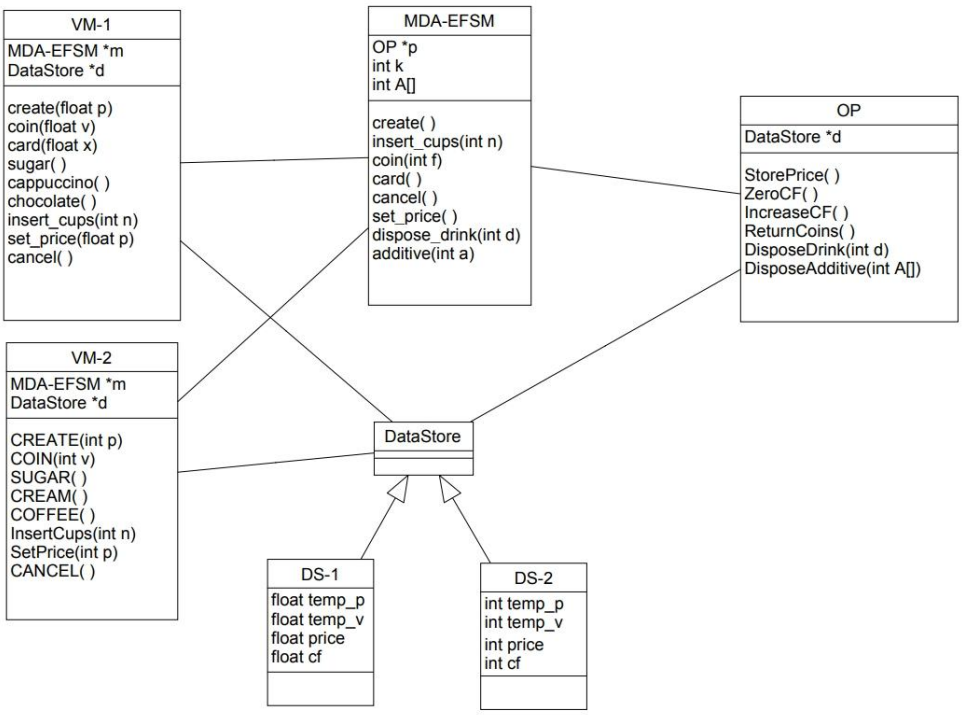
1. MDA-EFSM Model



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:

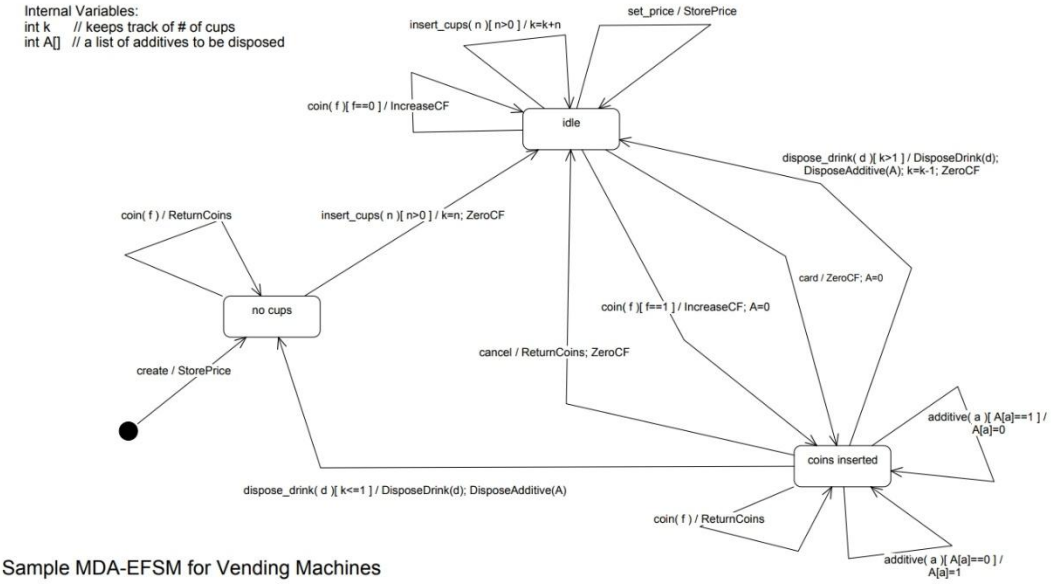
1. StorePrice()
2. ZeroCF() // zero Cumulative Fund *cf*
3. IncreaseCF() // increase Cumulative Fund *cf*
4. ReturnCoins() // return coins inserted for a drink
5. DisposeDrink(int d) // dispose a drink with *d* id
6. DisposeAdditive(int A[]) //dispose marked additives in *A* list,
   // where additive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

Vending-Machine-1	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 a price for a drink

```
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);
}

card(float x) {
        if (x>=d->price) m->card();
}
sugar() {
        m->additive(1);
}

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

chocolate() {
        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();
}
```

Vending-Machine-2	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 a price for a drink

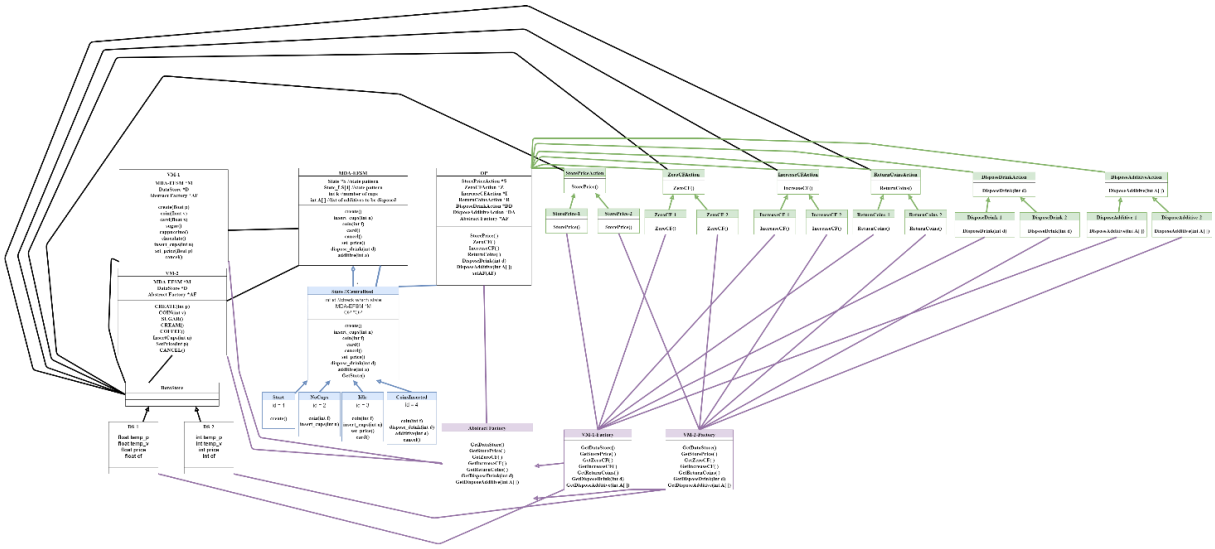
```
CREATE(int p) {
        d->temp_p=p;
        m->create();
}

COIN(int v) {
        d->temp_v=v;
        if (d->cf+v>=d->price) m->coin(1);
        else m->coin(0);
}

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();
}
```



