# Object Oriented Design Pattern Continuous Assignment

|  |  |
| --- | --- |
| Team members(PT03) | |
| Xiong Gaogao |  |
| Wang Jiqin |  |
| Zhang Shizhen |  |
| Chen Hao |  |
| Han Kai |  |
| Chong Wenchang |  |

## Problem 1

### Problem description

### Candidate patterns

### Motivation to choose abstract factory pattern

### Structure of the pattern

### E:\ISS\OODP\docs\AbstractFactory\AbstractFactory.jpg

### Collaborations

### E:\ISS\OODP\docs\AbstractFactory\AbstractFactorySeq.jpg

### Implementation decisions

* Because for every single UI Factory, typically we only need one instance, we use singleton design pattern per UI factory.
* For each UI factory, we can apply factory method pattern or prototype pattern. Since in this case, the UI components like buttons, labels, layouts, combox and images etc are different from each other, instead of the prototype design pattern, we will use factory method pattern to create each special UI component.
* Adding a parameter to the UI component creation interfaces so that we can more easily to extend any other possible UI components.

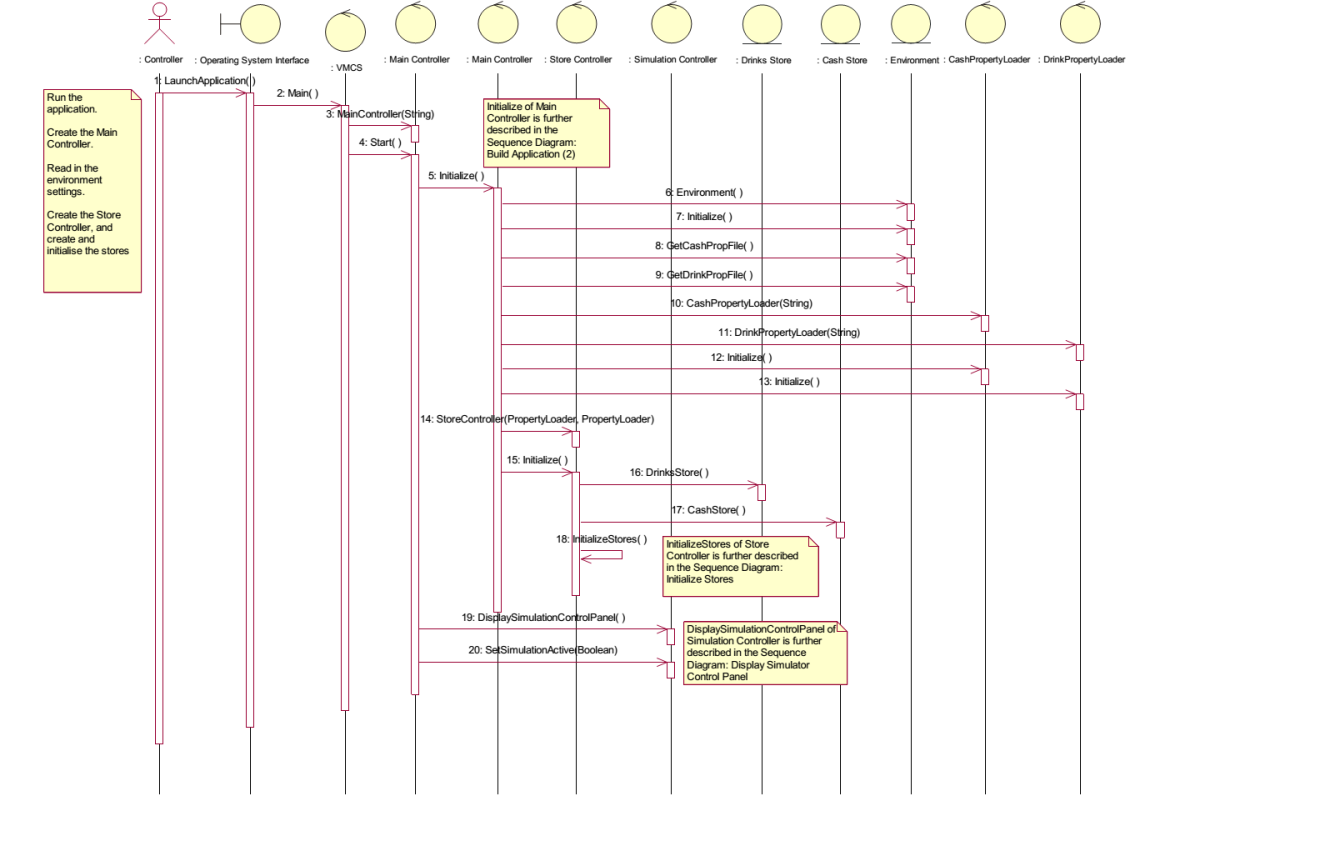
