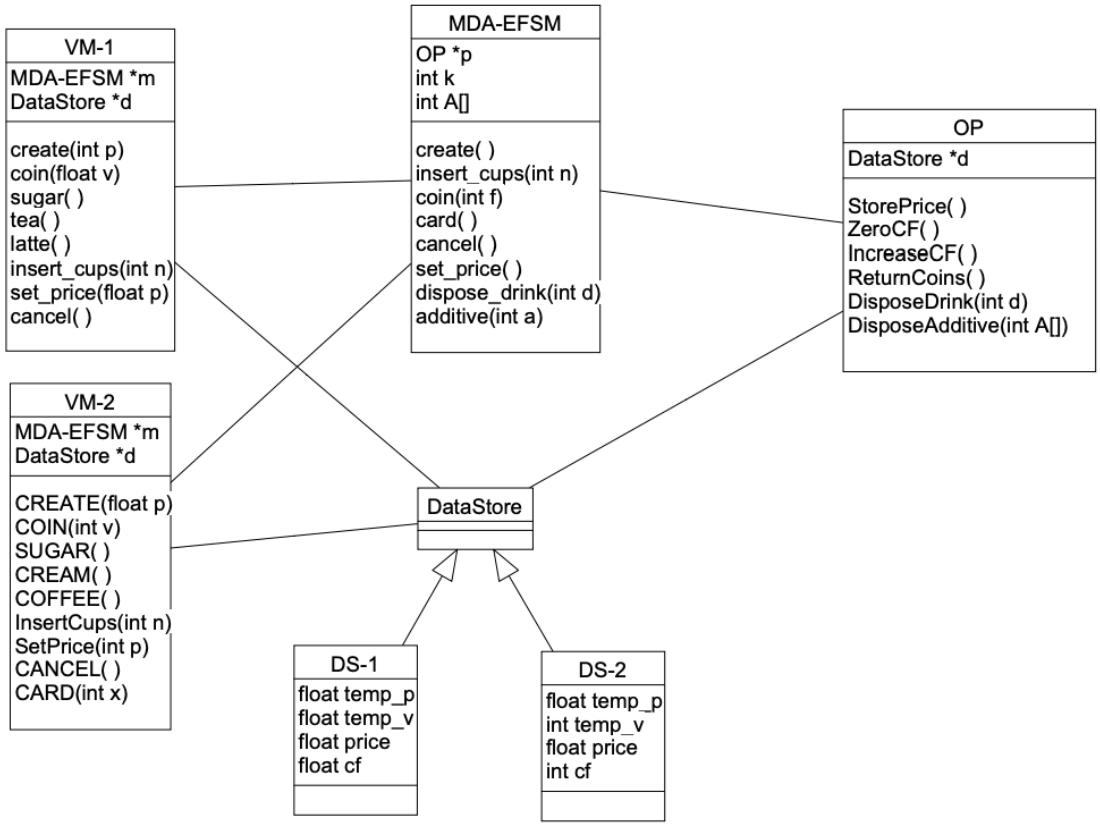
CS586 project report

Part1 MDA-EFSM model for the VM components



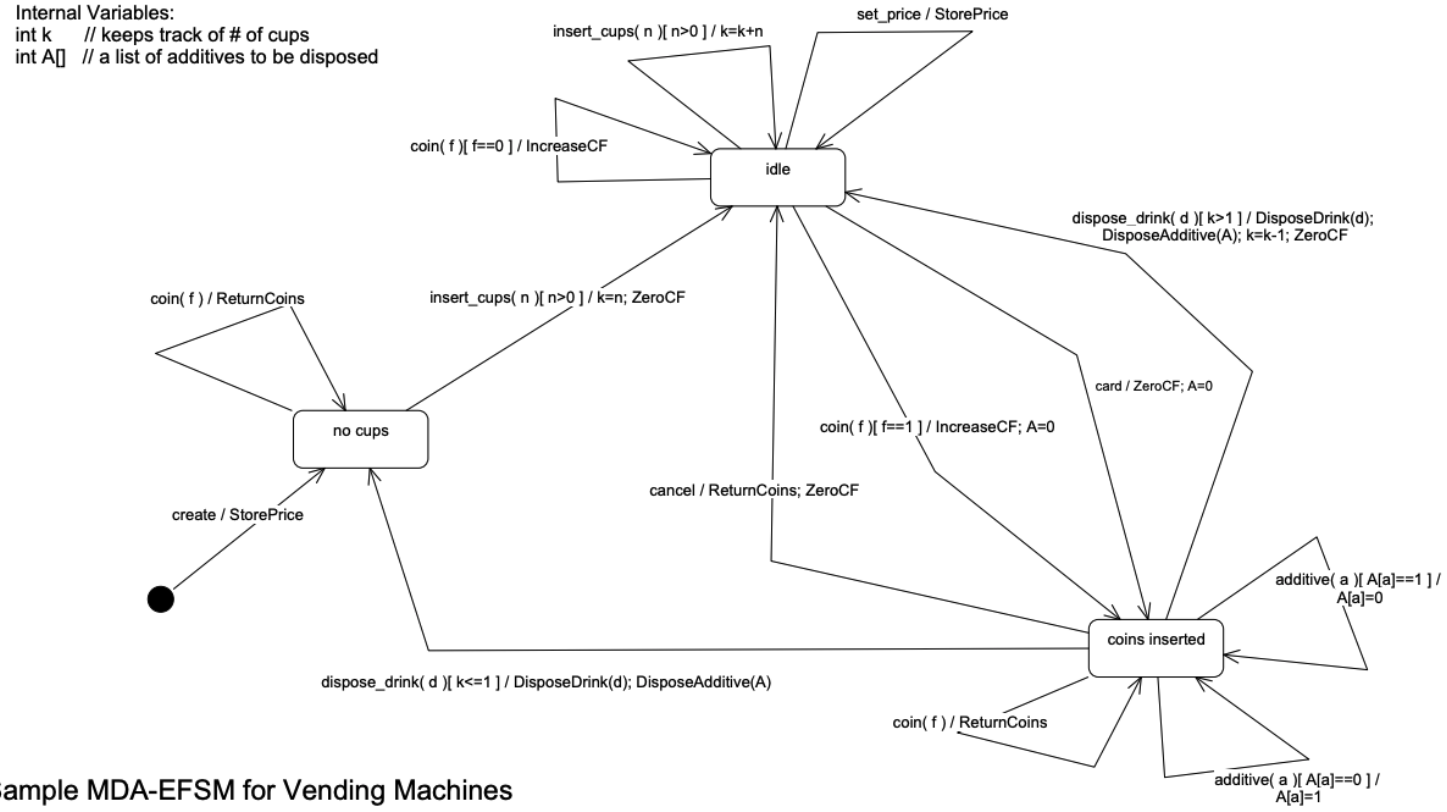
MDA-EFSM Events:

1. create()
2. insert_cups(int n) // n represents # of cups
3. coin(int f) // f=1: sufficient funds inserted for a drink
   // f=0: not sufficient funds for a drink
4. card()
5. cancel()
6. set_price()
7. dispose_drink(int d) // d represents a drink_id
8. additive(int a) // a represents additive id

MDA-EFSM 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(int d) // dispose a drink with d id
6. DisposeAdditive(int A[]) //dispose marked additives in A list,
   // where additive with i id is disposed when A[i]=1

c. A state diagram of the MDA-EFSM



Sample MDA-EFSM for Vending Machines

d. Pseudo-code of all operations of Input Processors of Vending Machines: VM-1 and VM-2

Vending-Machine-1

```
create(int 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(1);
}

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

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

insert_cups(int n) {
        m->insert_cups(n);
}

set_price(float p) {
        d->temp_p=p;
        m->set_price()
}

cancel() {
        m->cancel();
}
```

where,
m: pointer to the MDA-EFSM
d: pointer to the data store DS-1

In the data store:
cf: represents a cumulative fund
price: represents the price for a drink

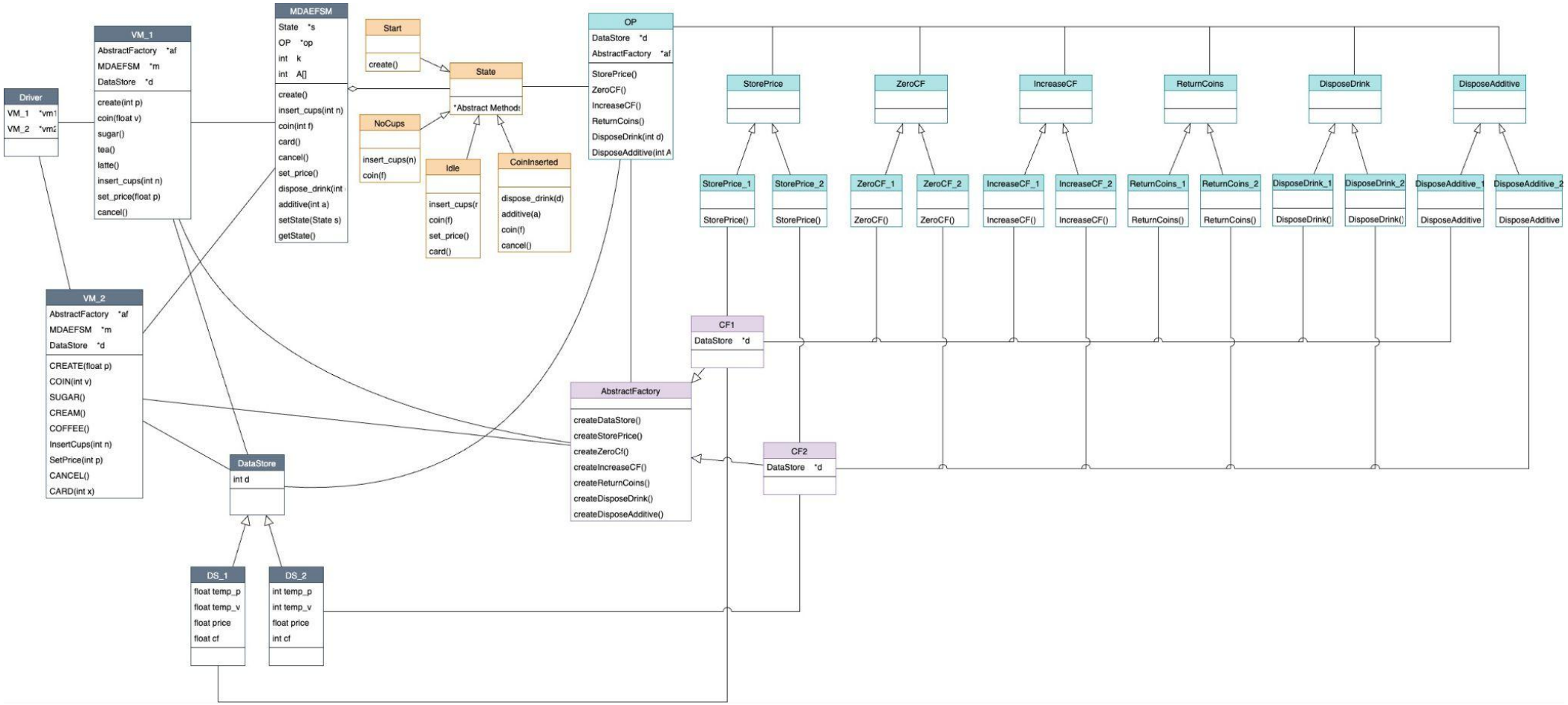
Vending-Machine-2

```
CREATE(float 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(int x) {
        if (x>=d->price) m->card();
}

SUGAR() {
        m->additive(2);
}

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

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

InsertCups(int n) {
        m->insert_cups(n);
}

SetPrice(int p) {
        d->temp_p=p;
        m->set_price()
}

CANCEL() {
        m->cancel();
}
```

where,
m: pointer to the MDA-EFSM
d: pointer to the data store DS-2

In the data store:
cf: represents a cumulative fund
price: represents the price for a drink

Part2 - Class diagram(s)