Sample MDA-EFSM for Vending Machines

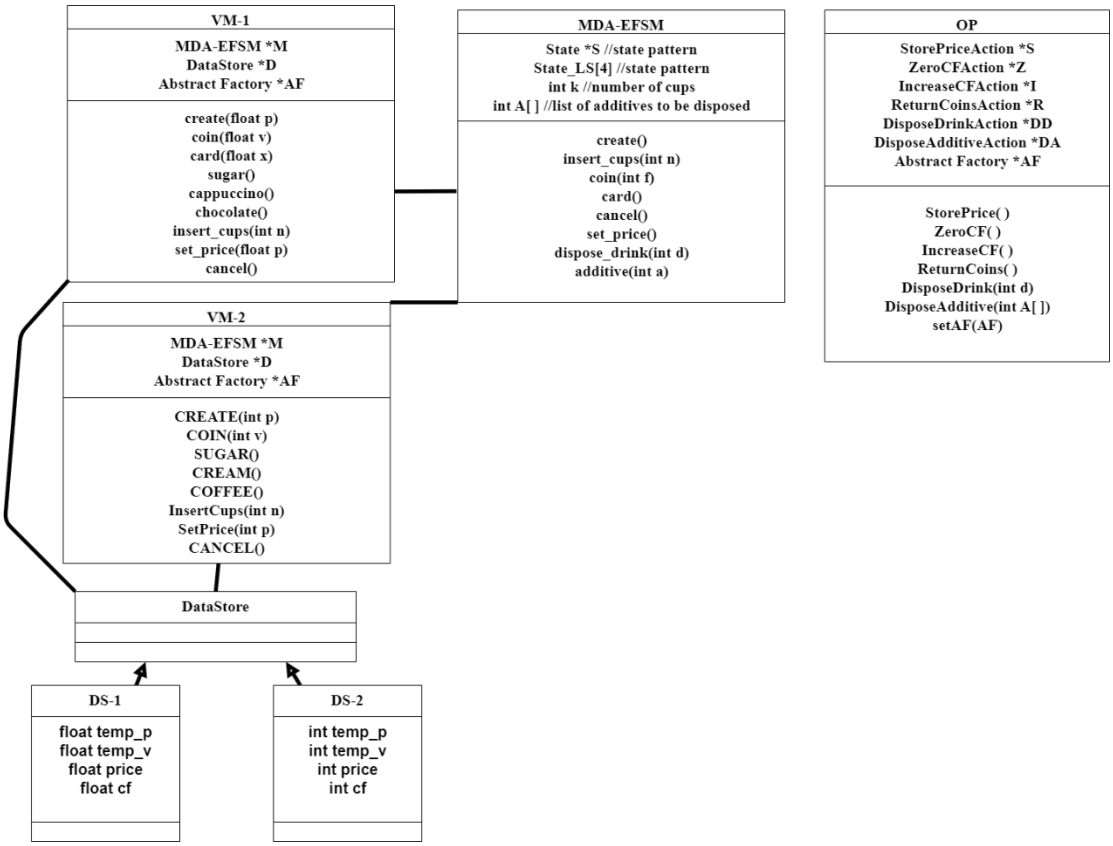
2. Class diagram(s) and Responsibilities

a. Complete class diagram



This complete class diagram just wants to show how different patterns can connect to each other. (Detail are show below)

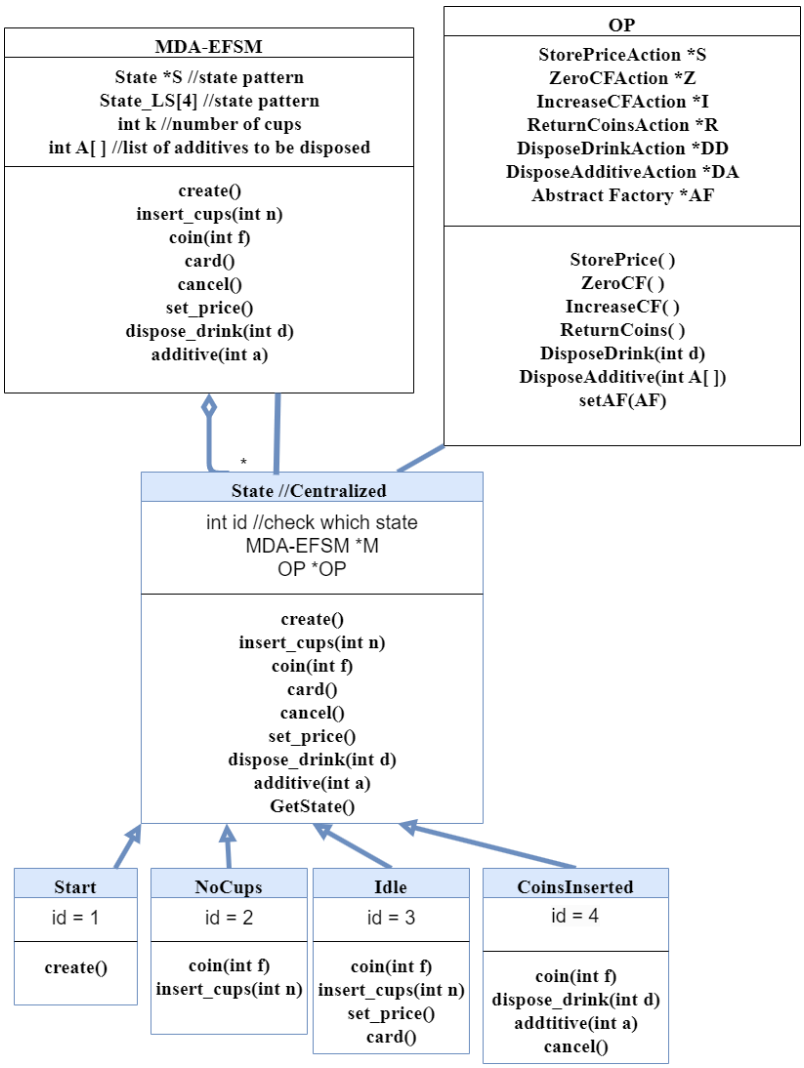
b. MDA-EFSM:



Class responsibilities:

1. **VM-1 and VM-2** are contain VM operation (connect with the abstract pattern and Datastore).
2. **DS-1 and DS-2** are store VM data (connect with the concrete action).
3. **MDA-EFSM** contain Meta event and changing state (Connect with the state pattern)
4. **OP** contain Meta action. (connect with the state pattern, abstract pattern, strategy pattern)

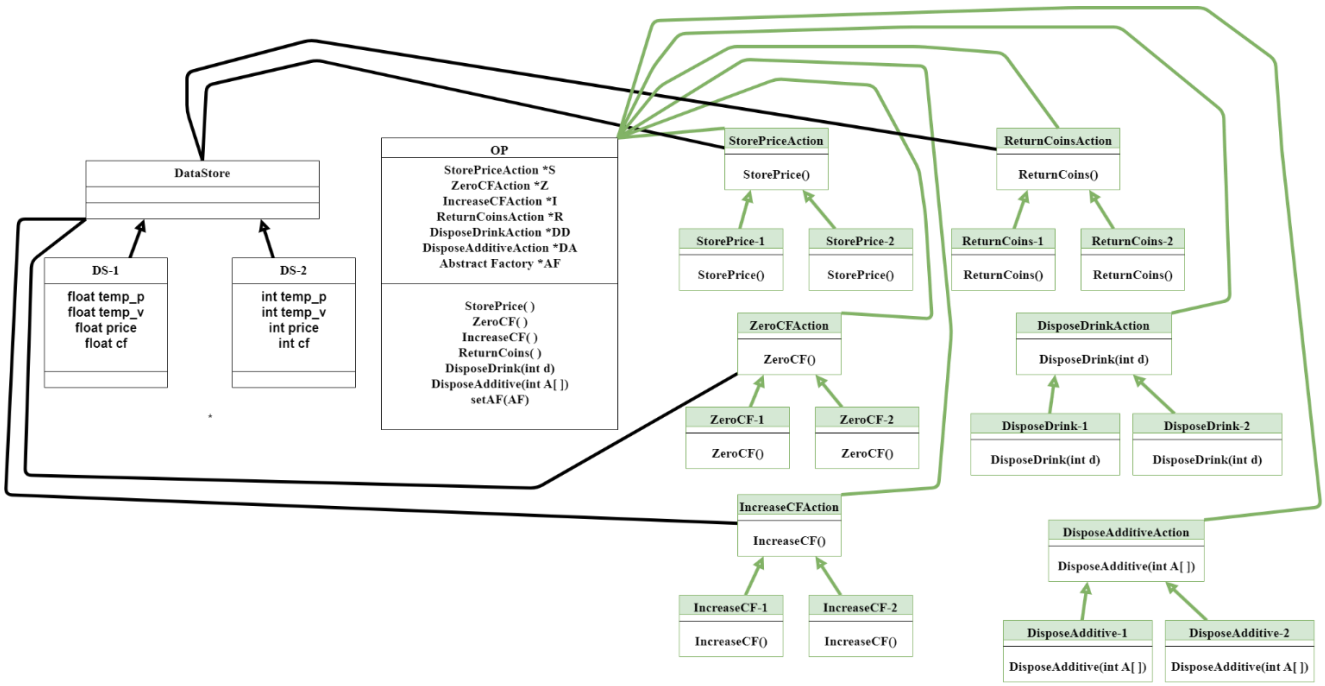
c. State pattern:



