Rodrigo Aranguren Carmona
A20432501

Project report

1. MDA-EFSM MODEL FOR THE VENDING MACHINE COMPONENTS

a. Meta events

1. create()
2. insert_cups(int n) // n represents # of cups
3. coin(int f) // f=true: sufficient funds inserted for a drink f=false: not sufficient funds for a drink
4. card()
5. cancel()
6. set_price()
7. dispose_drink(String d) // d represents a drink id
8. additive(String a) // a represents additive id

b. Meta actions

1. StorePrice()
2. ZeroCF() // zero Cumulative Fund cf
3. IncreaseCF() // increase Cumulative Fund cf
4. ReturnCoins() // return coins inserted for a drink
5. DisposeDrink(String d) // dispose a drink with d id
6. DisposeAdditive(HashMap<String,Boolean> A) //dispose marked additives in A map where additive with i id is disposed when A[i]=true

c. Pseudo-code of all operations of VM1 and VM2

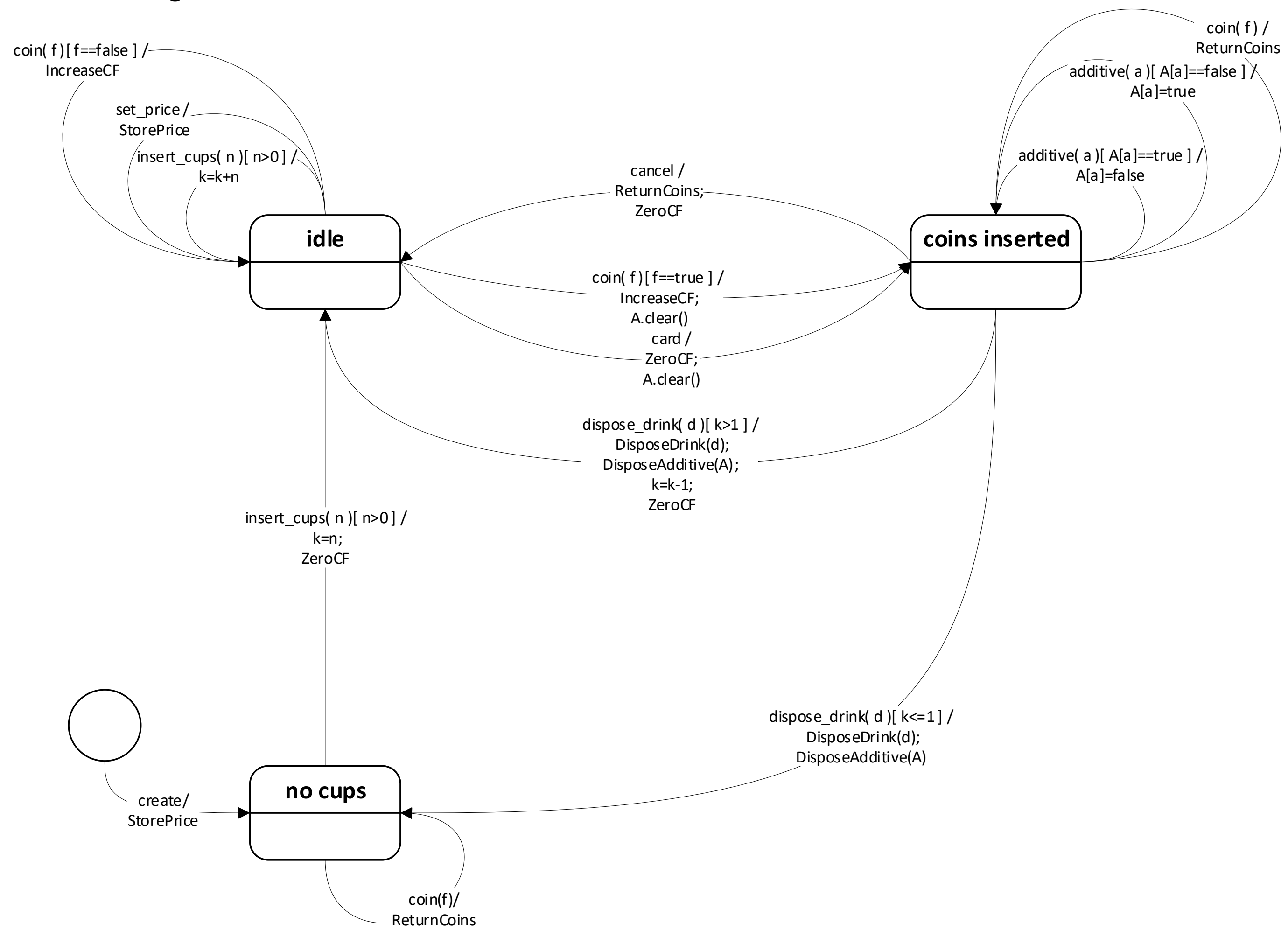
VM1

```
create(int p) {
 d->temp_p=p;
 m->create();
}
coin(int v) {
 d->temp_v=v;
 if (d->cf+v>=d->price) m->coin(1);
else m->coin(0);
}
card(float x) {
 if (x>=d->price) m->card();
}

sugar() { m->additive(1); }

tea() { m->dispose_drink(1); }

chocolate() { m->dispose_drink(2); }

insert_cups(int n) {
 m->insert_cups(n);
}
set_price(int p) {
 d->temp_p=p;
 m->set_price()
}
cancel() {
 m->cancel();
}
```

