CS 586: Software Systems Architecture

Final Project Report

Lisa He

04/27/2023

Table of Contents

MDA-EFSM Model for VM Components

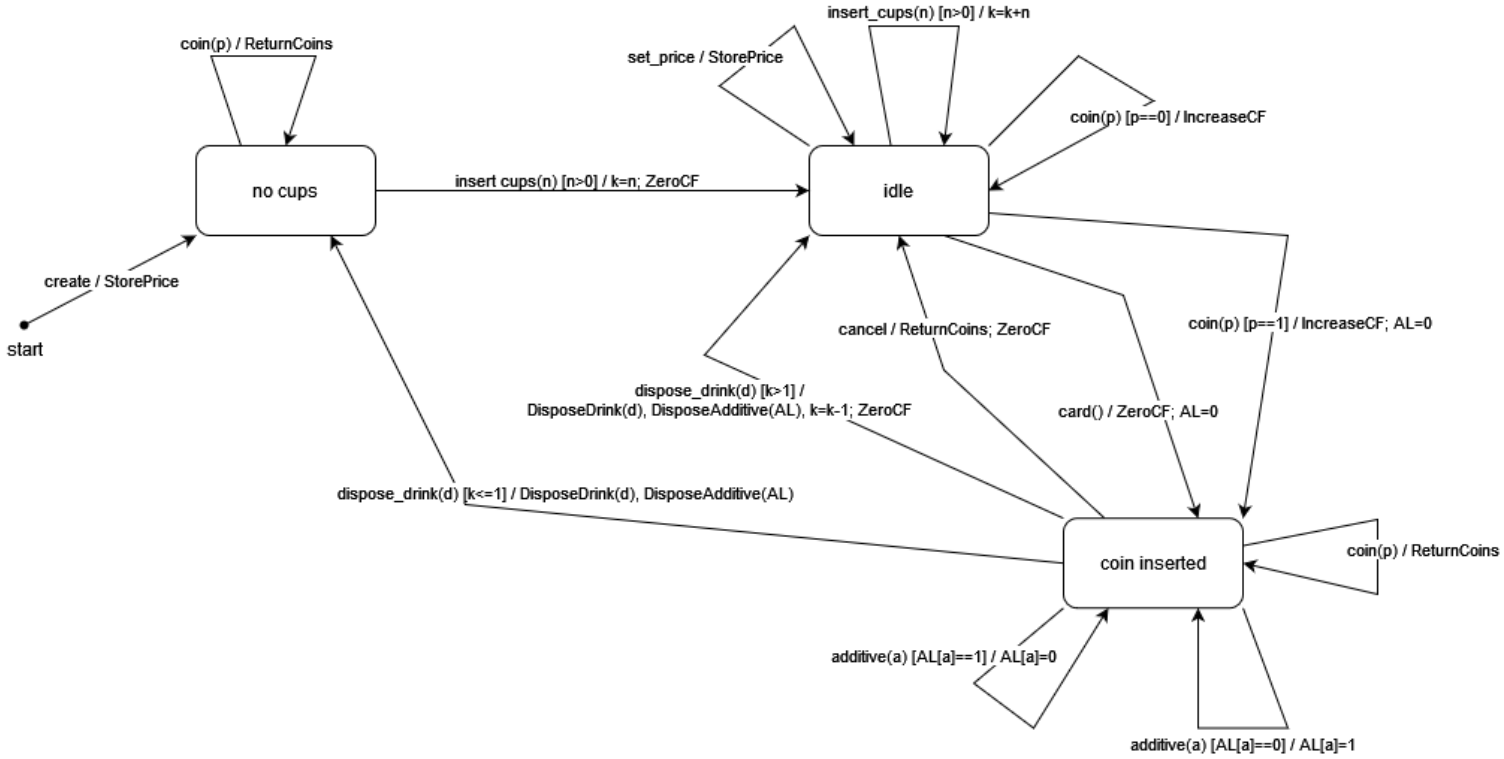
Meta events

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

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

State diagram



Pseudo-code for Input Processors (VM_1, VM_2)