## Problem 2

### Problem description

Class diagram before bridge pattern

### E:\ISS\OODP\docs\Bridge\bridge_cls_before.png

Collaboration diagram before bridge pattern



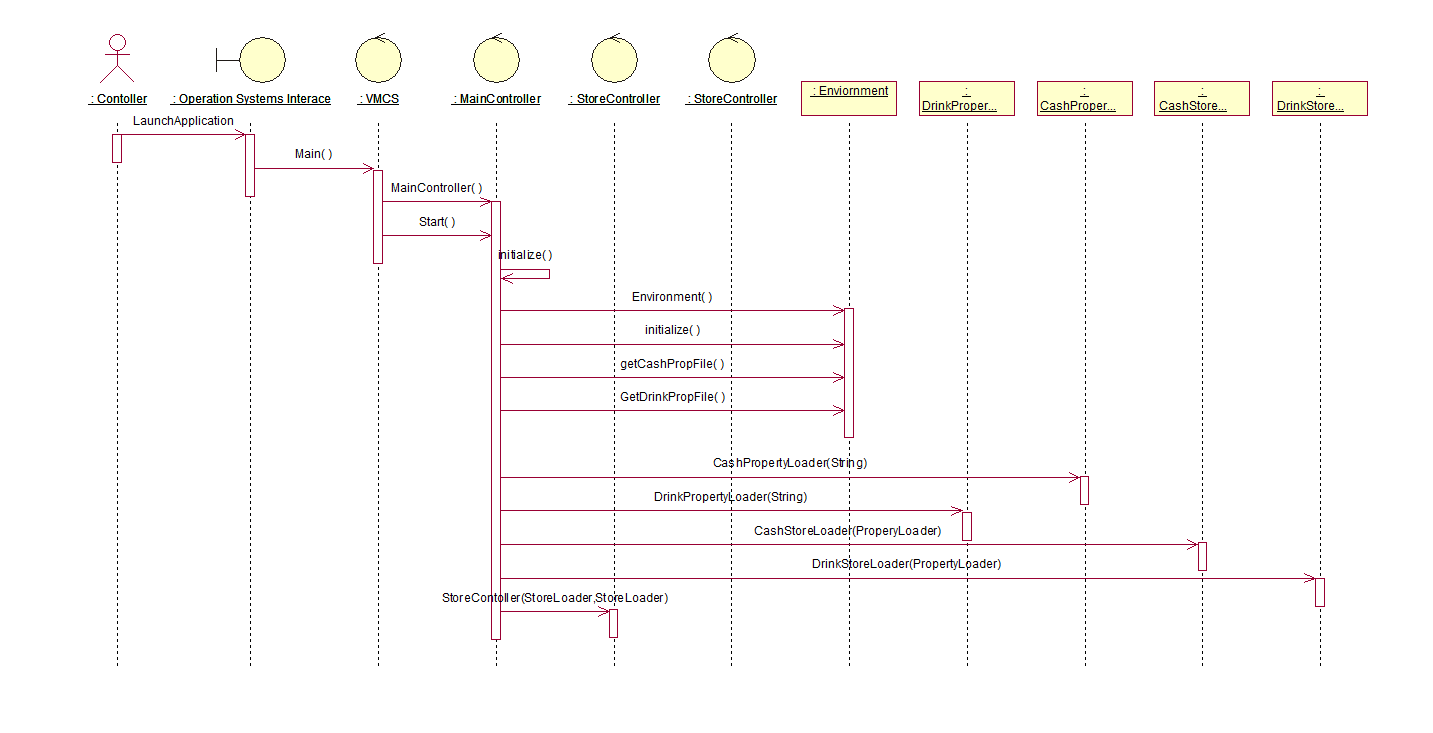
### Candidate patterns

### Motivation to choose bridge pattern

### Structure of the pattern

### E:\ISS\OODP\docs\Bridge\bridge_cls_after.png

### Collaborations



### Implementation decisions

* Although in this vmcs application, it is highly possible that we only need to implement one property file system, say txt file for example. But it will still be better if we use bridge design pattern to do a separation for the exact implementation of property file system and the abstracted interfaces, because when some specific changed need to be made on the property file readings, we only need to re-link the whole system rather than re-compile the whole system.
* When this vmcs system is designed to be deployed on different platforms, on the implementation side we use a parameter in the constructor to decide which platform interfaces should be used, in other words, we will choose the correct implementer dynamically. In this