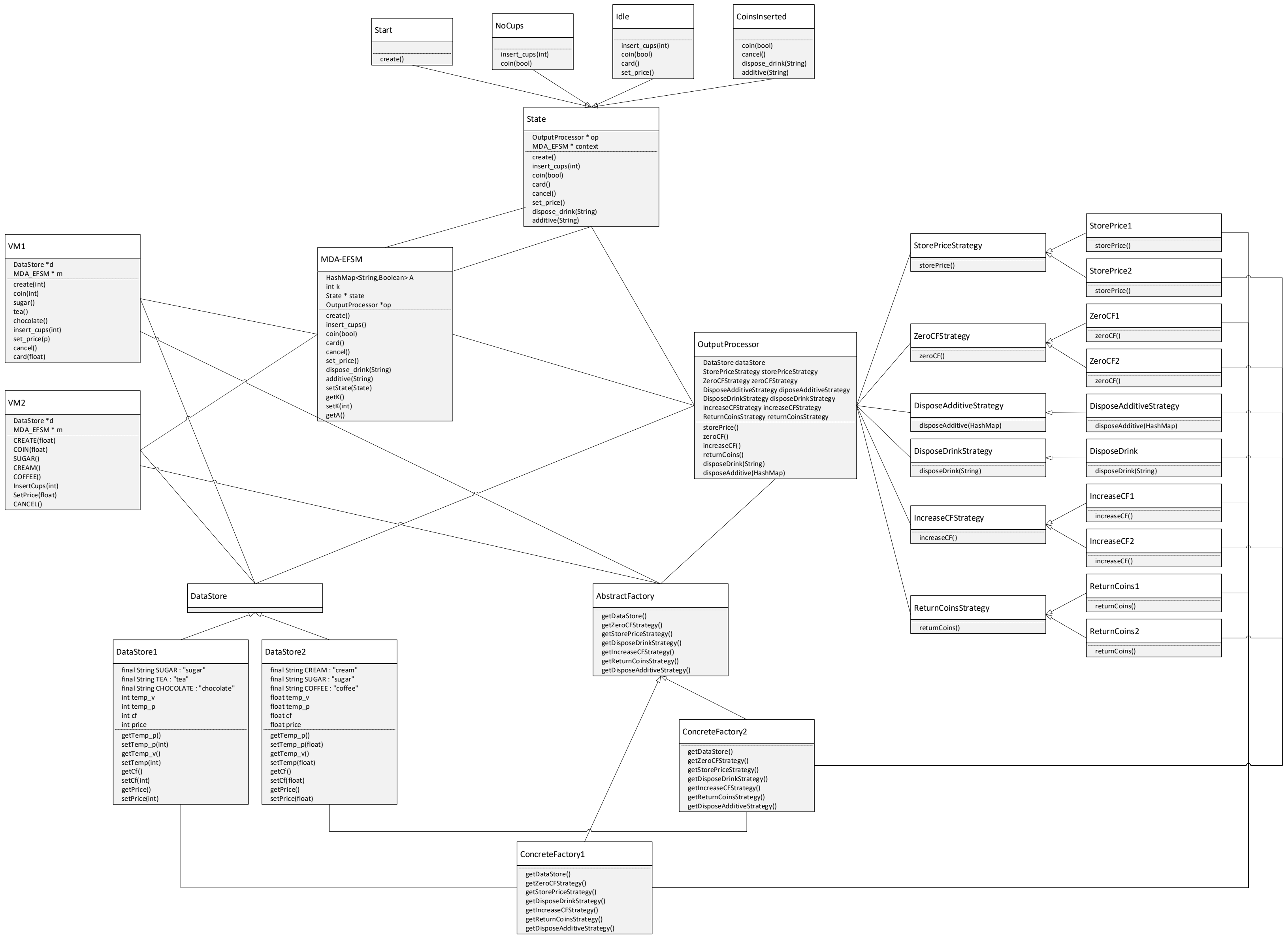
VM2

```
CREATE(float p) {
 d->temp_p=p;
 m->create();
}
COIN(float v) {
 d->temp_v=v;
 if (d->cf+v>=d->price) m->coin(1);
else m->coin(0);
}
SUGAR() {
 m->additive(2);
}
CREAM() {
 m->additive(1);
}
COFFEE() {
 m->dispose_drink(1);
}
InsertCups(int n) {
 m->insert_cups(n);
}
SetPrice(float p) {
 d->temp_p=p;
 m->set_price()
}
CANCEL() {
 m->cancel();
}
```