class VM_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(0);
}

tea() {
        m->dispose_drink(0);
}

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

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

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

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

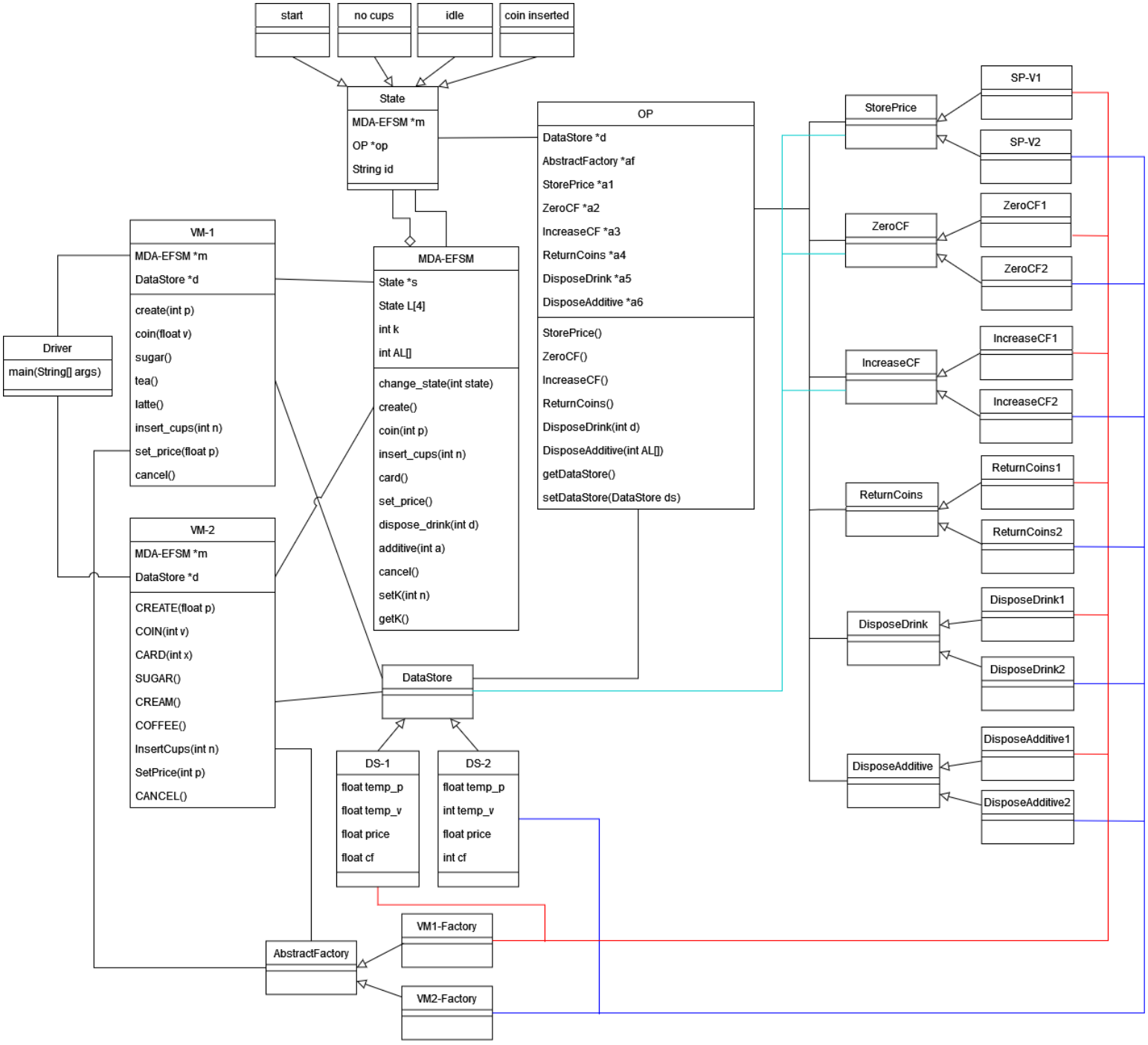
class VM_2

```
CREATE(float p) {
        d->temp_p=p;
```