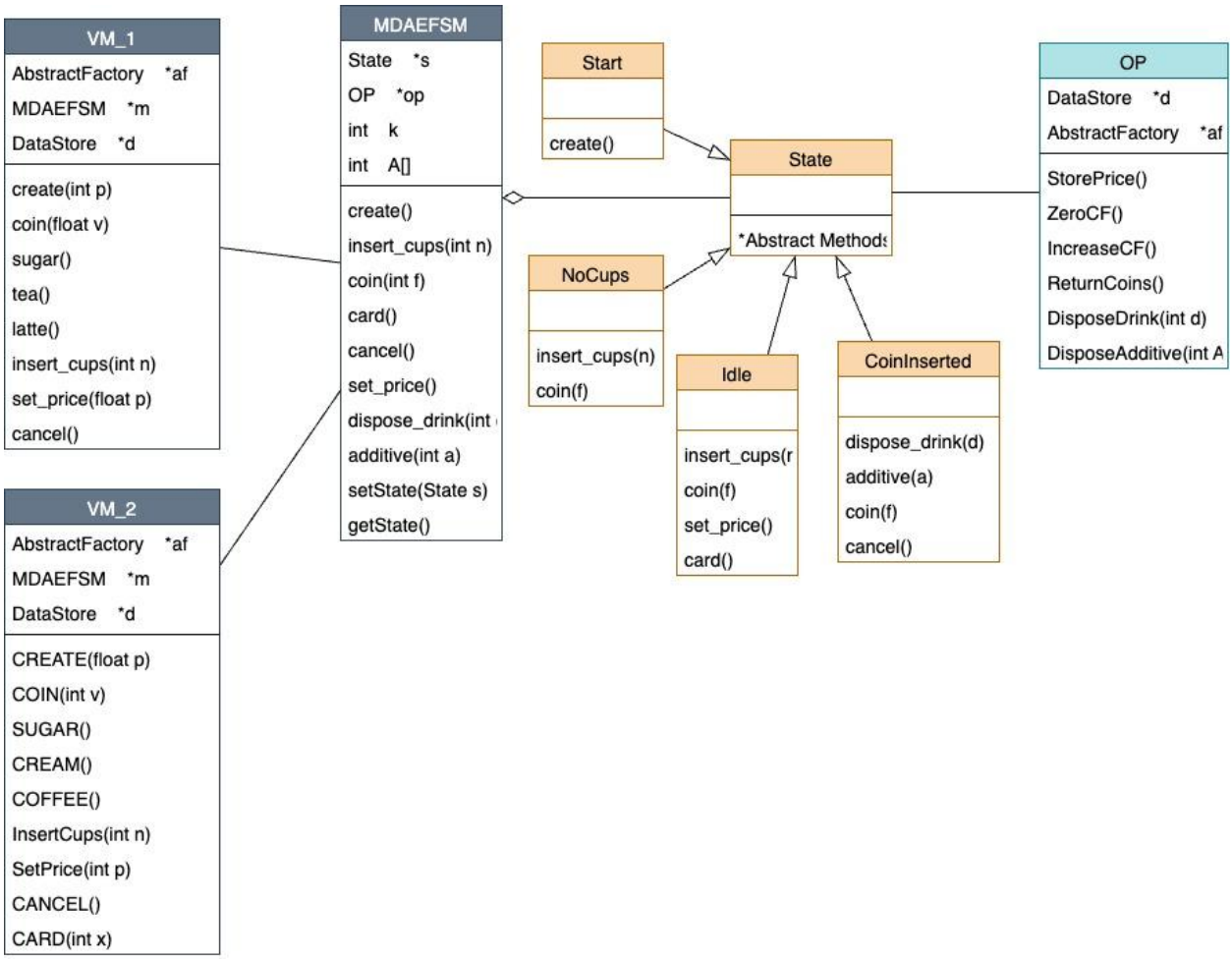
Yellow blocks are the State pattern, blue blocks are the Strategy pattern and purple blocks are the Abstract Factory pattern
In State pattern, typically there would be additional classes for each specific state, such as Sugar_pressed_1 or Sugar_Cream_pressed_2, etc. However, I realized that the CoinInserted state can handle all the specific states as long as the additives are clear. Therefore, I believe that having only four additional classes would be sufficient

Part3

a. Describe the purpose of the class, i.e., responsibilities.
b. Describe the responsibility of each operation supported by each class.

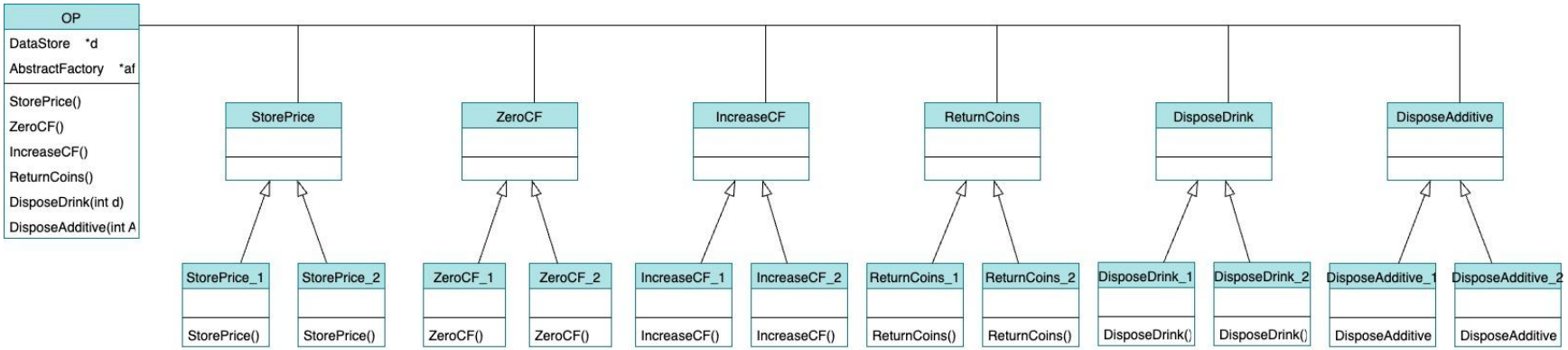
VM_1 and VM_2 are responsible for handling inputs from the users. Then it passed those inputs to MDAEFSM. MDAEFSM is the main context class. It is responsible for controlling the actions which can behave differently in each States. MDAEFSM call action based on States variable it stores. States classes responsibility are implementing correct actions. These classes also invoke actions in OP class. OP class handle the actual actions of the machine, which will continue in Strategy pattern.