d. State diagram for the MDA-EFSM

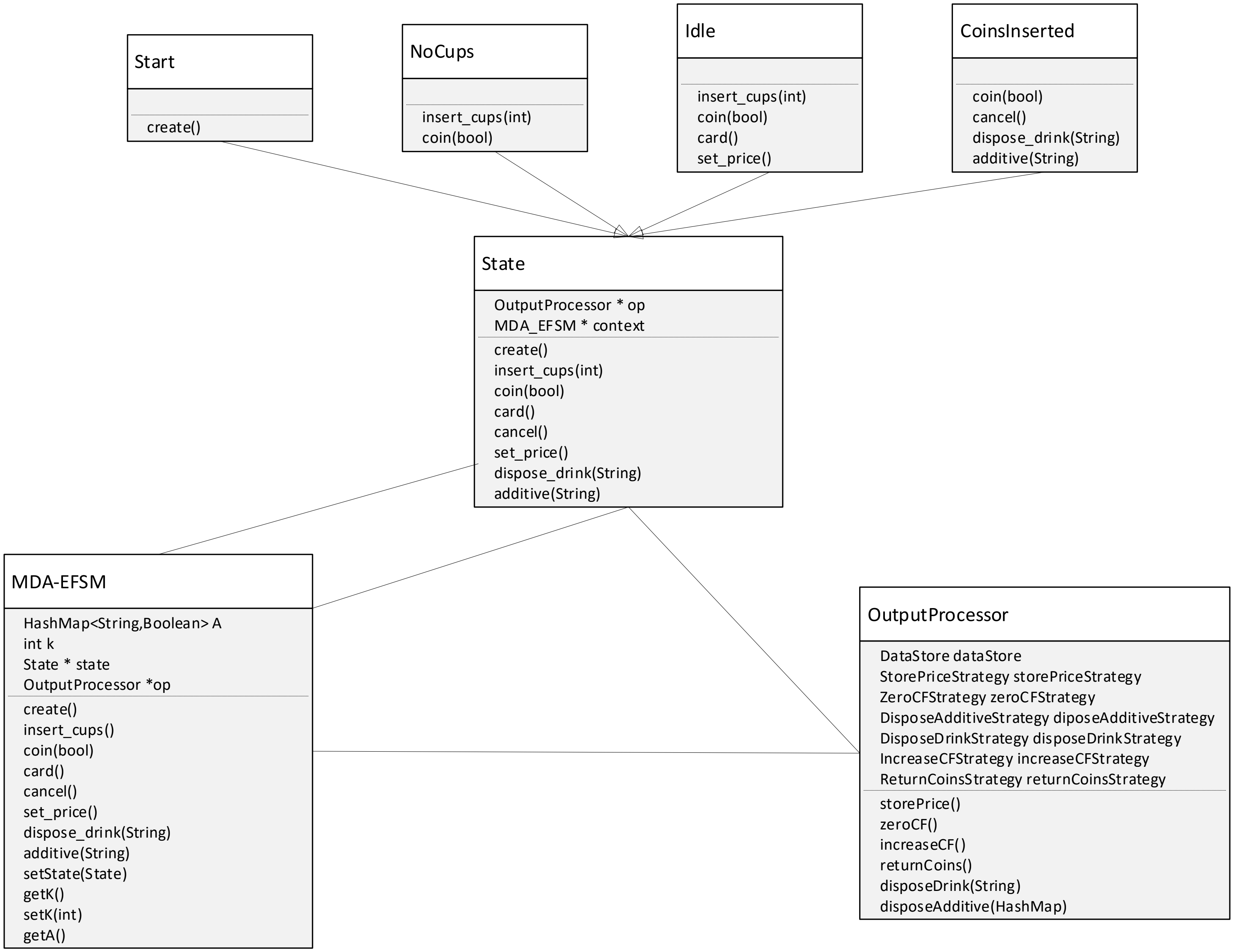


2. CLASS DIAGRAM (GENERAL VIEW)

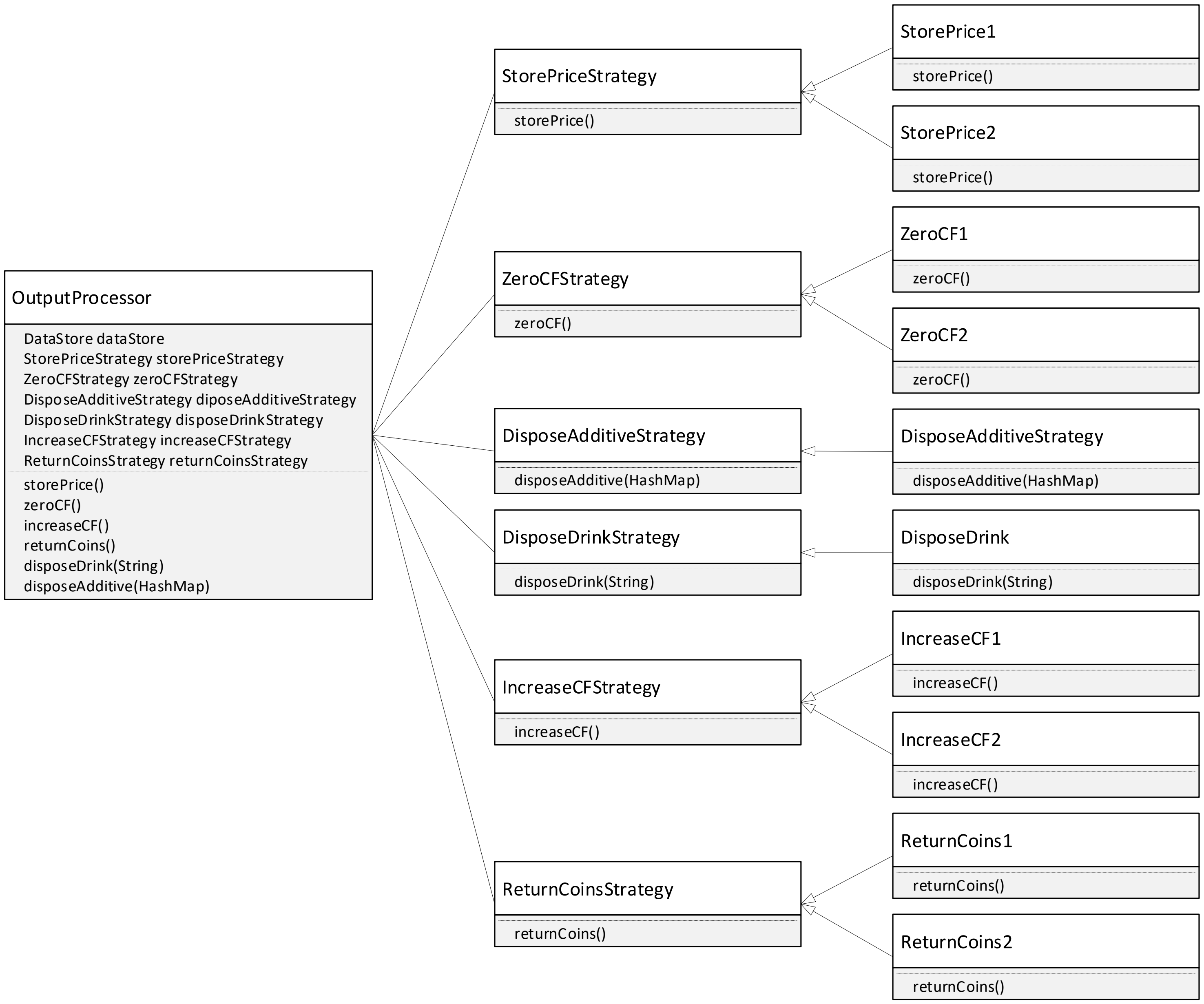


Rodrigo Aranguren Carmona
A20432501

a. State pattern (detail)

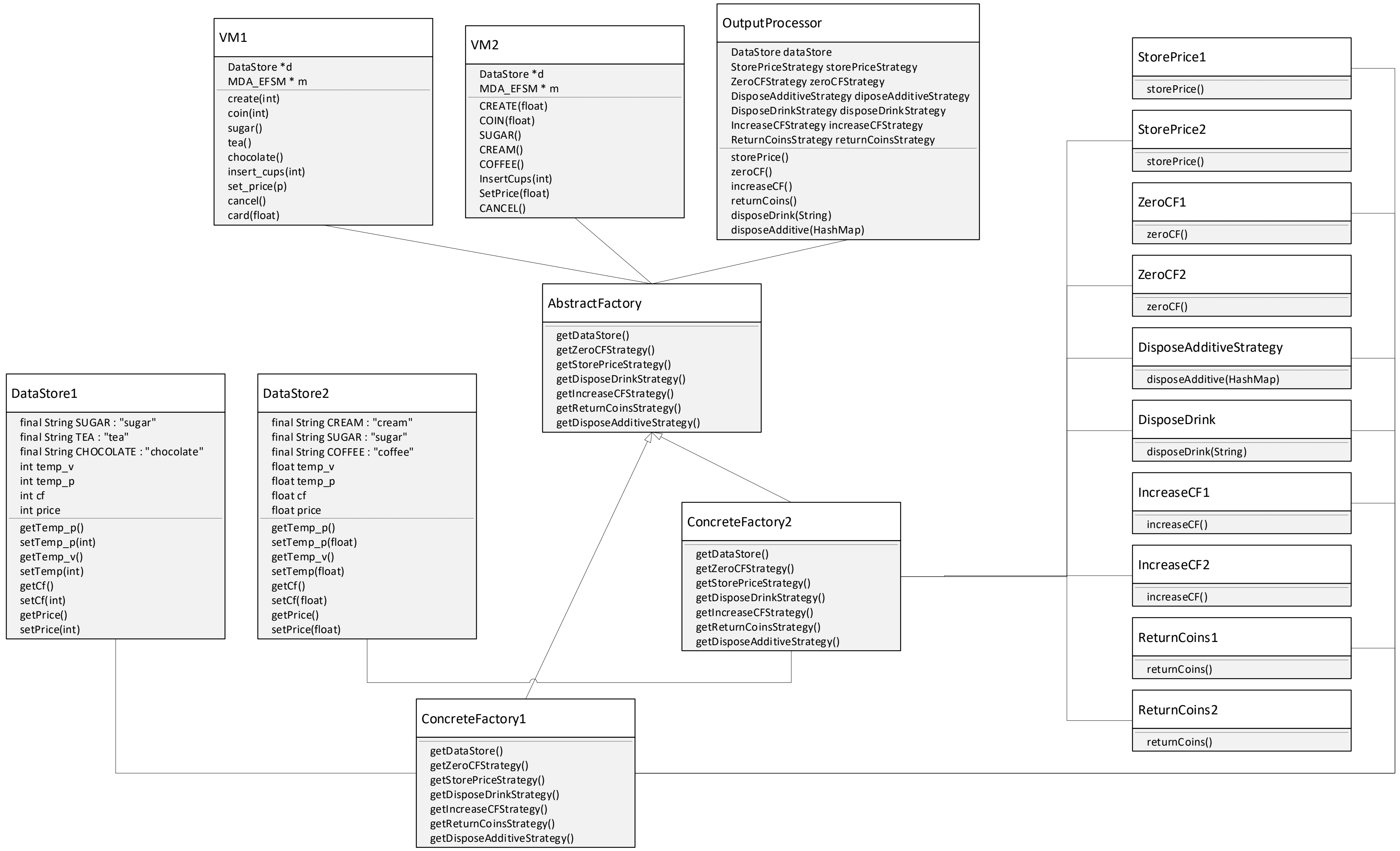


b. Strategy Pattern (detail)



Rodrigo Aranguren Carmona
A20432501

c. Abstract Factory Pattern (detail)