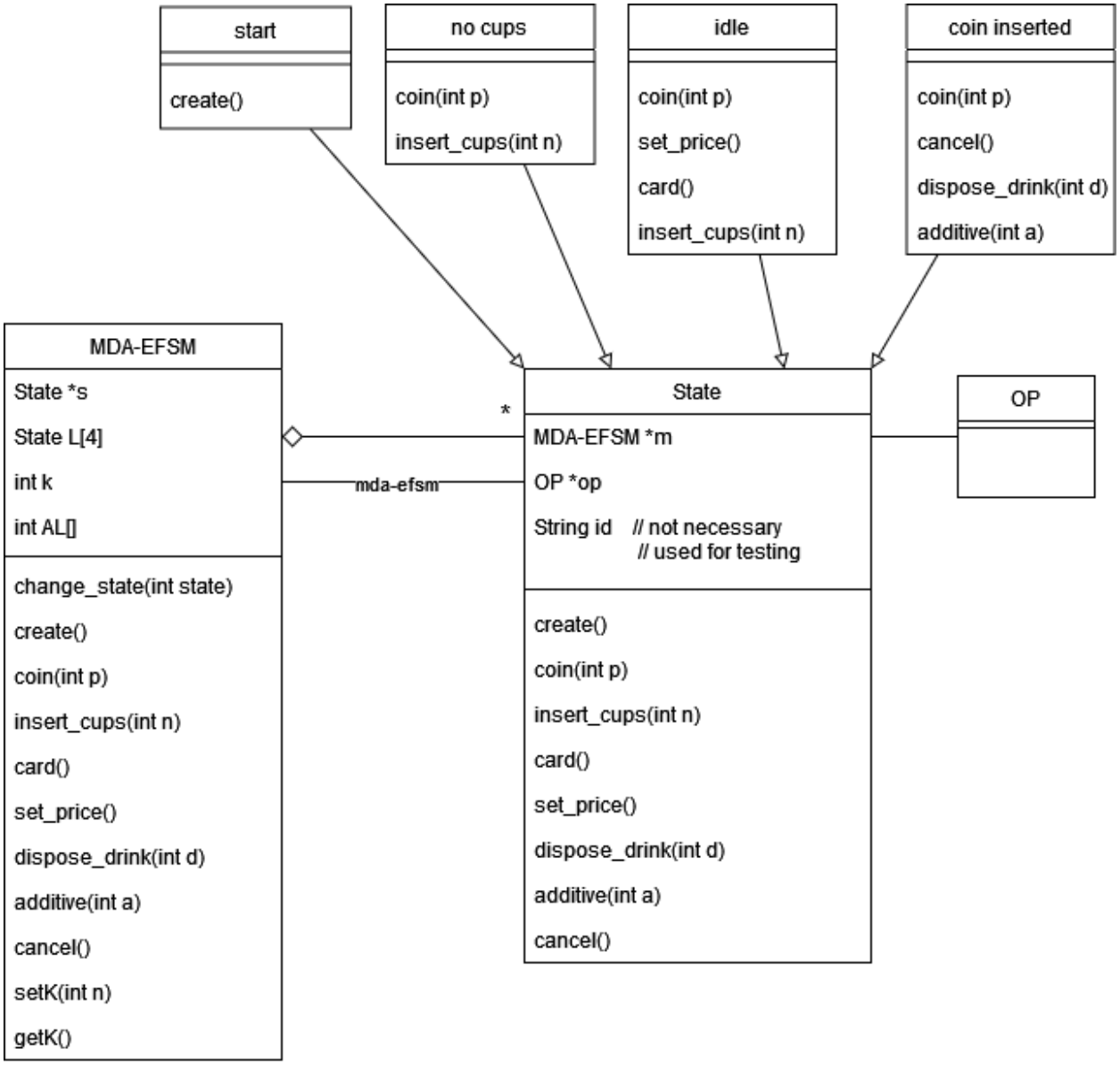
```
        m->create();
}

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(0);
}

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

COFFEE() {
        m->dispose_drink(0);
}