State pattern closer look



OP is a class that handles actions to be done. It takes command and invoke the correct action, making other class less coupled from the action classes, also making adding or deleting actions easy. Moreover, it also consists if AbstractFactory and DataStore for the purpose of accessing CF and DS objects
StorePrice, ZeroCF, IncreaseCF, ReturnCoins, DisposeDrink, DisposeAdditive, these are all abstract class. The actions are defined in its child class. Child classes have number to indicate which machine are supposed to use them.

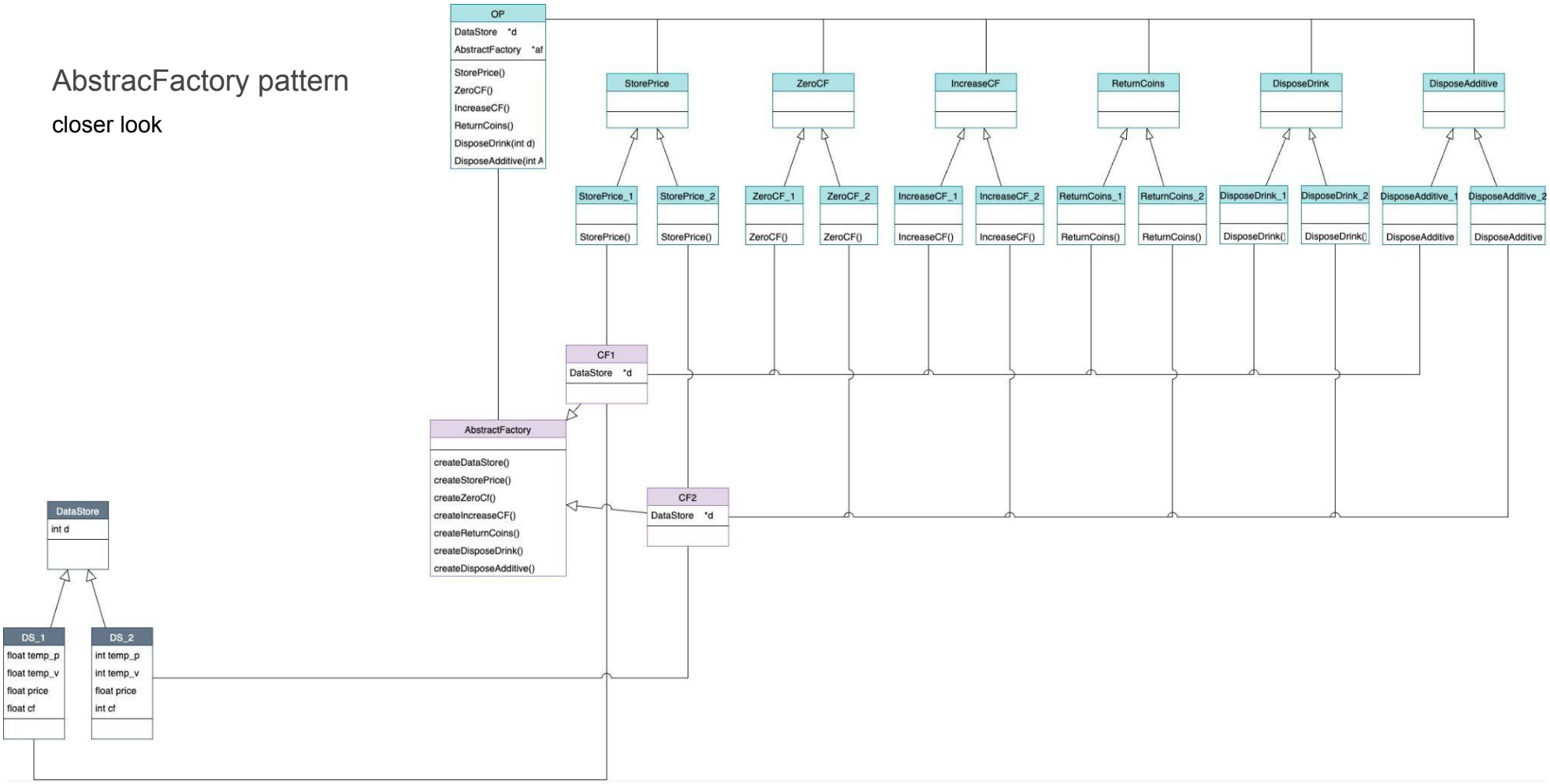
Strategy pattern closer look



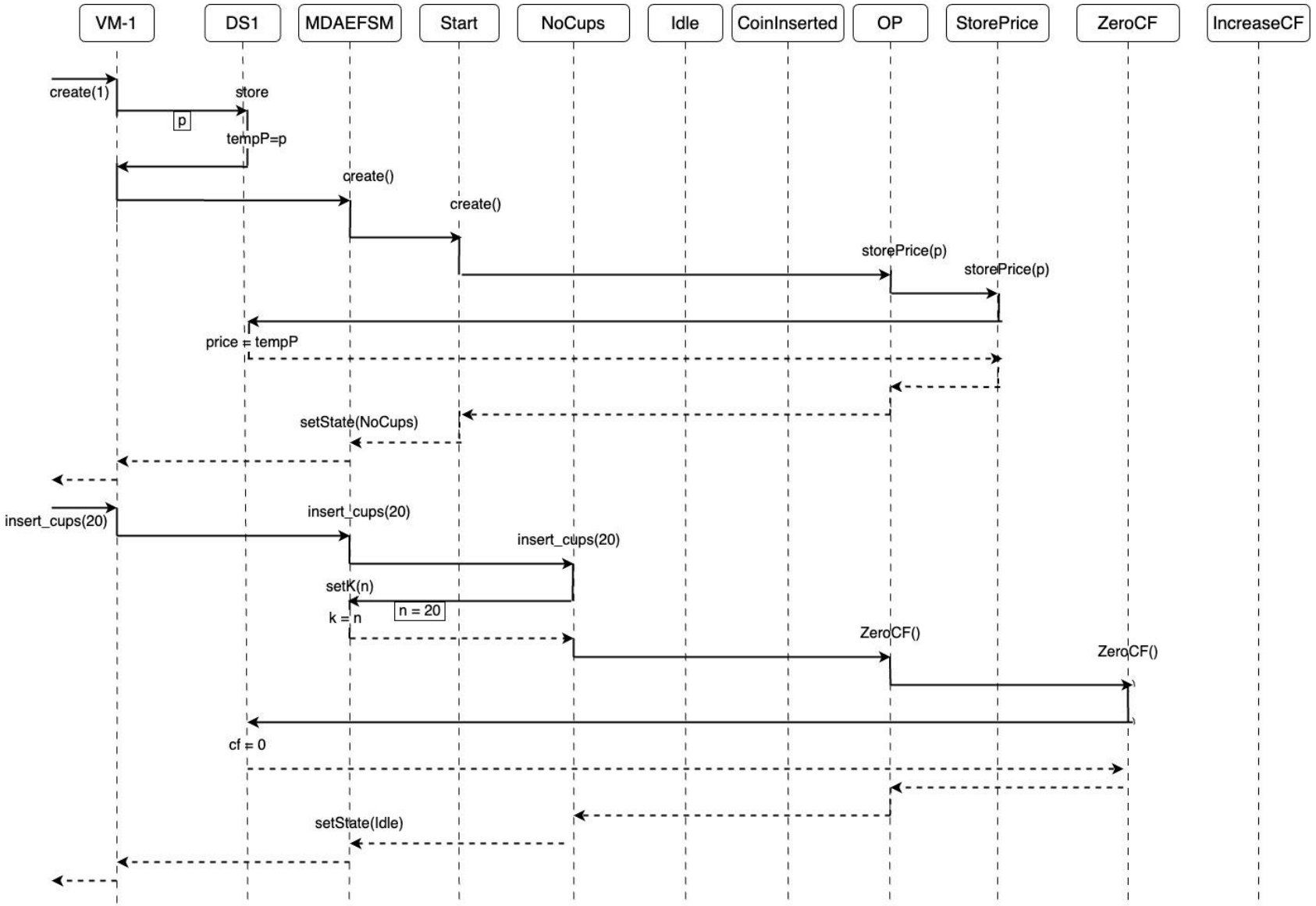
AbstractFactory class provide abstraction for CF1 and CF2. CF1 and CF2 stores their VM's DataStore and extends the abstraction. They create strategy objects and use them as needed. VM_1 and VM_2 also have these AbstractFactories because they only need to access actions specific to their needs.