3. CLASS DESCRIPTIONS

Class	Category	Description	Methods
VM1	Input Processor	IP for Vending Machine system 1	create(int): init the vending machine system with a given drink price coin(int): insert a coin of a given value in the system sugar(): request sugar or cancel sugar tea(): request tea drink chocolate(): request chocolate drink insert_cups(int): insert cups into the system set_price(int): set the drink price cancel(): cancel the drink request card(float): swipe card with a given credit score
VM2	Input Processor	IP for Vending Machine system 2	CREATE(float): init the system with a given drink price COIN(float): inserted coins for a given value SUGAR(): request sugar or cancel sugar CREAM(): request cream or cancel cream COFFEE(): request coffee drink InsertCups(int): insert cups into the system SetPrice(float): set the drink price CANCEL(): cancel the drink request
MDA_EFSM	MDA-EFSM, State pattern	Model independent logic	create(): event to create system insert_cups(): event where cups are inserted into system coin(bool): event when coins are inserted (enough or not for a drink) card(): event where card was swiped in the system cancel(): event when a cancel is requested set_price(): event when the price is to e set to a value dispose_drink(String): event when a specific drink is requested additive(String): event when an additive is requested setState(State): setter for the next state
Output Processor	Output processor, Strategy pattern,	Output processor, where the model triggers the actions	storePrice(): action that stores definitive drink price from temporal field zeroCF(): action that zeroes accumulated funds

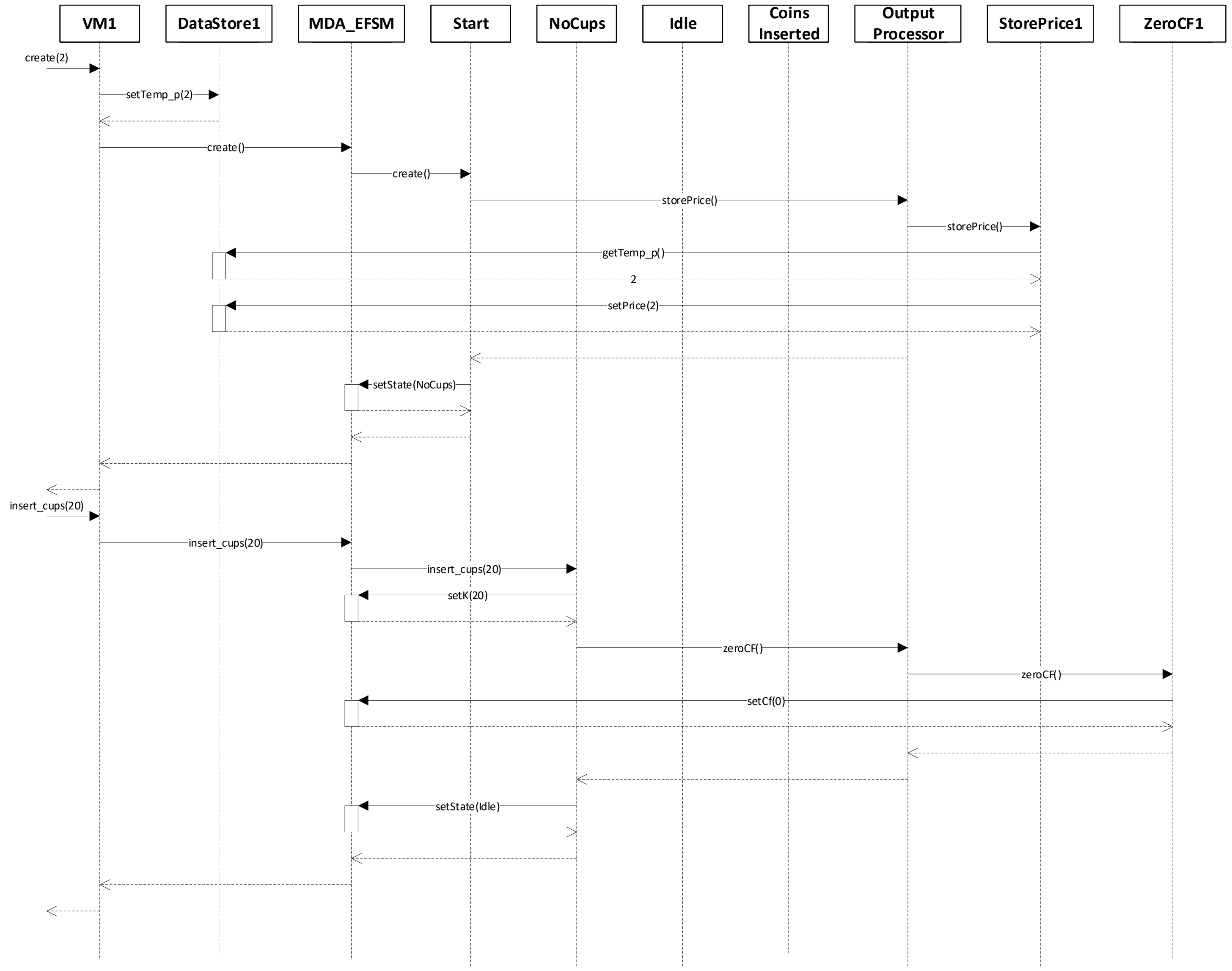
	Abstract Factory pattern		increaseCF(): action that increases the accumulated funds returnCoins(): action that returns coins disposeDrink(String): action that disposes a drink disposeAdditive(HashMap): action that disposes all configured additives
StorePrice Strategy	Strategy Pattern	Interface	storePrice(): abstract
StorePrice 1	Strategy Patter	Concrete strategy for VM1	storePrice(): action that stores definitive drink price from temporal field
StorePrice 2	Strategy Patter	Concrete strategy for VM2	storePrice(): action that stores definitive drink price from temporal field
ZeroCF Strategy	Strategy Pattern	Interface	zeroCF(): abstract
ZeroCF1	Strategy Pattern	Concrete strategy for VM1	zeroCF(): action that zeroes accumulated funds
ZeroCF2	Strategy Pattern	Concrete strategy for VM2	zeroCF(): action that zeroes accumulated funds
IncreaseCF Strategy	Strategy Pattern	Interface	increaseCF(): abstract
IncreaseCF 1	Strategy Pattern	Concrete strategy for VM1	increaseCF(): action that increases the accumulated funds
IncreaseCF 2	Strategy Pattern	Concrete strategy for VM2	increaseCF(): action that increases the accumulated funds
Return Coins Strategy	Strategy Pattern	Interface	returnCoins(): abstract
ReturnCoins1	Strategy Pattern	Concrete strategy for VM1	returnCoins(): action that returns coins
ReturnCoins2	Strategy Pattern	Concrete strategy for VM2	returnCoins(): action that returns coins
Dispose Drink Strategy	Strategy Pattern	Interface	disposeDrink(String): abstract
Dispose Drink	Strategy Pattern	Concrete strategy for VM1 and VM2	disposeDrink(String): action that disposes a drink
Dispose Additive Strategy	Strategy Pattern	Interface	disposeAdditive(HashMap): abstract
Disepose Additive	Strategy Pattern	Concrete strategy for VM1 and VM2	disposeAdditive(HashMap): action that disposes all configured additives
DataStore	Data Storage	Abstract class	
DataStore1	Data Storage	Model dependent data storage for VM1	Setters and getters for all fields
DataStore1	Data Storage	Model dependent data storage for VM1	Setters and getters for all fields