Class responsibilities:

1. **State** is an abstract class and have the connection with MDA-EFSM and OP.
2. **Start** contain the event that will operate in start state
3. **Nocups** contain the event that will operate in start state
4. **Idle** contain the event that will operate in start state
5. **CoinInserted** contain the event that will operate in start state

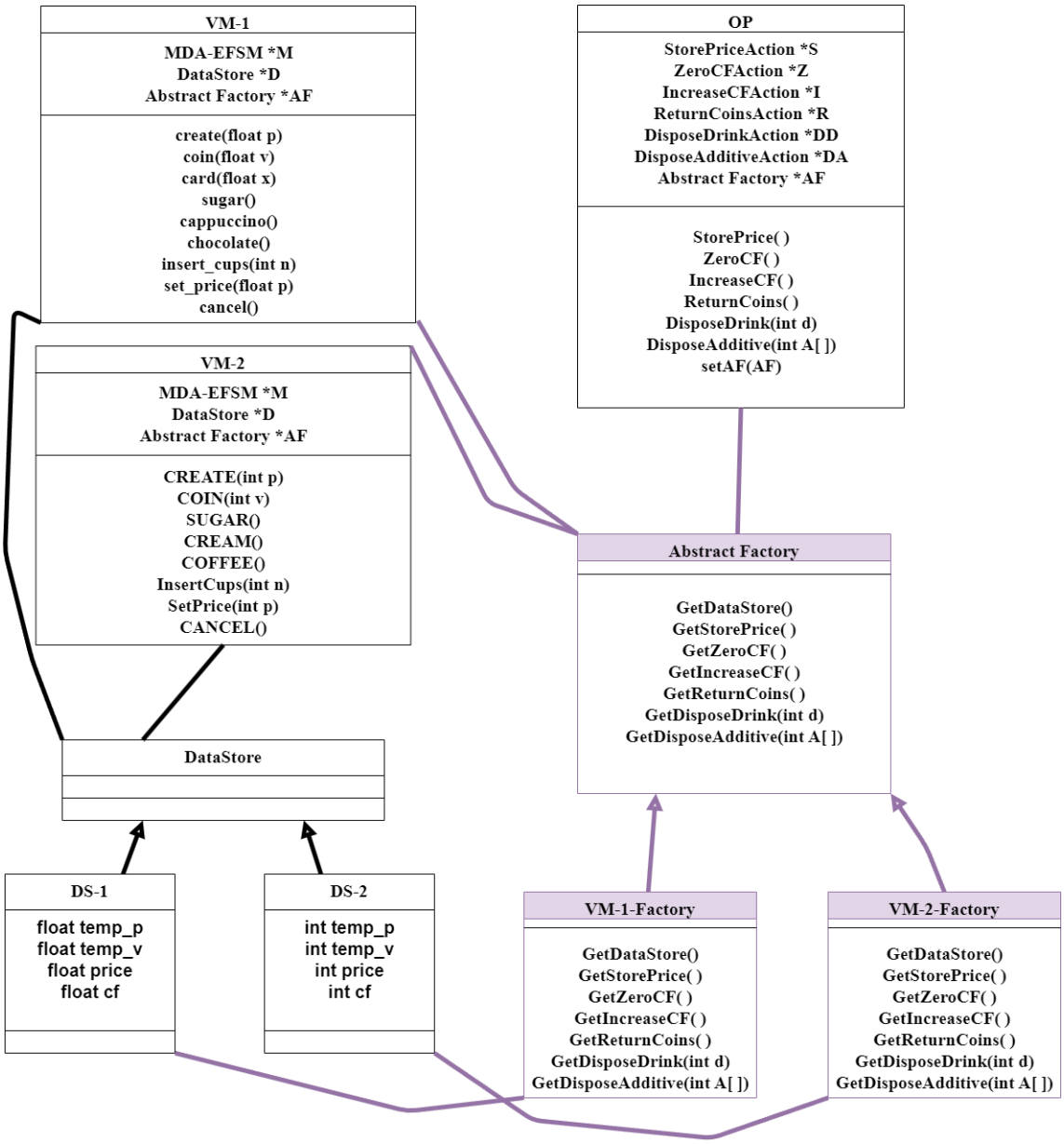
d. Strategy pattern:



Class responsibilities:

1. **Action** class (such as StorePriceAction) are all abstract class and OP has the pointer to all of them.
2. **Concrete** class (such as StorePrice-1) is an operation class which means that these classes will really do something inside the function not just calling the other functions.
3. **Concrete class** will connect to the concrete factory.
4. Some of the **Action** class will need to connect to the datastore because these classes need to do some work with the data.

e. Abstract factory pattern:



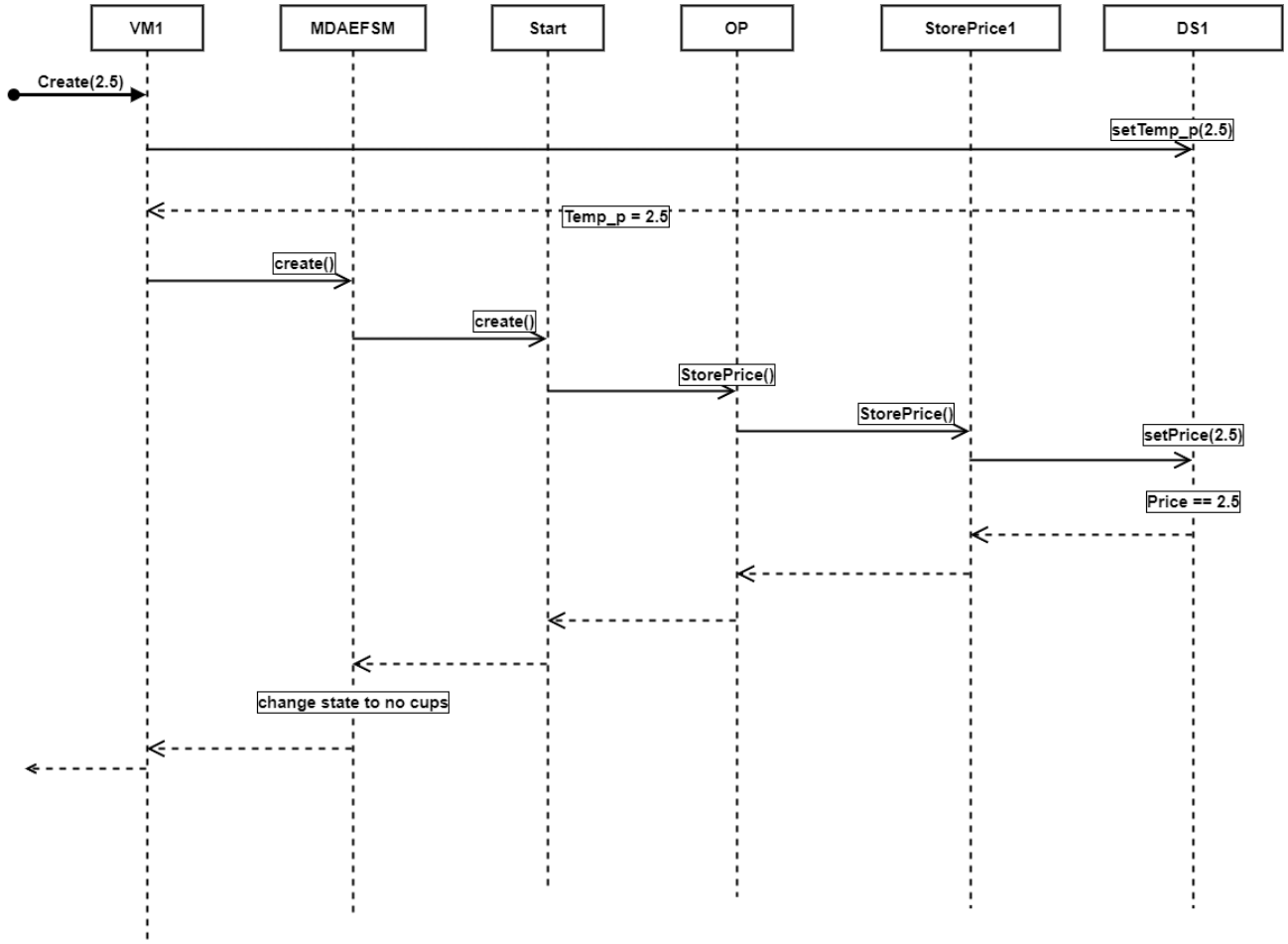
Class responsibilities:

1. **Abstract Factory** is an abstract class have the connect with VM-1 VM-2 and OP
2. **VM-1 VM-2 factory** are VM-1 VM-2 concrete factory, these two factories will create VM datastore object (such as DS1) and concrete action object(such as StorePrice1) to the correct VM-1 VM-2.