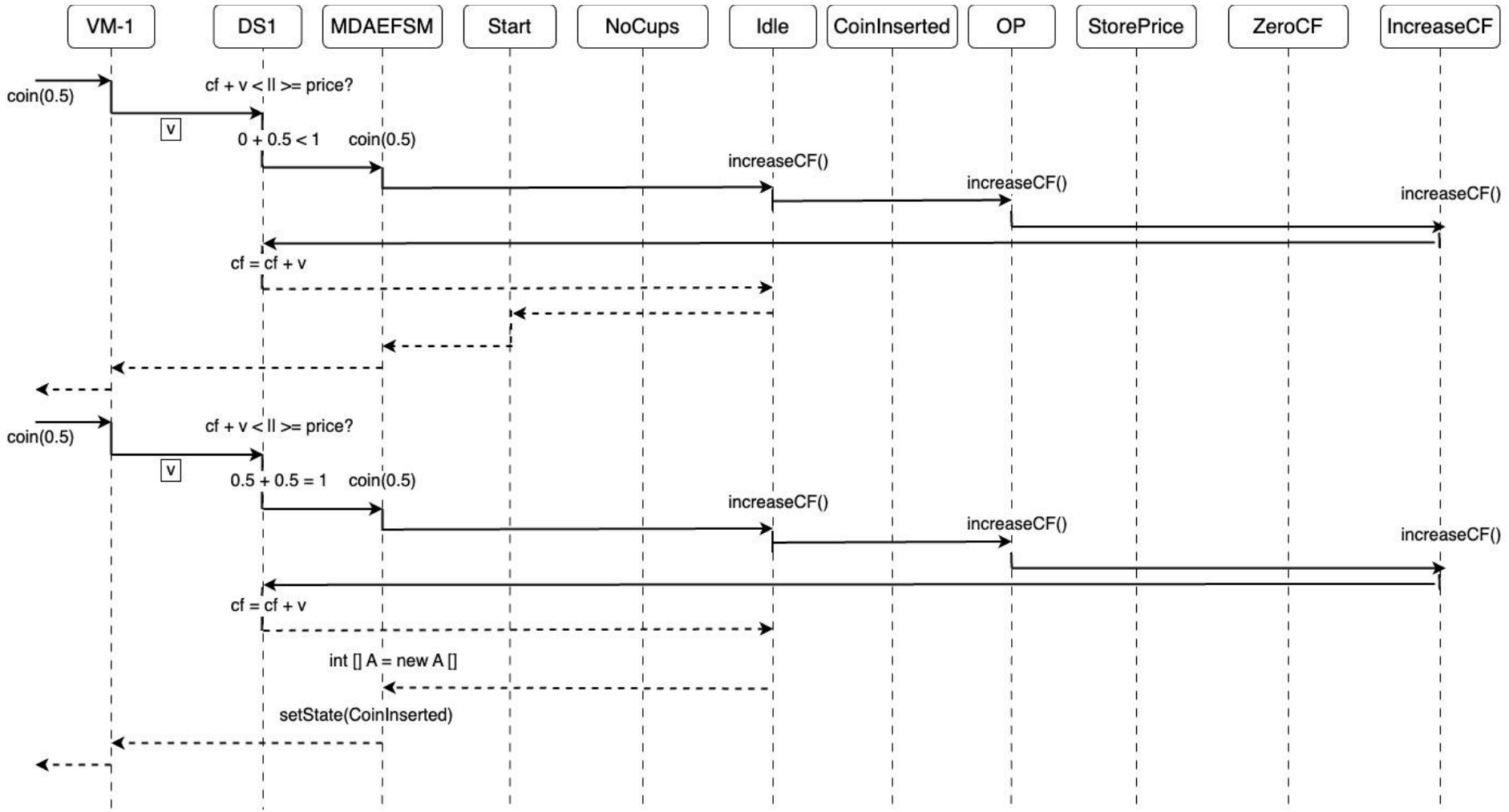
AbstracFactory pattern

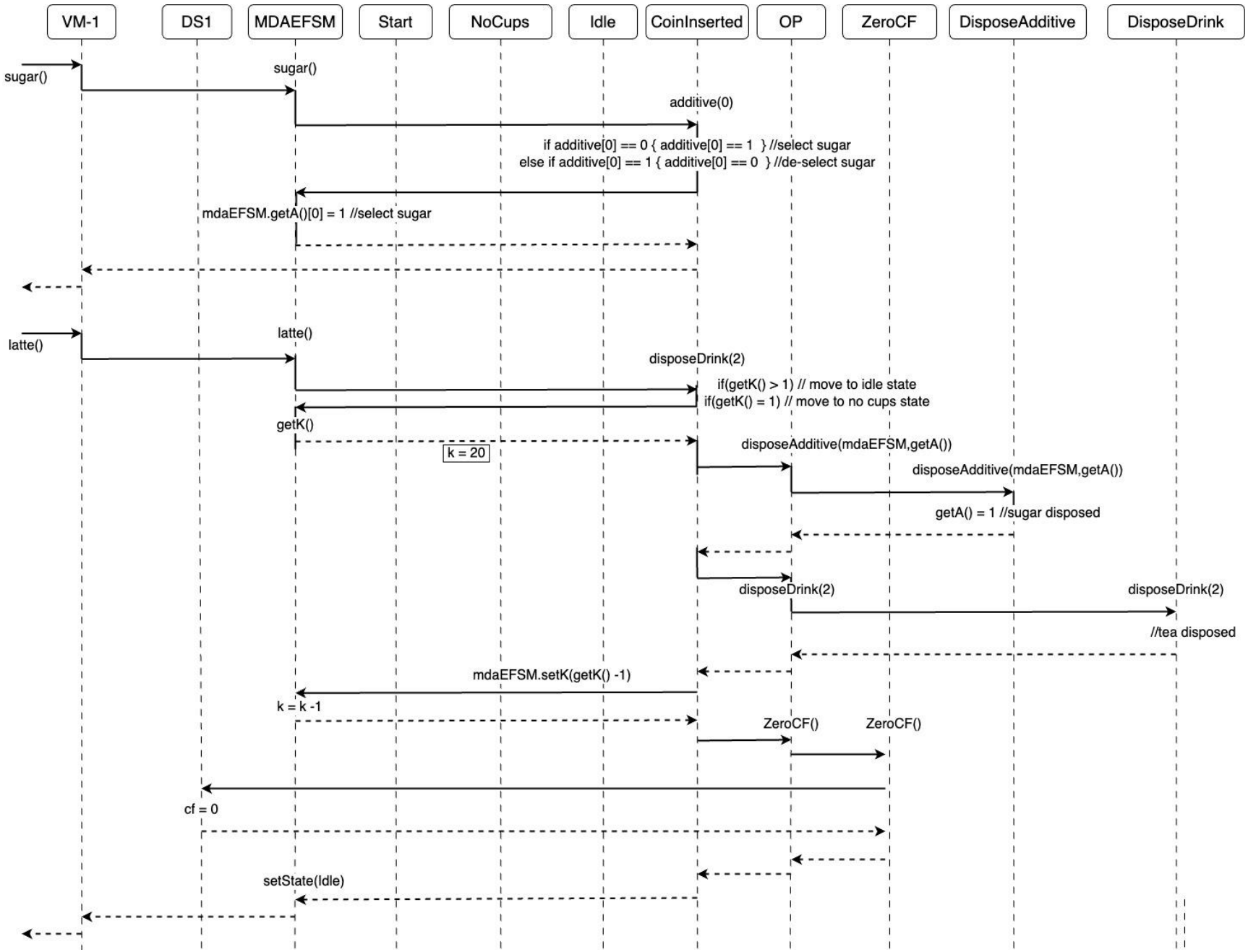
closer look



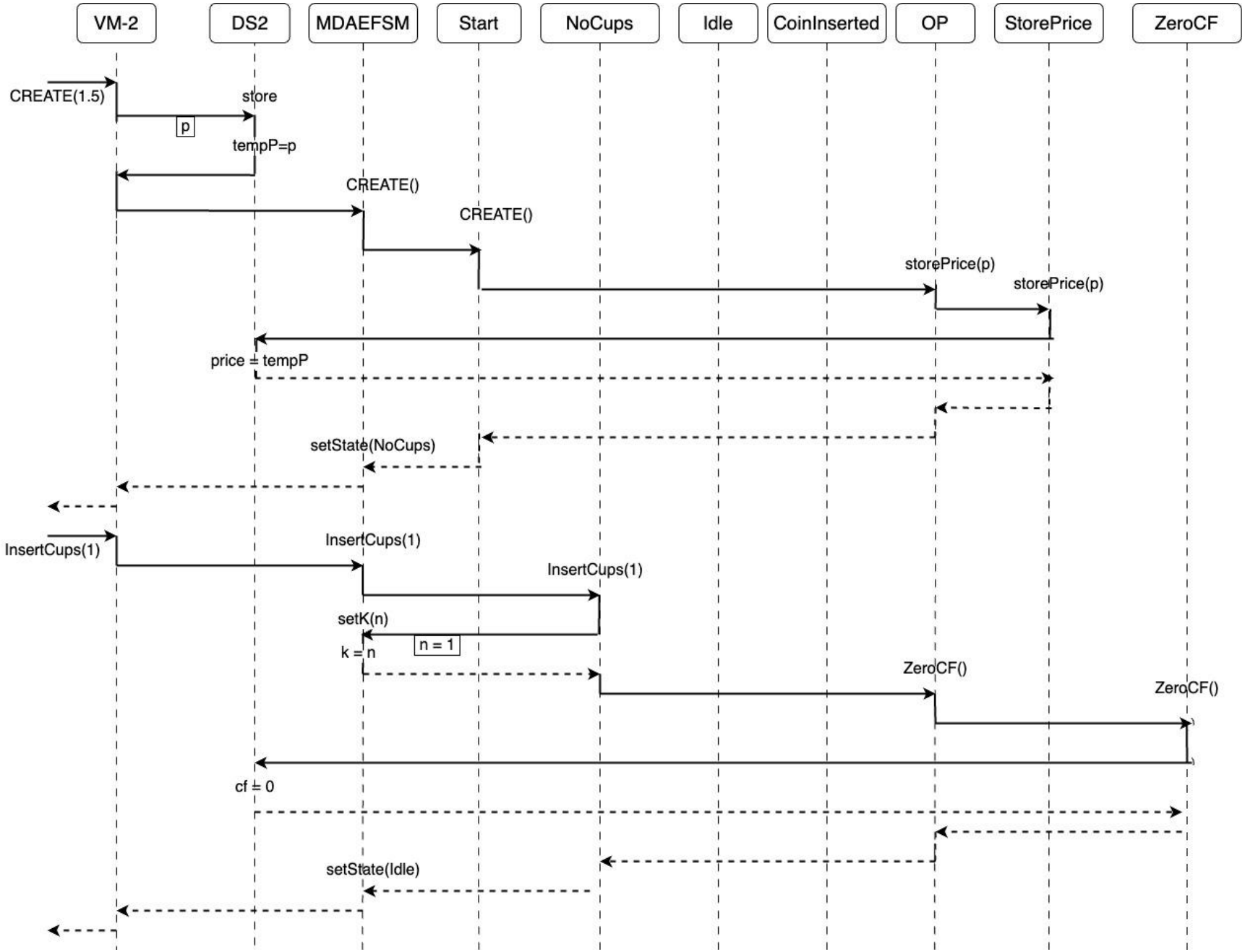
PART 4: VM_1 SEQUENCE DIAGRAM

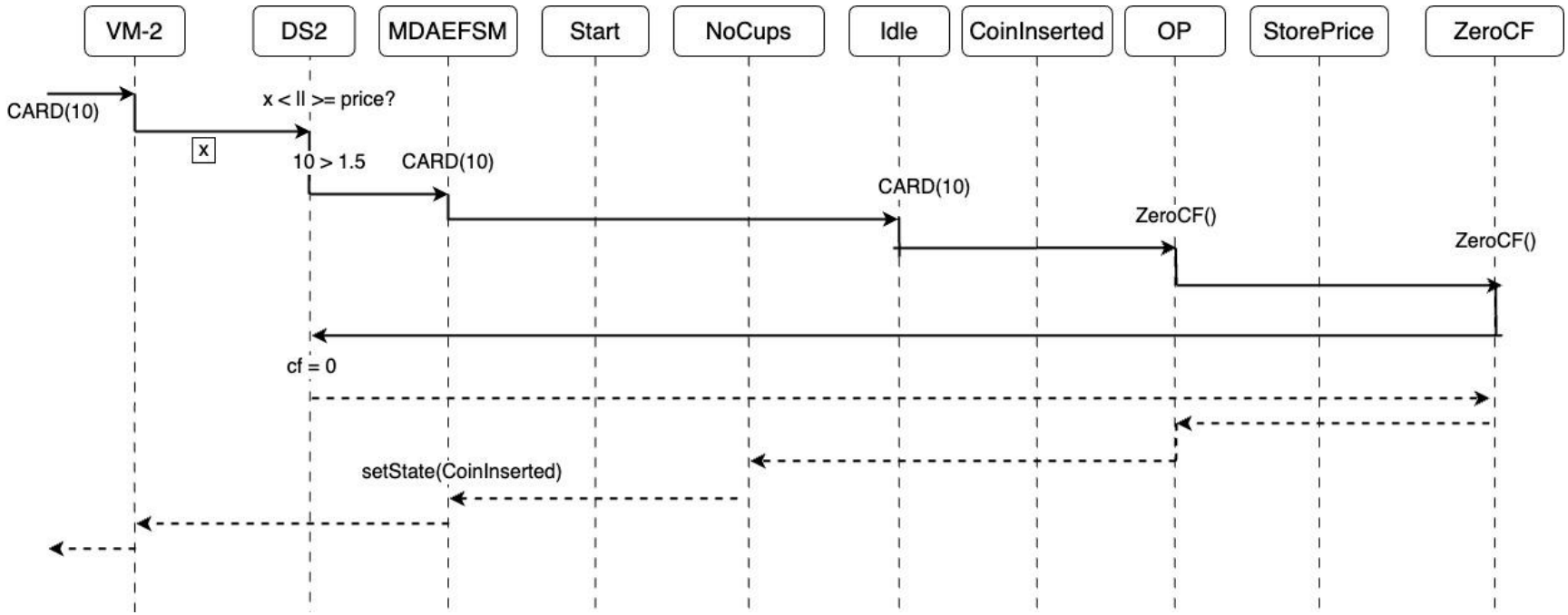


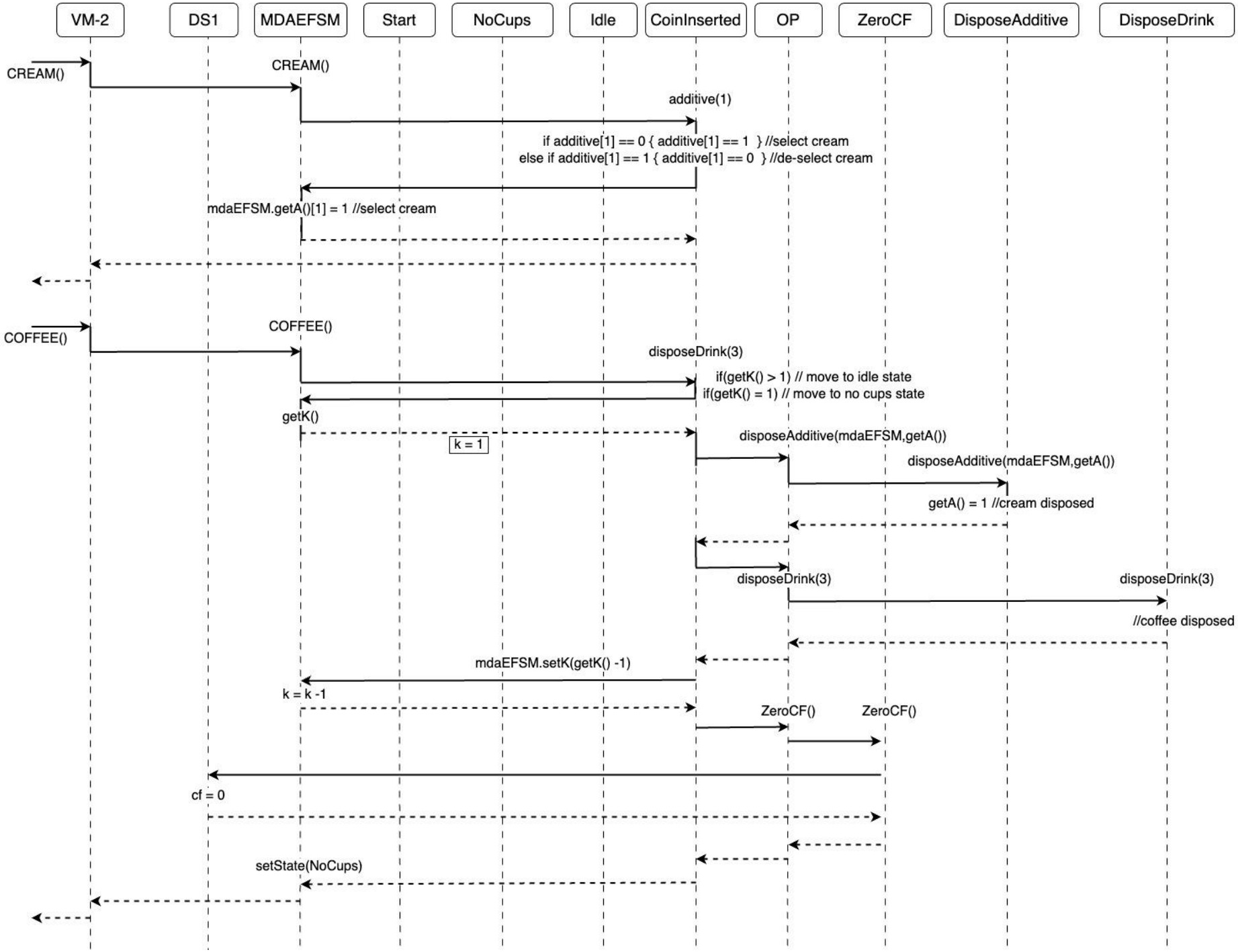

VM-1
DS1
MDAEFSM
Start
NoCups
Idle
CoinInserted
OP
StorePrice
ZeroCF
IncreaseCF
coin(0.5)
cf + v < ll >= price?
v
0 + 0.5 < 1
coin(0.5)
increaseCF()
increaseCF()
increaseCF()
cf = cf + v
coin(0.5)
cf + v < ll >= price?
v
0.5 + 0.5 = 1