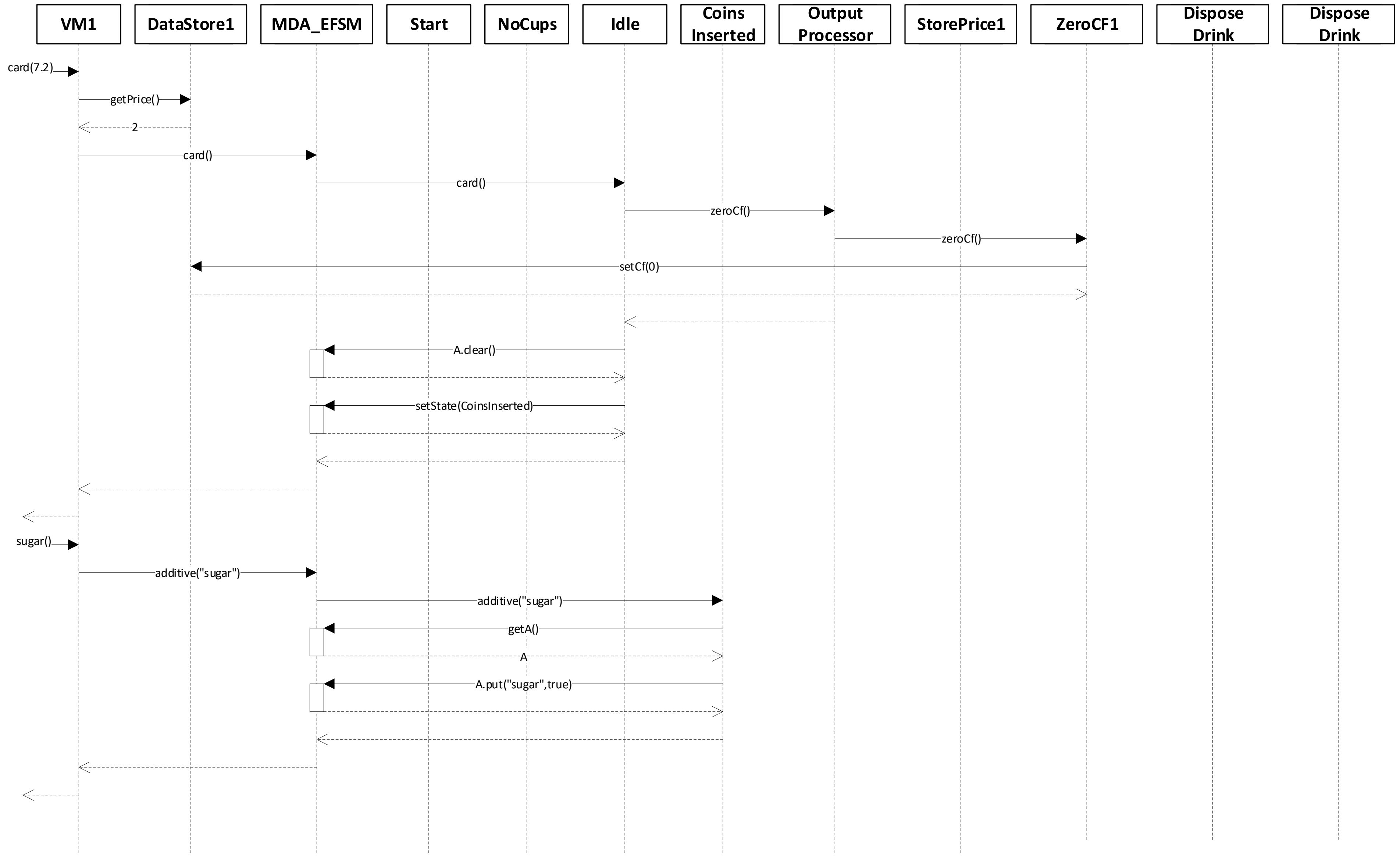
Abstract Factory	Abstract factory pattern	Abstract class	getDataStore(): abstract getZeroCFStrategy(): abstract getStorePriceStrategy(): abstract getDisposeDrinkStrategy(): abstract getIncreaseCFStrategy(): abstract getReturnCoinsStrategy(): abstract getDisposeAdditiveStrategy(): abstract
Concrete Factory1	Abstract factory pattern	Factory that initializes the appropriate objects for VM1	getDataStore(): get DataStore for VM1 getZeroCFStrategy(): get ZeroCF method for VM1 getStorePriceStrategy(): get StorePrice mehod for VM1 getDisposeDrinkStrategy(): get DisposeDrink method for VM1 getIncreaseCFStrategy(): get IncreaseCF method for VM1 getReturnCoinsStrategy(): get ReturnCoins mehod for VM1 getDisposeAdditiveStrategy(): get DisposeAdditive method for VM1
Concrete Factory2	Abstract factory pattern	Factory that initializes the appropriate objects for VM2	getDataStore(): get DataStore for VM2 getZeroCFStrategy(): get ZeroCF method for VM2 getStorePriceStrategy(): get StorePrice mehod for VM2 getDisposeDrinkStrategy(): get DisposeDrink method for VM2 getIncreaseCFStrategy(): get IncreaseCF method for VM2 getReturnCoinsStrategy(): get ReturnCoins mehod for VM2 getDisposeAdditiveStrategy(): get DisposeAdditive method for VM2
State	State pattern	Abstract class	create(): abstract insert_cups(int): abstract coin(bool): abstract card(): abstract cancel(): abstract set_price(): abstract dispose_drink(String): abstract additive(String): abstract
Start	State pattern	Concrete state. The initial state.	create(): initialize, switch state insert_cups(int): ignore coin(bool): ignore card(): ignore cancel(): ignore set_price(): ignore dispose_drink(String): ignore

