        (
        buyPrice := 10 * getStateValue();
        sellPrice := 1.3 * buyPrice
        )
pre state <> undefined
post buyPrice > 0 and sellPrice >= buyPrice;
```

The relevant invariant under analysis is:

```
inv sellPrice >= buyPrice;
```

After the execution of the body block we have:

```
buyPrice = 10 * getStateValue() and sellPrice = 1.3 * 10 * getStateValue()
```

We have to prove that this implies that the invariant holds, i.e., that the following condition holds:

```
buyPrice = 10 * getStateValue() and sellPrice = 1.3 * 10 * getStateValue()
=> sellPrice >= buyPrice
```

which can be rewritten as:

```
1.3 * 10 * getStateValue() >= 10 * getStateValue()
```

which is obviously true.

6 Code generation

The project had its Java code generated through Overture. Then, a console-based menu was created to interact with it, being that the Kid2Kid system is accessible anywhere in the menu as a singleton. Be advised, though, that the pre and post-conditions defined in the VDM++ model are not carried on to Java - there shouldn't be a problem with this as the interface itself limits the interactions possible with the system.

In the code sent in this delivery, the Java project sits inside the folder generated/java. In the 'src' folder, the package 'vdm' contains the code generated by Overture and the remaining packages form the console interface.

7 Conclusions

This project has modelled in VDM++ all the listed requirements and business rules. In addition, a console interface was developed in Java to better demonstrate the system's capabilities.

In terms of future work, the following items could be worked on:

- Improvements to the product price calculation - currently, it only relies on the product's state, it doesn't take into account what it actually is.
- Make test cases for exceptional conditions - no such tests were built as we didn't figure out how to expect pre and post-condition failures from operations.
- Pre and post-condition verification in the generated Java code - however, this should be done automatically by Overture.
- A better, graphical interface.

Work was divided equally between team members.

References

[1] Peter Gorm Larsen et al, *VDM-10 Language Manual*, March 2014.

[2] Overture tool web site, `http://overturetool.org`