coin(0.5)
increaseCF()
increaseCF()
increaseCF()
cf = cf + v
int [] A = new A []
setState(CoinInserted)


VM-1
DS1
MDAEFSM
Start
NoCups
Idle
CoinInserted
OP
ZeroCF
DisposeAdditive
DisposeDrink
sugar()
sugar()
additive(0)
if additive[0] == 0 { additive[0] == 1 } //select sugar
else if additive[0] == 1 { additive[0] == 0 } //de-select sugar
mdaEFSM.getA()[0] = 1 //select sugar
latte()
latte()
disposeDrink(2)
if(getK() > 1) // move to idle state
if(getK() = 1) // move to no cups state
getK()
k = 20
disposeAdditive(mdaEFSM,getA())
disposeAdditive(mdaEFSM,getA())
getA() = 1 //sugar disposed
disposeDrink(2)
disposeDrink(2)
//tea disposed
mdaEFSM.setK(getK() -1)
k = k -1
ZeroCF()
ZeroCF()
cf = 0
setState(Idle)

PART 4: VM_2 SEQUENCE DIAGRAM




VM-2
DS2
MDAEFSM
Start
NoCups
Idle
CoinInserted
OP
StorePrice
ZeroCF
CARD(10)
x < ll >= price?
x
10 > 1.5
CARD(10)
CARD(10)
ZeroCF()
ZeroCF()
cf = 0
setState(CoinInserted)


VM-2
DS1
MDAEFSM
Start
NoCups
Idle
CoinInserted
OP
ZeroCF
DisposeAdditive
DisposeDrink
CREAM()
CREAM()
additive(1)
if additive[1] == 0 { additive[1] == 1 } //select cream
else if additive[1] == 1 { additive[1] == 0 } //de-select cream
mdaEFSM.getA()[1] = 1 //select cream
COFFEE()
COFFEE()
disposeDrink(3)
if(getK() > 1) // move to idle state
if(getK() = 1) // move to no cups state
getK()
k = 1
disposeAdditive(mdaEFSM,getA())
disposeAdditive(mdaEFSM,getA())
getA() = 1 //cream disposed
disposeDrink(3)
disposeDrink(3)
//coffee disposed
mdaEFSM.setK(getK() -1)
k = k -1
ZeroCF()
ZeroCF()
cf = 0
setState(NoCups)