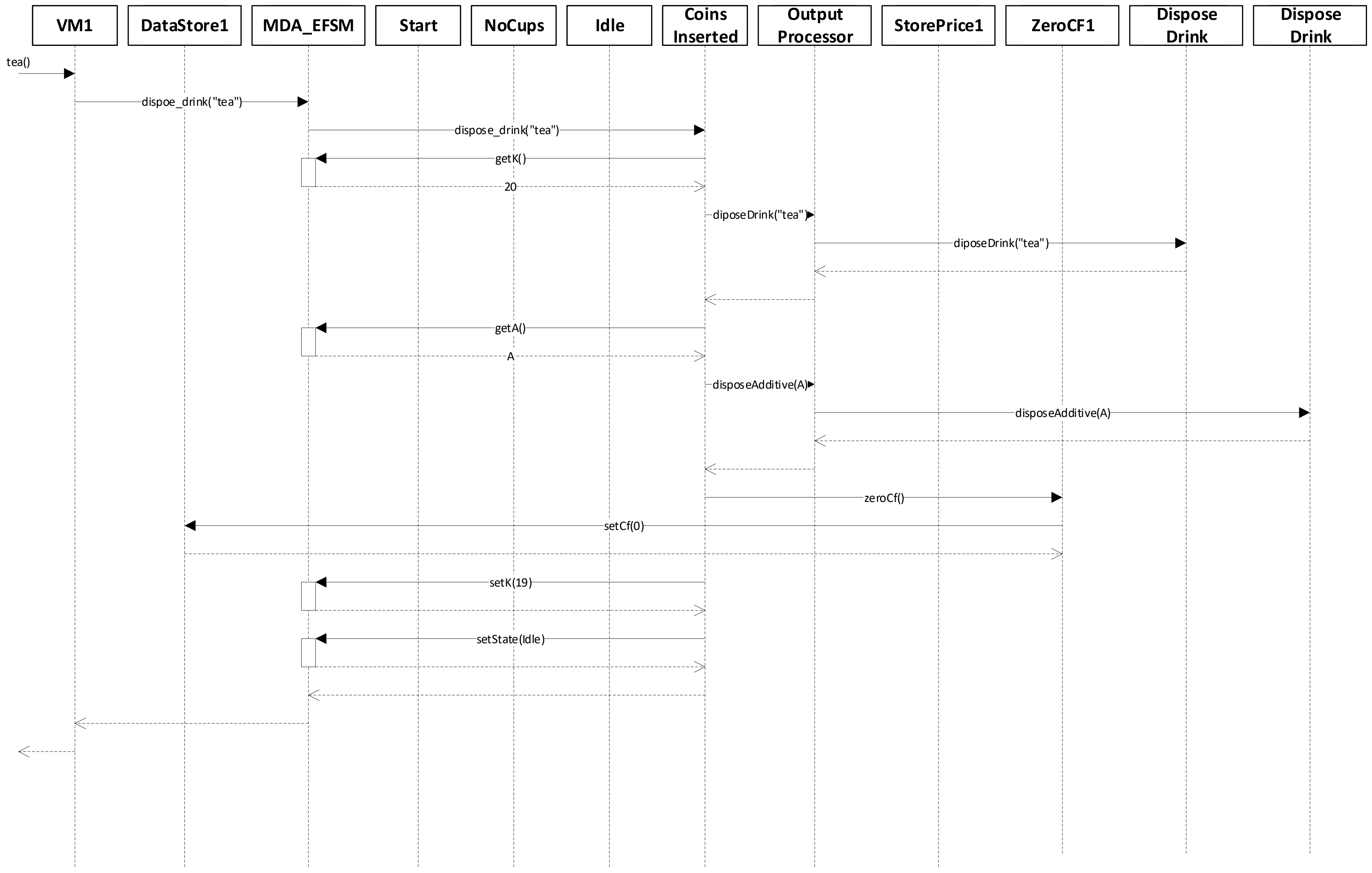
			additive(String): ignore
NoCups	State pattern	Concrete state. Symbolizes when there are no cups in system.	create(): ignore insert_cups(int): add cups to system, switch state coin(bool): return coins card(): ignore cancel(): ignore set_price(): ignore dispose_drink(String) : ignore additive(String): ignore
Idle	State pattern	Concrete state. Symbolizes when there are cups in system. Awaits until there are enough funds for a drink.	create(): ignore insert_cups(int): add cups to system coin(bool): increase funds. Switch state if needed card(): switch state cancel(): ignore set_price(): update drink price dispose_drink(String) : ignore additive(String) : ignore
Coins Inserted	State pattern	Concrete state. Symbolizes when a payment is done. Awaits for additives and	create(): ignore insert_cups(int): ignore card(): ignore cancel(): return coins and switch state set_price(): ignore dispose_drink(String) : dispose selected drink, switch state additive(String) : add additive or cancel additive