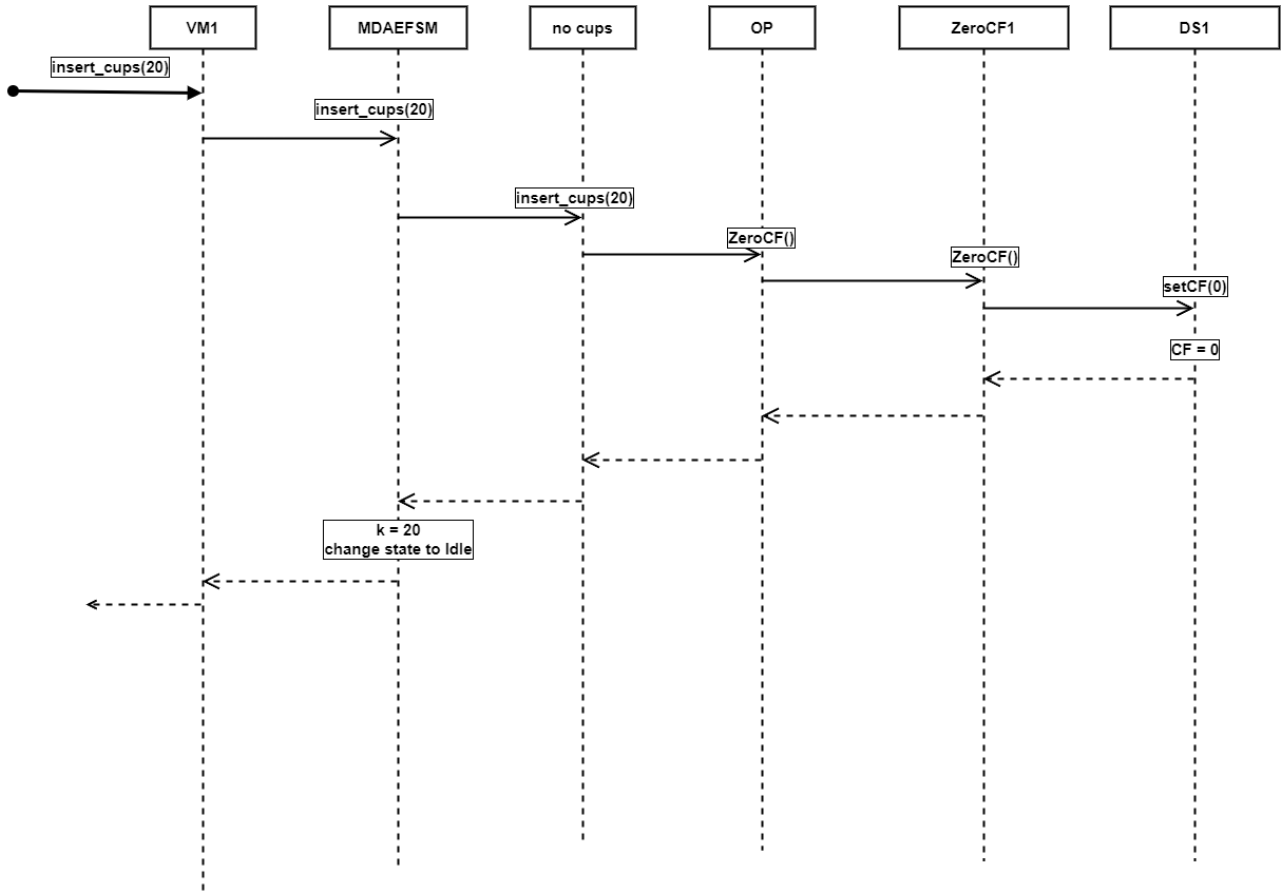
3. Sequence diagram:

Scenario1:

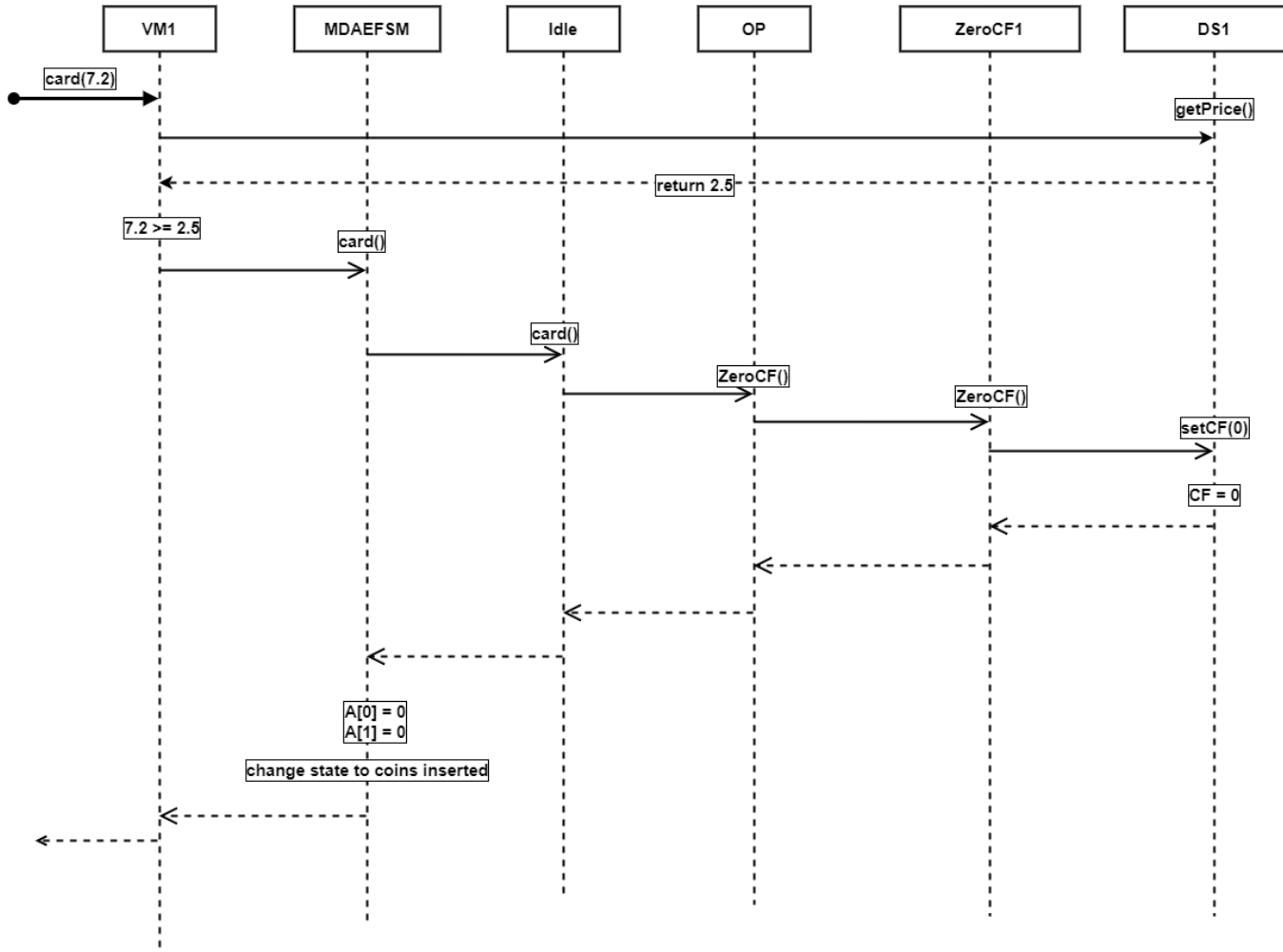
create(2.5):