case, the abstract factory pattern also can be applied. Adding a parameter to the UI component creation interfaces so that we can more easily to extend any other possible UI components.

## Problem 3

### Problem description

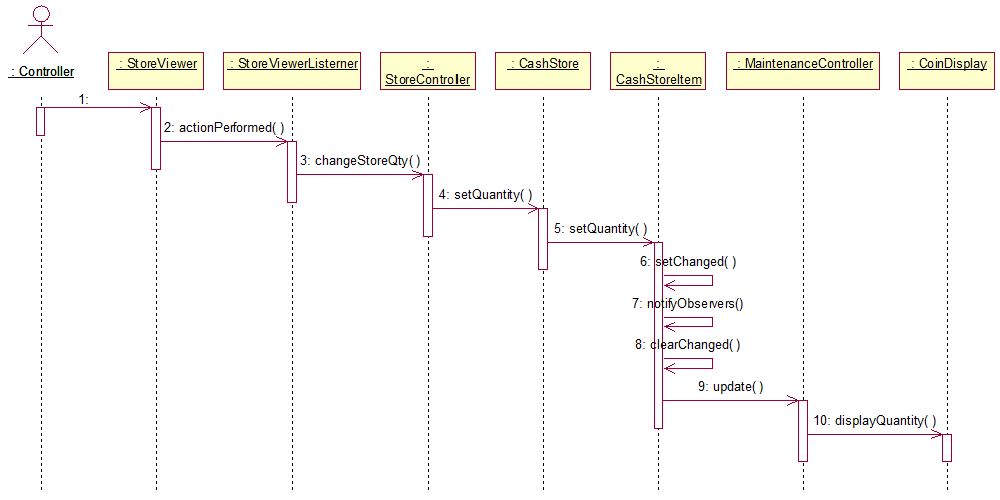
### Candidate patterns

### Motivation to choose observer pattern

### Structure of the pattern

### E:\ISS\OODP\docs\Observer\Observer.jpg

### Collaborations



### Implementation decisions

* Direct mapping subjects to their observers. Since in this application, there are only a few subjects need to be observed such as CashStoreItem and DrinkStoreItem, storage is not a problem.
* 2. Within this system, the TransactionController, MaintenanceController and MachineryController are the observers of CashStoreItem and DrinkStoreItem, in order that each controller can know which subject is sending the update notification, we put a parameter of the subject itself to indicate the observers.
* Subject will be responsible for sending the notification. Because all of the backend controller need to immediately know the updated information of CashStoreItem and DrinkStoreItem.