4. SEQUENCE DIAGRAMS

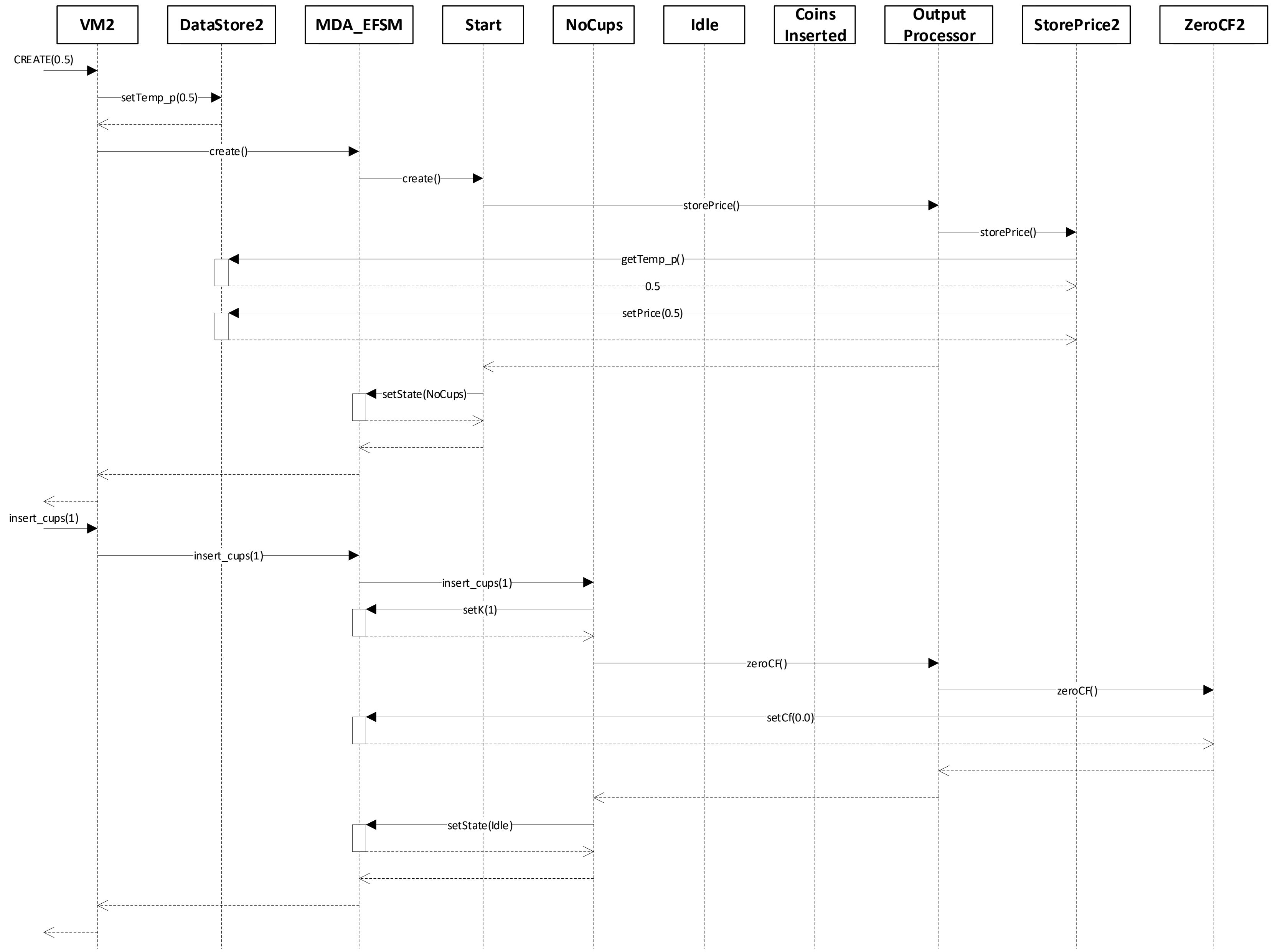
a. Scenario I

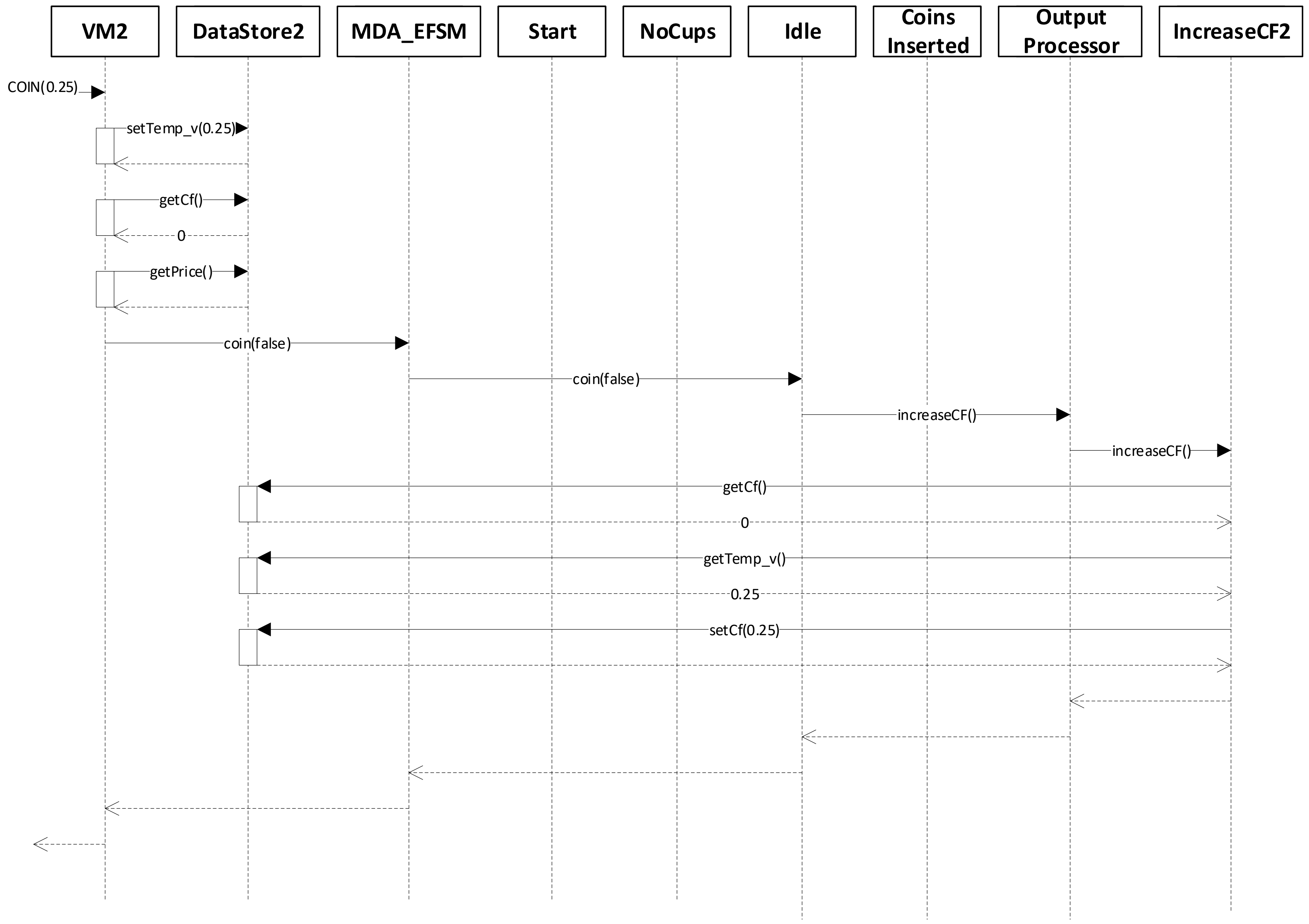


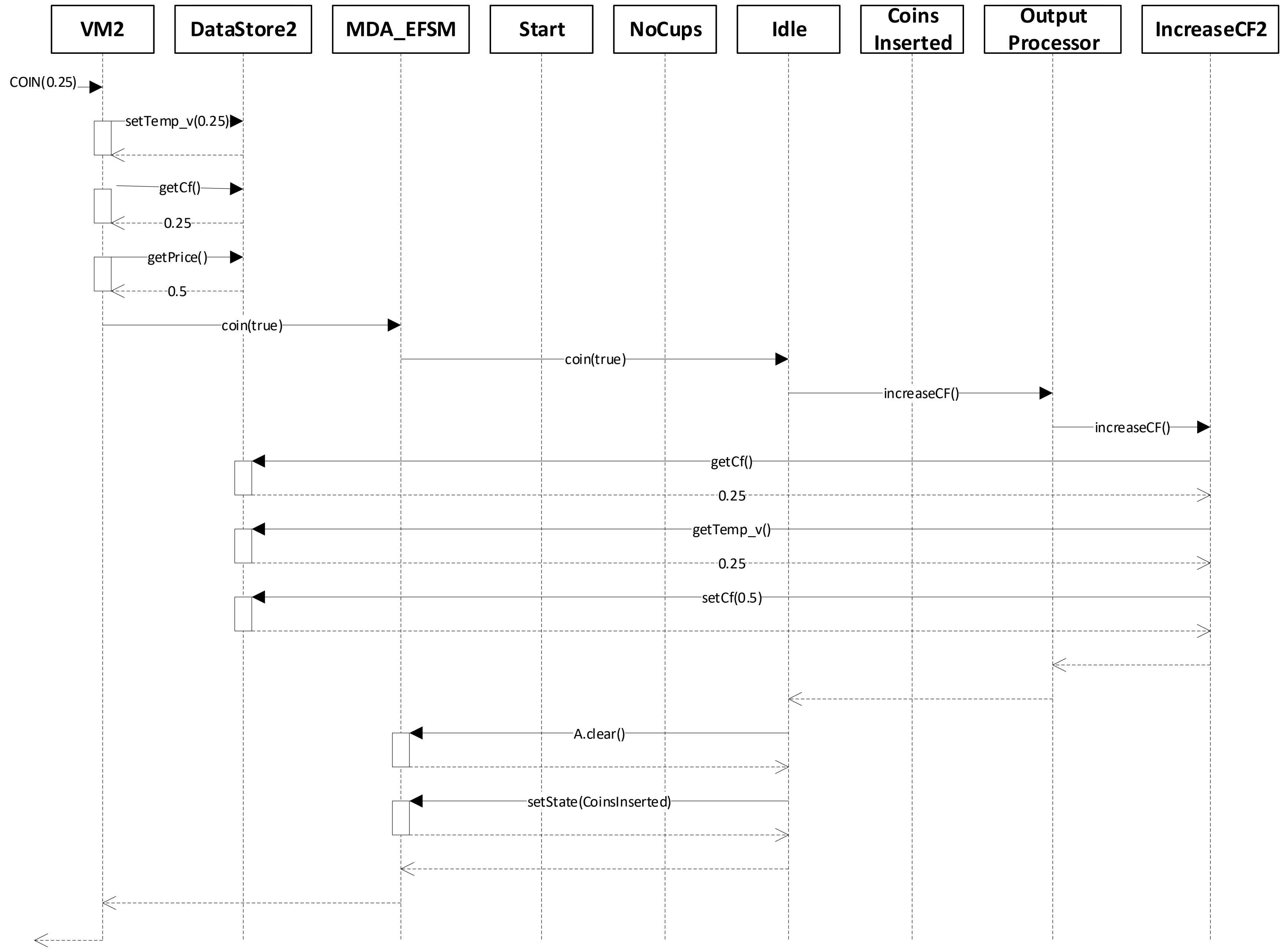

VM1
DataStore1
MDA_EFSM
Start
NoCups
Idle
Coins Inserted
Output Processor
StorePrice1
ZeroCF1
Dispose Drink
Dispose Drink
card(7.2)
getPrice()
2
card()
card()
zeroCf()
zeroCf()
setCf(0)
A.clear()
setState(CoinsInserted)
sugar()
additive("sugar")
additive("sugar")
getA()
A
A.put("sugar",true)

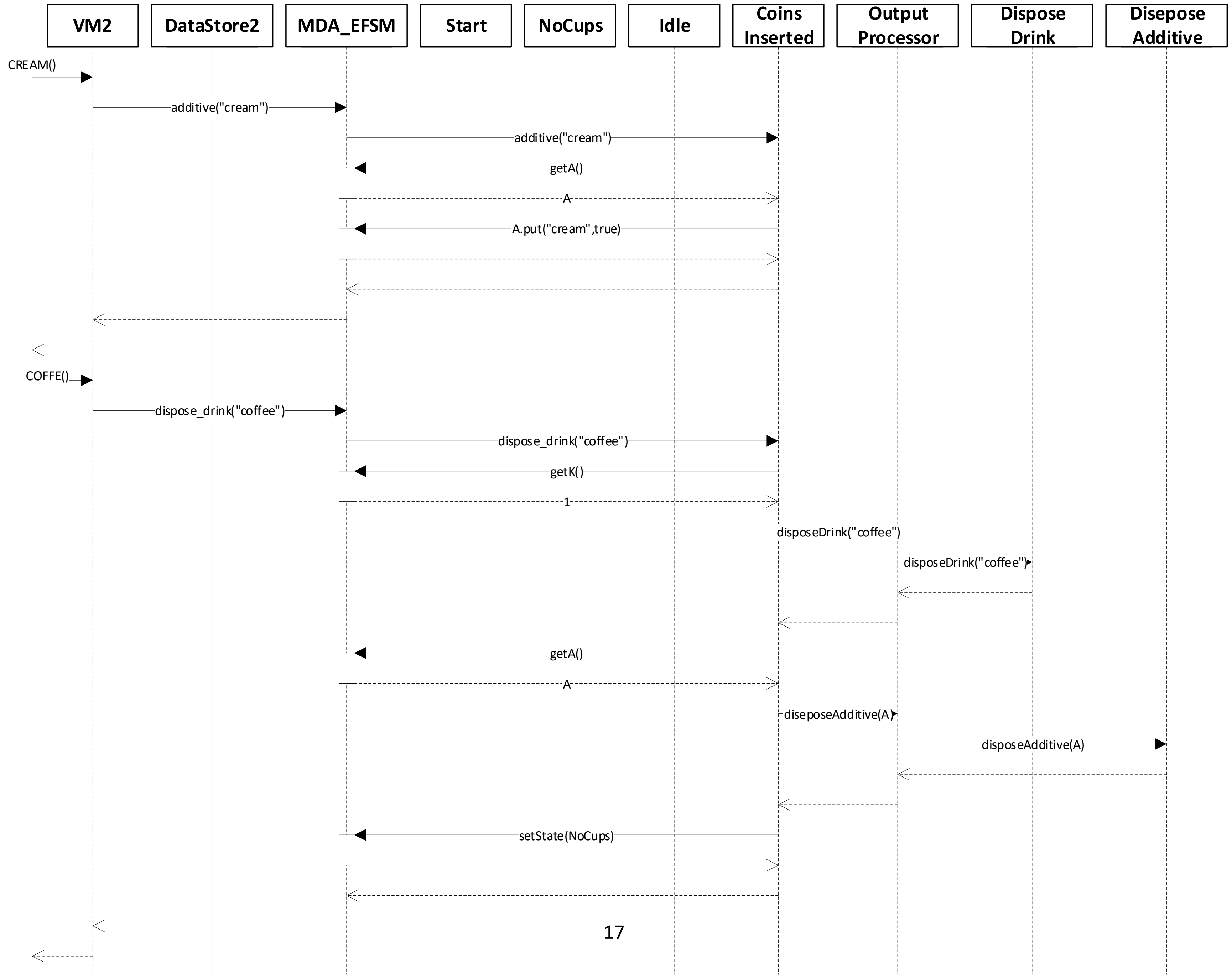

VM1
DataStore1
MDA_EFSM
Start
NoCups
Idle
Coins Inserted
Output Processor
StorePrice1
ZeroCF1
Dispose Drink
Dispose Drink
tea()
dispoe_drink("tea")
dispose_drink("tea")
getK()
20
diposeDrink("tea")
diposeDrink("tea")
getA()
A
disposeAdditive(A)
disposeAdditive(A)
zeroCf()
setCf(0)
setK(19)
setState(Idle)

b. Scenario II




VM2
DataStore2
MDA_EFSM
Start
NoCups
Idle
Coins Inserted
Output Processor
IncreaseCF2
COIN(0.25)
setTemp_v(0.25)
getCf()
0
getPrice()
coin(false)
coin(false)
increaseCF()
increaseCF()
getCf()
0
getTemp_v()
0.25
setCf(0.25)


VM2
DataStore2
MDA_EFSM
Start
NoCups