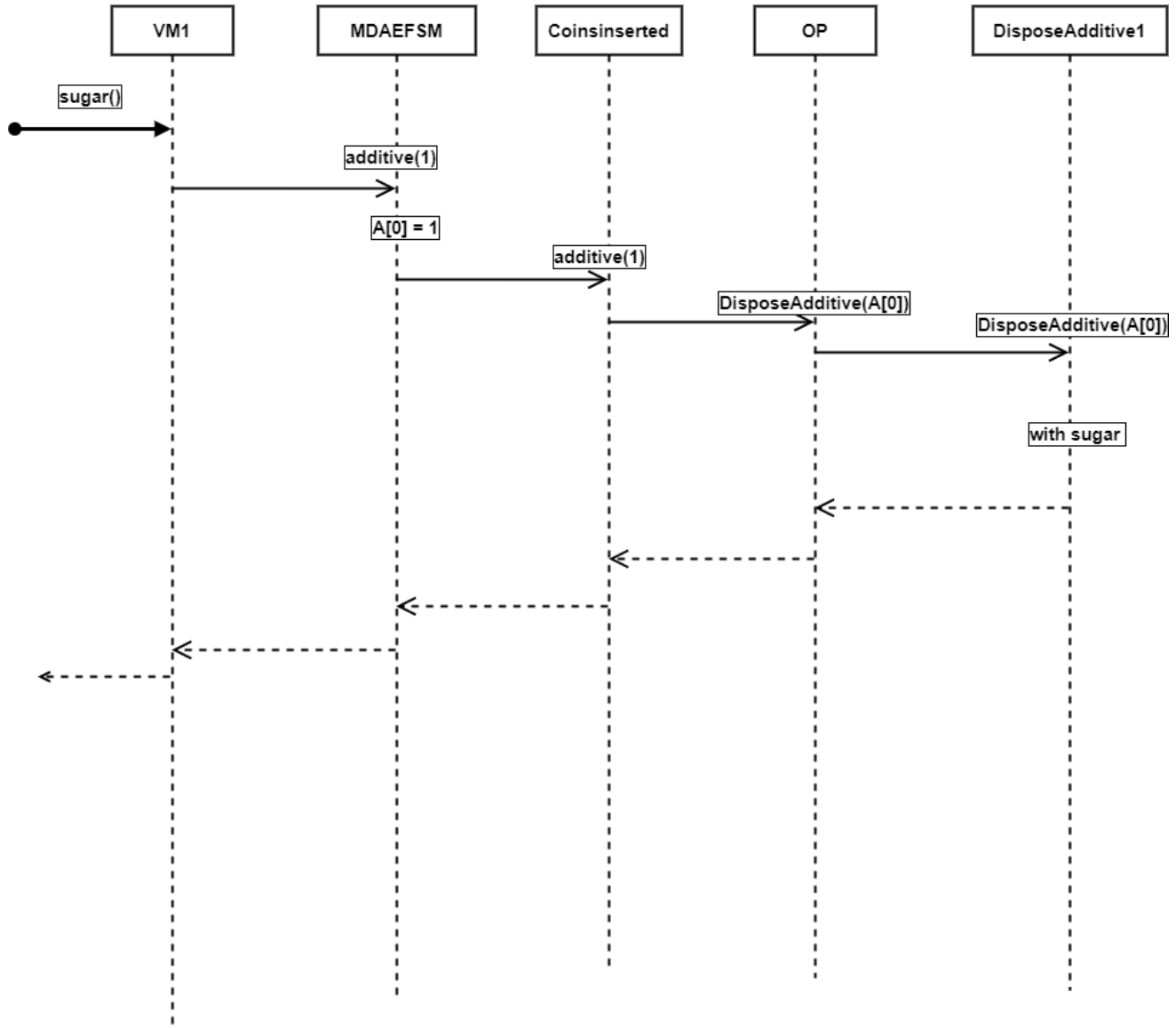
Insert_cups(20):



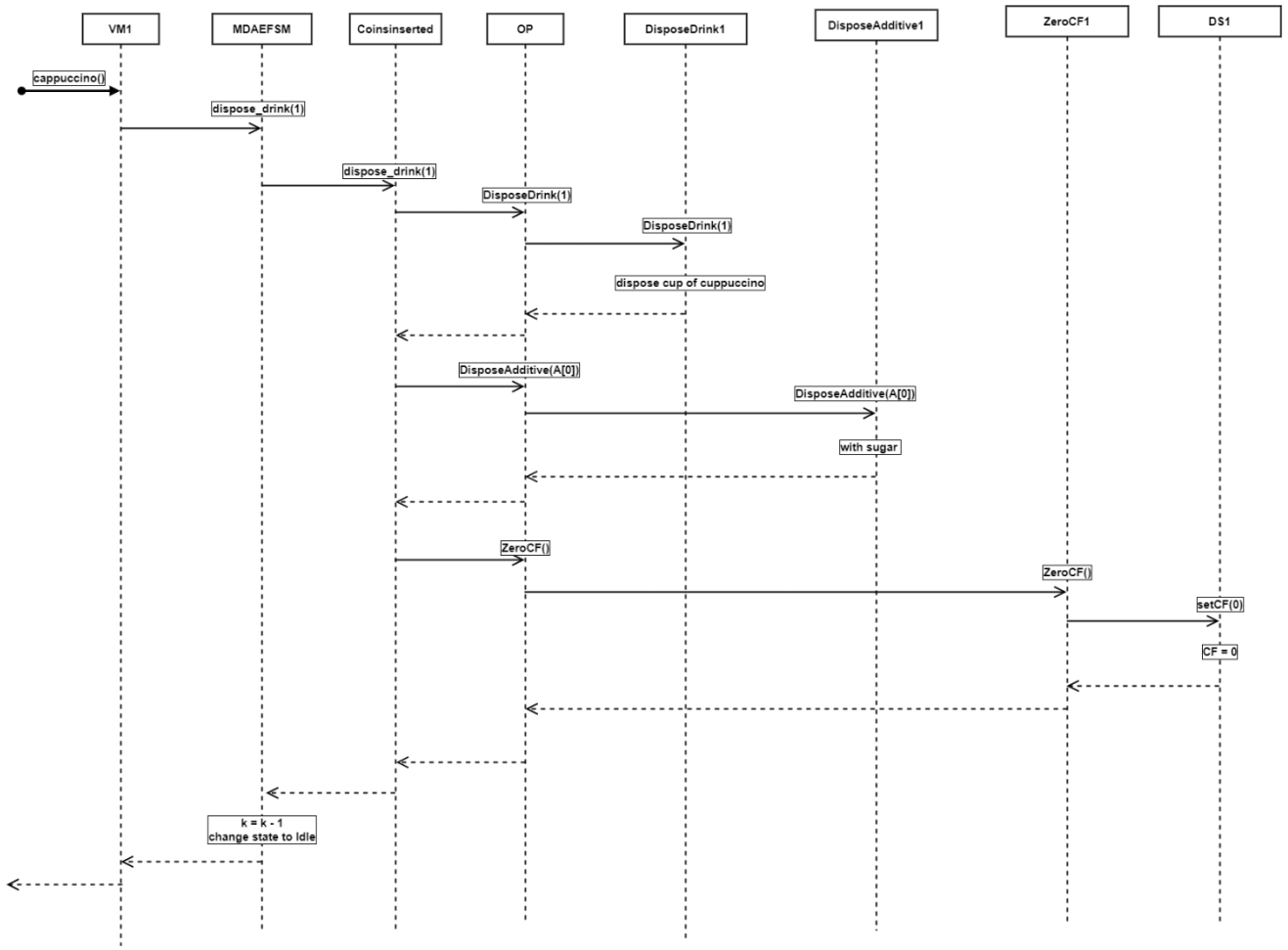
card(7.2):


VM1
MDAEFSM
Idle
OP
ZeroCF1
DS1
card(7.2)
getPrice()
return 2.5
7.2 >= 2.5
card()
card()
ZeroCF()
ZeroCF()
setCF(0)
CF = 0
A[0] = 0
A[1] = 0
change state to coins inserted

sugar():