## Problem 4

### Problem description

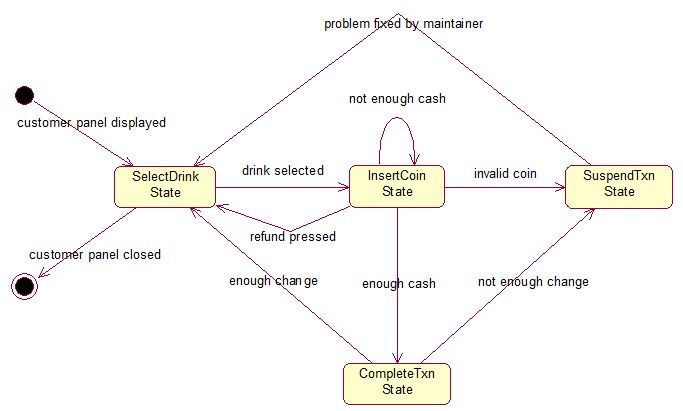
### Candidate patterns

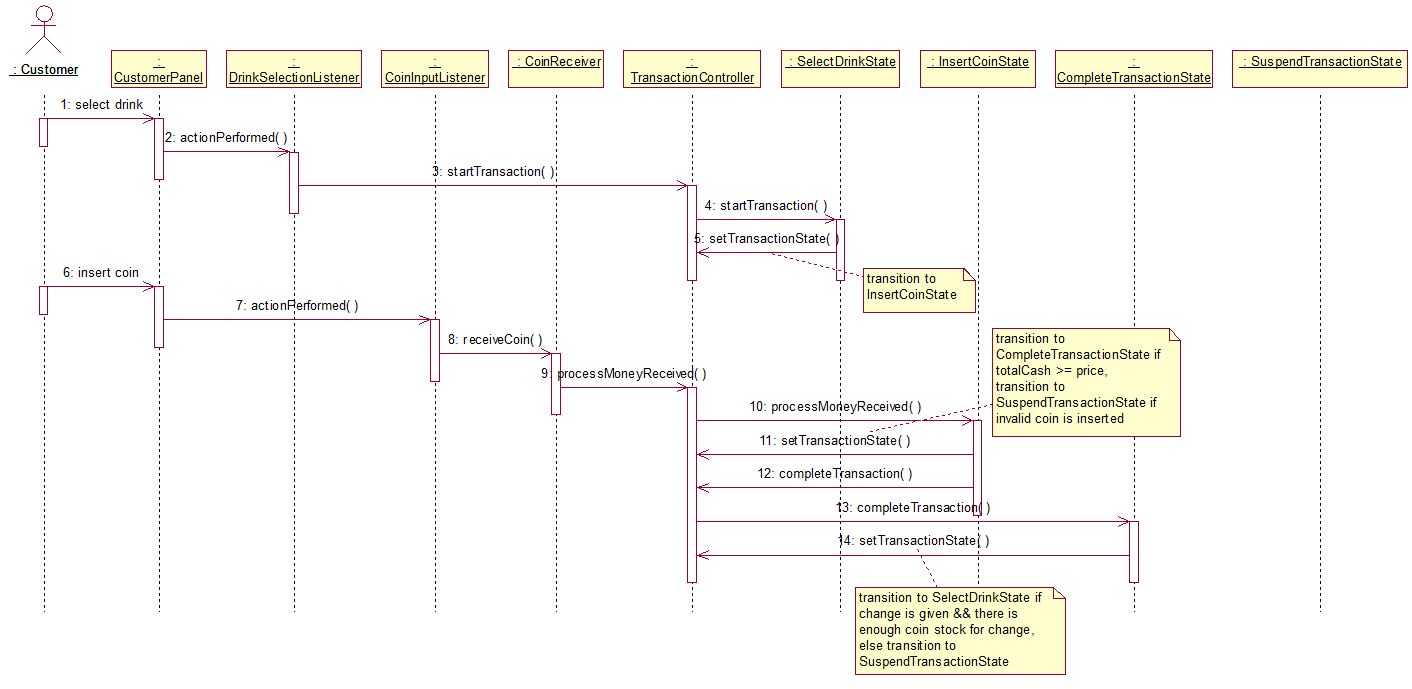
### Motivation to choose state pattern

### Structure of the pattern

### E:\ISS\OODP\docs\State\State.jpg

### Collaborations





### Implementation decisions

* Since the state transition criteria is pre-fixed, we control the state transition in a TransactionController context.
* When user initialize a new transaction, all the states will be created simultaneously. This approach will speed up the system performance which can bring a responsive experience to users.

## Problem 5

### Problem description

### Candidate patterns

### Motivation to choose

### Structure of the pattern

### Collaborations

### Implementation decisions

## Problem 6

### Problem description

### Candidate patterns

### Motivation to choose

### Structure of the pattern

### Collaborations

### Implementation decisions