Idle
Coins Inserted
Output Processor
IncreaseCF2
COIN(0.25)
setTemp_v(0.25)
getCf()
0.25
getPrice()
0.5
coin(true)
coin(true)
increaseCF()
increaseCF()
getCf()
0.25
getTemp_v()
0.25
setCf(0.5)
A.clear()
setState(CoinsInserted)


VM2
DataStore2
MDA_EFSM
Start
NoCups
Idle
Coins Inserted
Output Processor
Dispose Drink
Disepose Additive
CREAM()
additive("cream")
additive("cream")
getA()
A
A.put("cream",true)
COFFE()
dispose_drink("coffee")
dispose_drink("coffee")
getK()
1
disposeDrink("coffee")
disposeDrink("coffee")
getA()
A
diseposeAdditive(A)
disposeAdditive(A)
setState(NoCups)

5. SOURCE CODE

Source code is attached in the deliverable.

State pattern is implemented in the State class and subclasses (Start, NoCups, Idle, CoinsInserted). The context class is the MDA_EFSM class.

Strategy pattern is implemented in the output processor and all the pointer to the respective strategies.

Abstract Factory Pattern is implemented in the AbstractFactory and ConcreteFactory1 and ConcreteFactory2

In order to execute the provided jar file, simply run:

```
java -jar vmsystem.jar
```