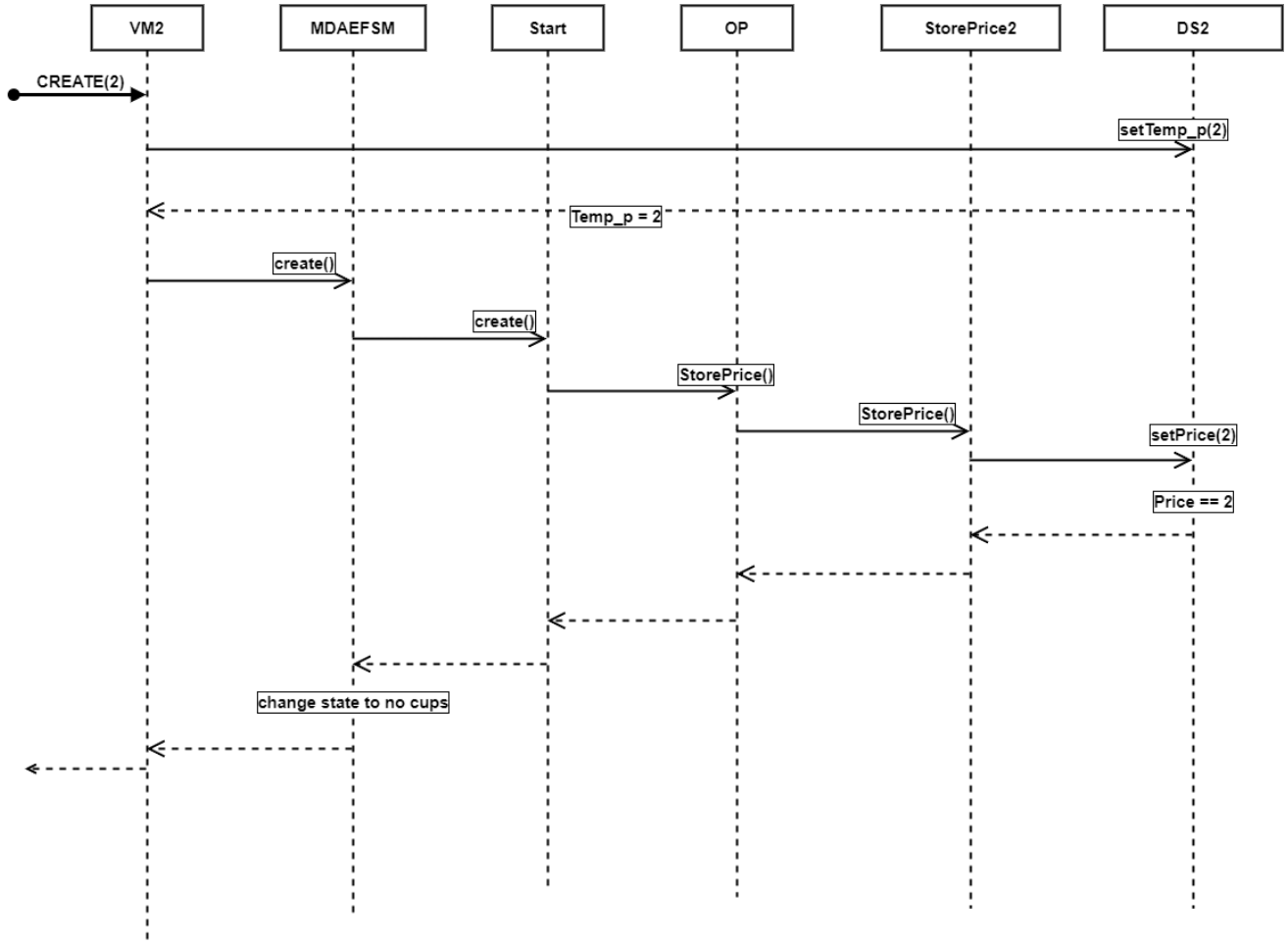

VM1
MDAEFSM
Coinsinserted
OP
DisposeAdditive1
sugar()
additive(1)
A[0] = 1
additive(1)
DisposeAdditive(A[0])
DisposeAdditive(A[0])
with sugar

cappuccino():

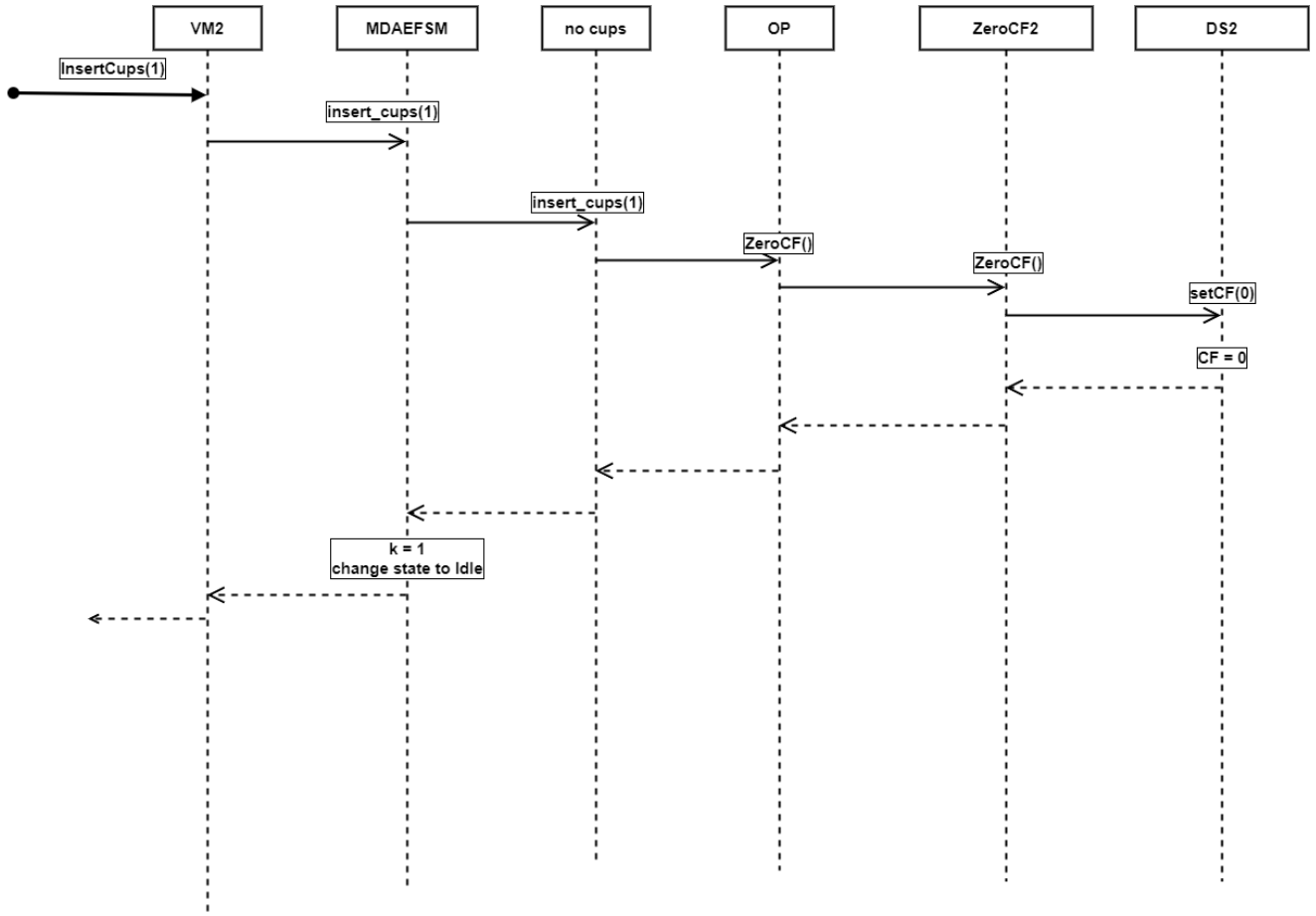

VM1
MDAEFSM
Coinsinserted
OP
DisposeDrink1
DisposeAdditive1
ZeroCF1
DS1
cappuccino()
dispose_drink(1)
dispose_drink(1)
DisposeDrink(1)
DisposeDrink(1)
dispose cup of cuppuccino
DisposeAdditive(A[0])
DisposeAdditive(A[0])
with sugar
ZeroCF()
ZeroCF()
setCF(0)
CF = 0
k = k - 1
change state to Idle

Scenario2:

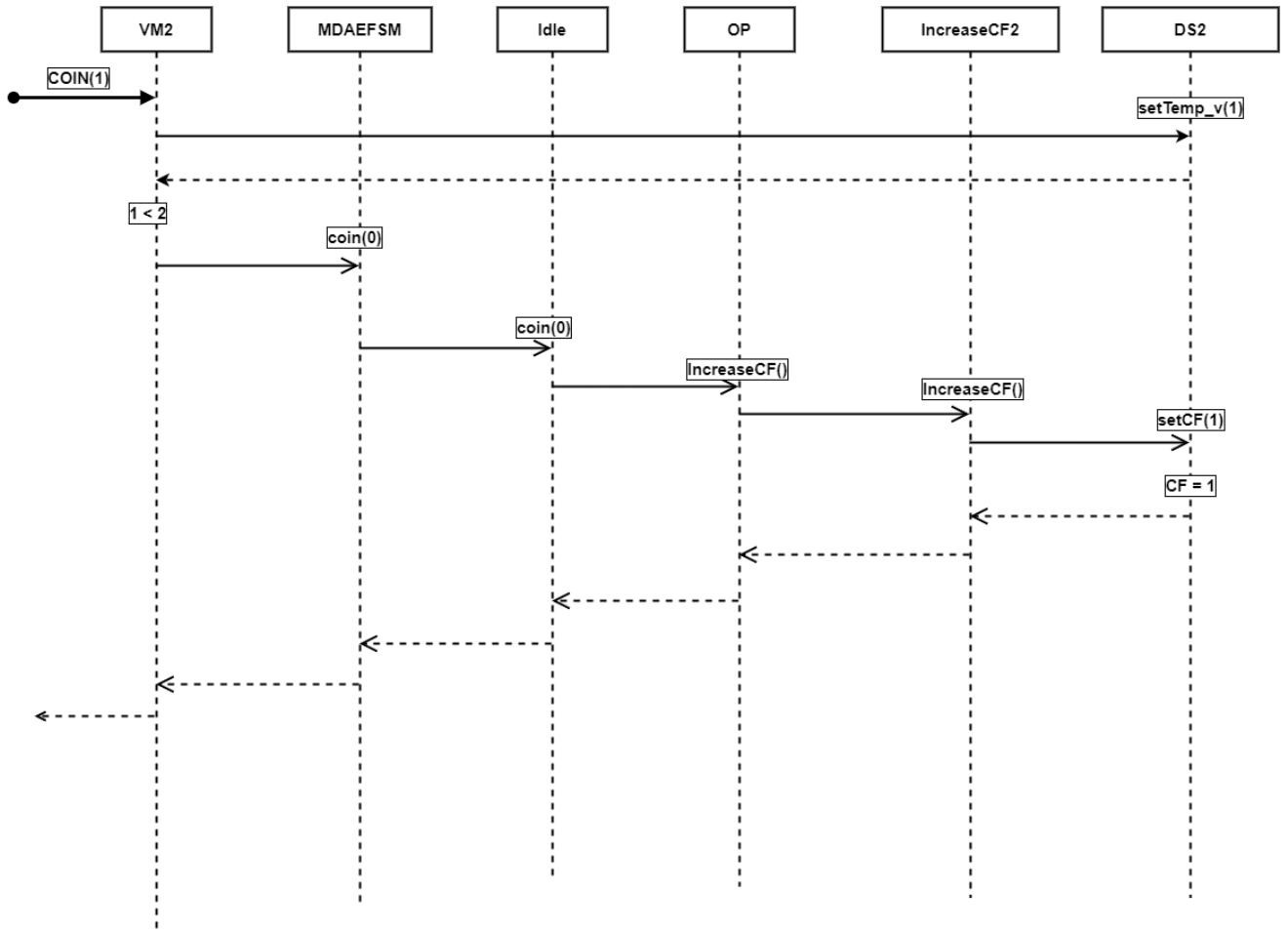
CREATE(2):



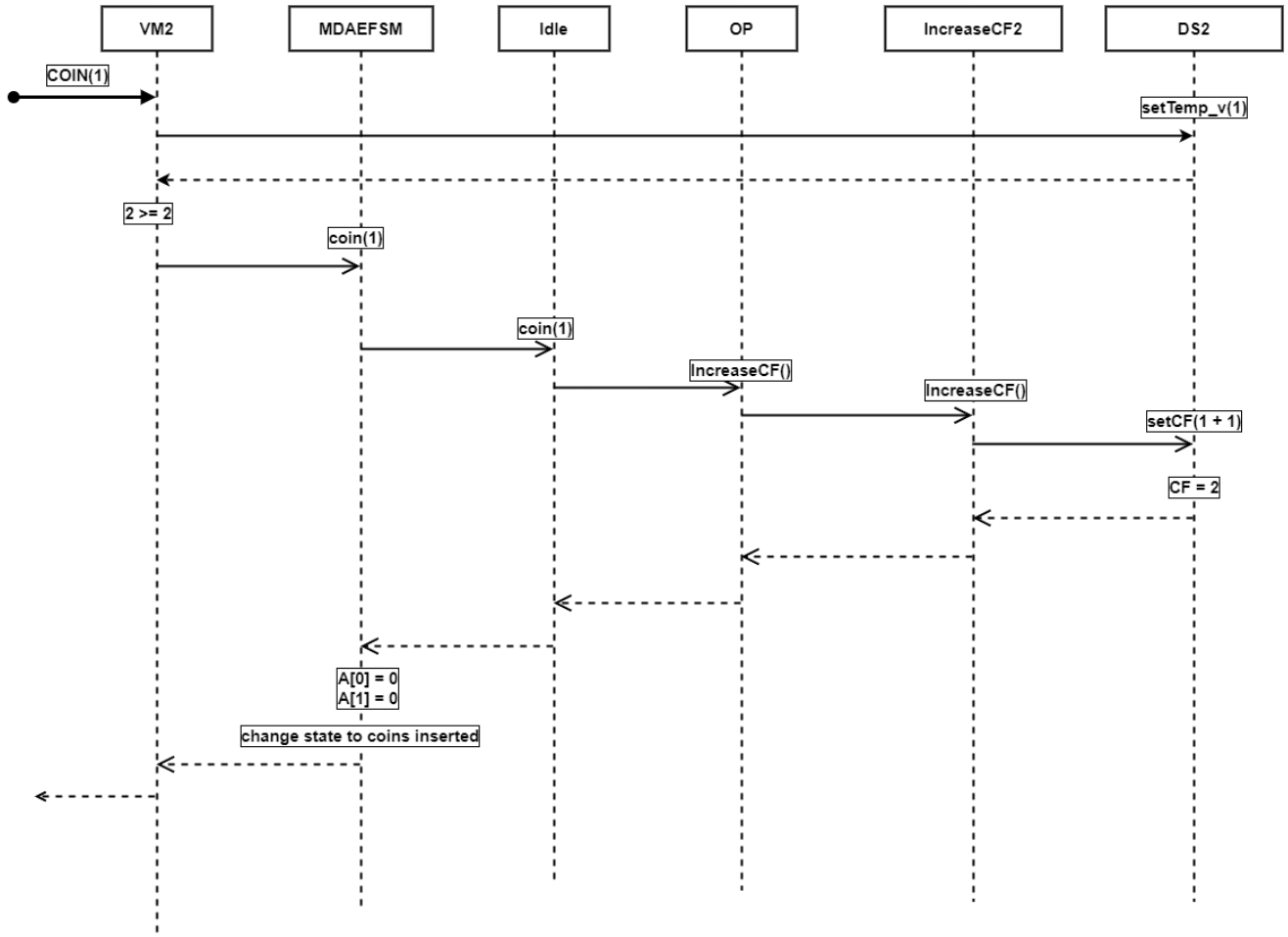
InsertCups(1):



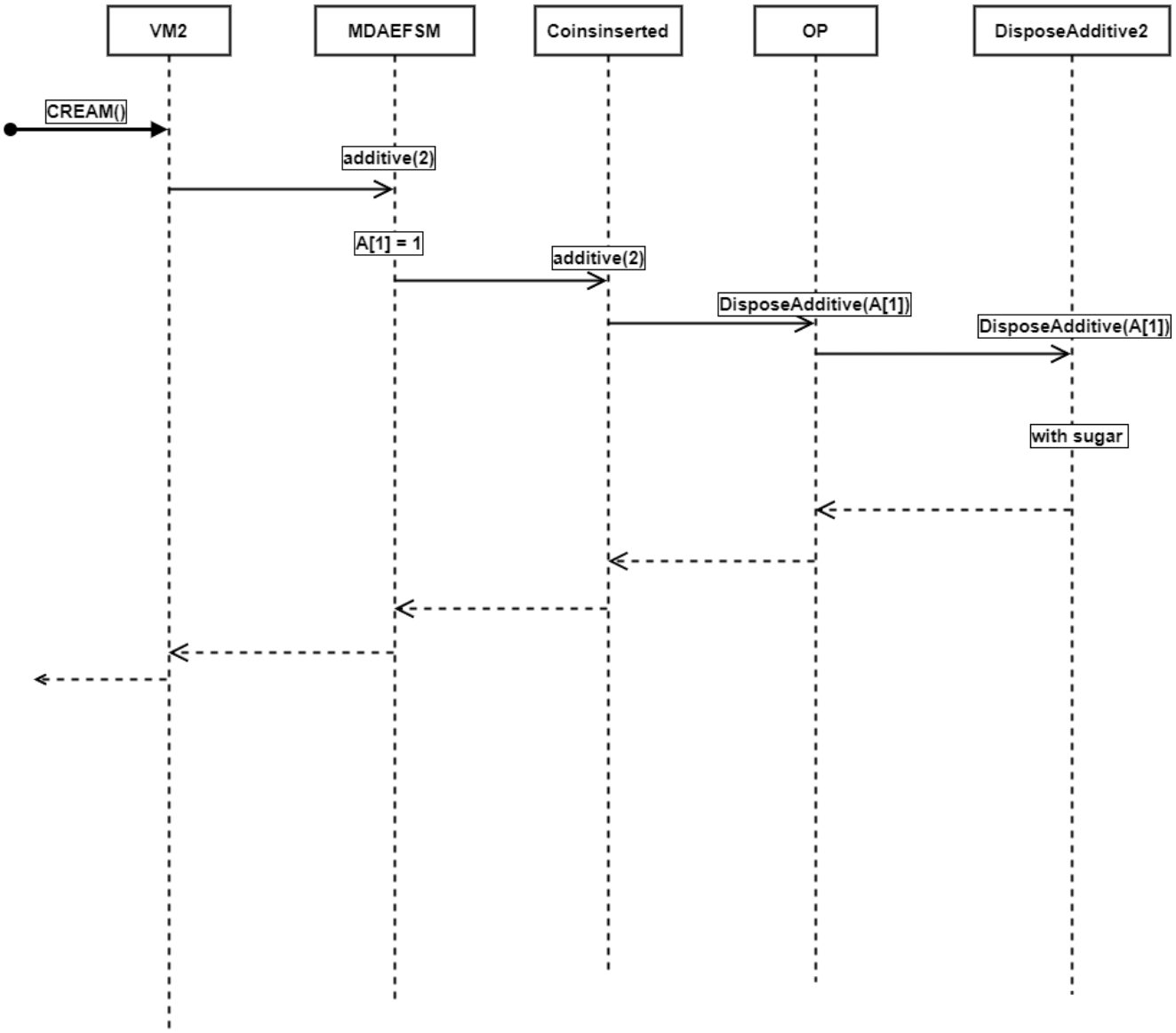
COIN(1):


VM2
MDAEFSM
Idle
OP
IncreaseCF2
DS2
COIN(1)
setTemp_v(1)
1 < 2
coin(0)
coin(0)
IncreaseCF()
IncreaseCF()
setCF(1)
CF = 1

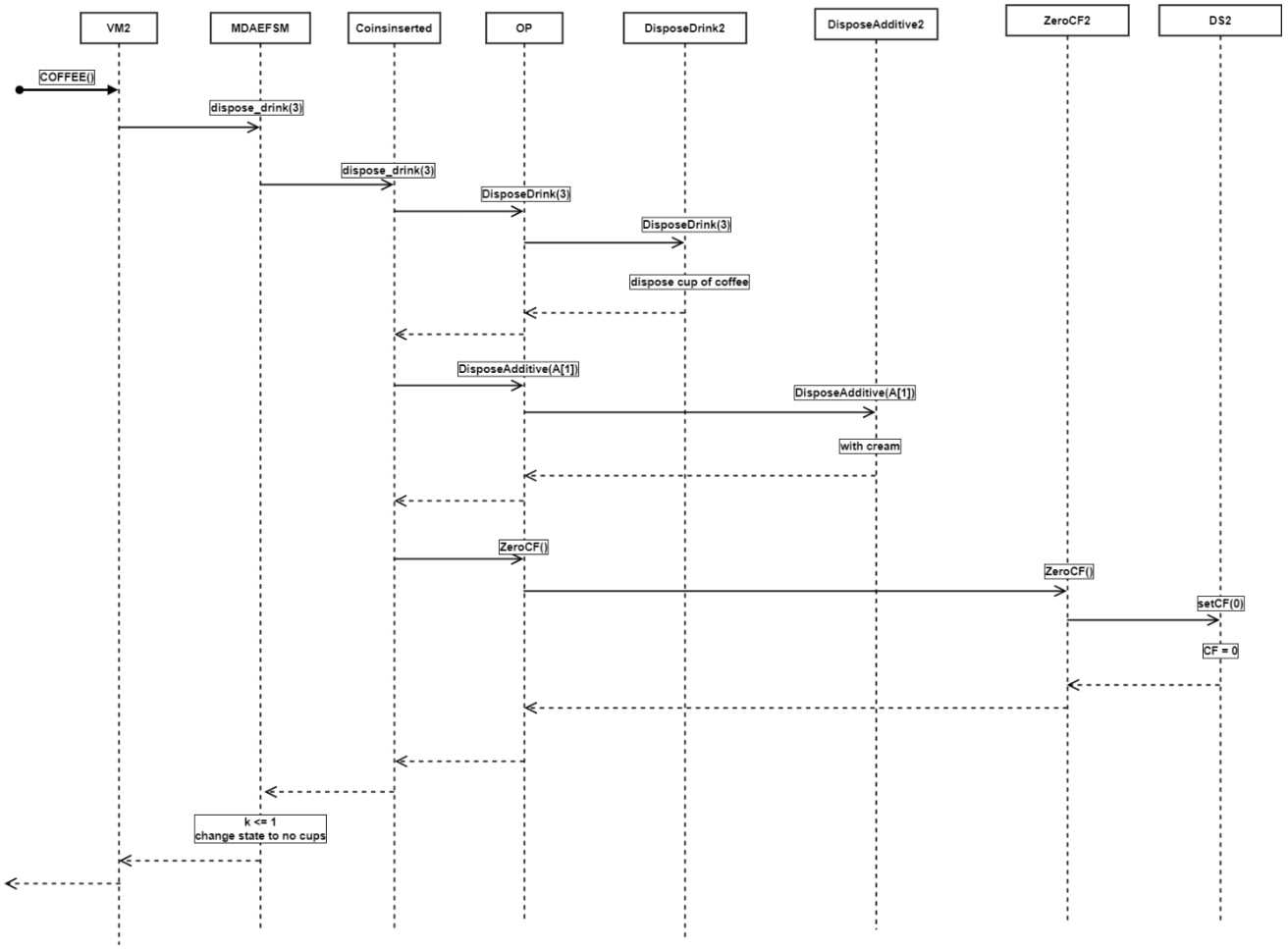
COIN(1):


VM2
MDAEFSM
Idle
OP
IncreaseCF2
DS2
COIN(1)
setTemp_v(1)
2 >= 2
coin(1)
coin(1)
IncreaseCF()
IncreaseCF()
setCF(1 + 1)
CF = 2
A[0] = 0
A[1] = 0
change state to coins inserted

CREAM():


VM2
MDAEFSM
Coinsinserted
OP
DisposeAdditive2
CREAM()
additive(2)
A[1] = 1
additive(2)
DisposeAdditive(A[1])
DisposeAdditive(A[1])
with sugar

COFFEE():


VM2
MDAEFSM
Coinsinserted
OP
DisposeDrink2
DisposeAdditive2
ZeroCF2
DS2
COFFEE()
dispose_drink(3)
dispose_drink(3)
DisposeDrink(3)
DisposeDrink(3)
dispose cup of coffee
DisposeAdditive(A[1])
DisposeAdditive(A[1])
with cream
ZeroCF()
ZeroCF()
setCF(0)
CF = 0
k <= 1
change state to no cups