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

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

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

Class Diagrams of Complete MDA-EFSM

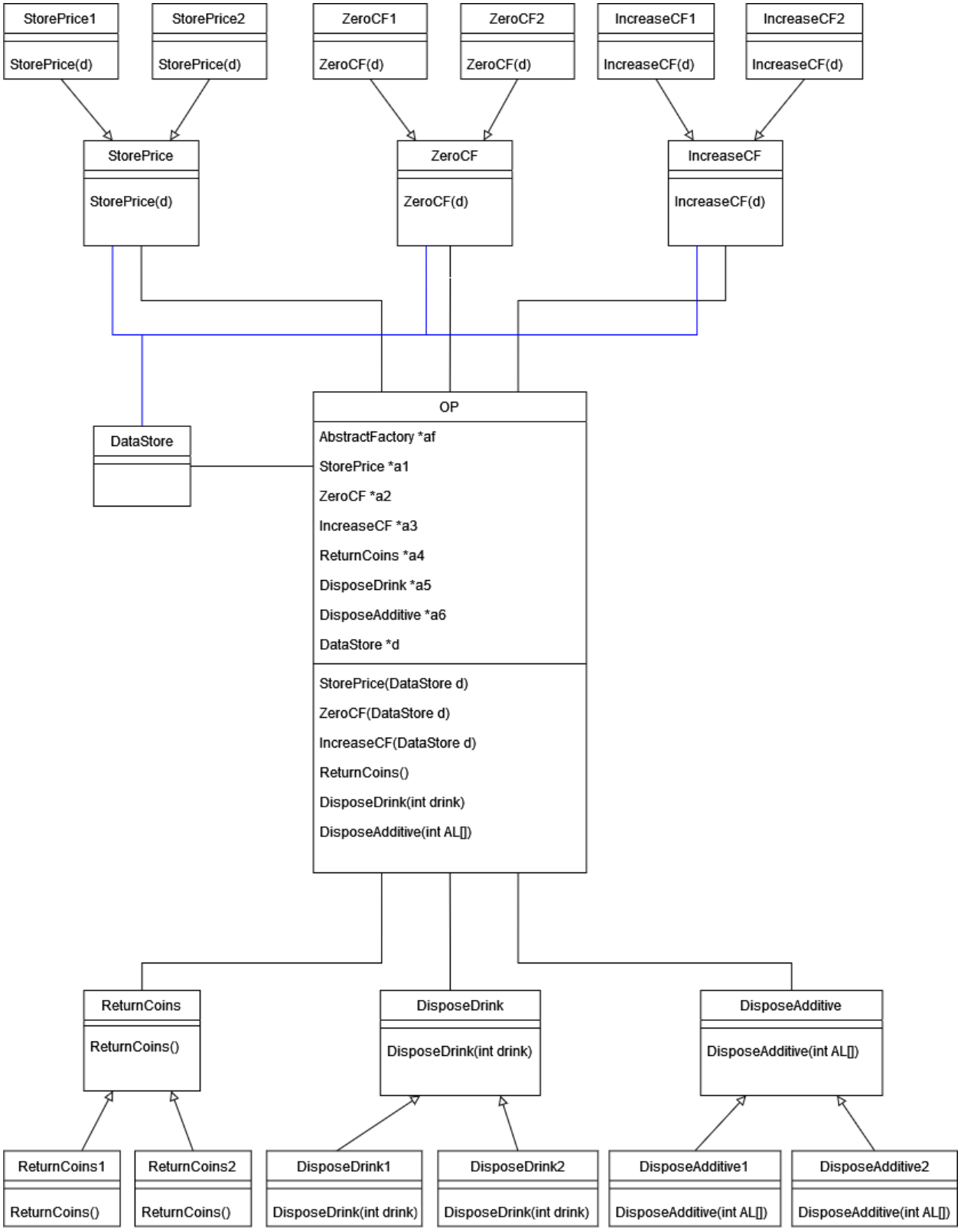
Overview



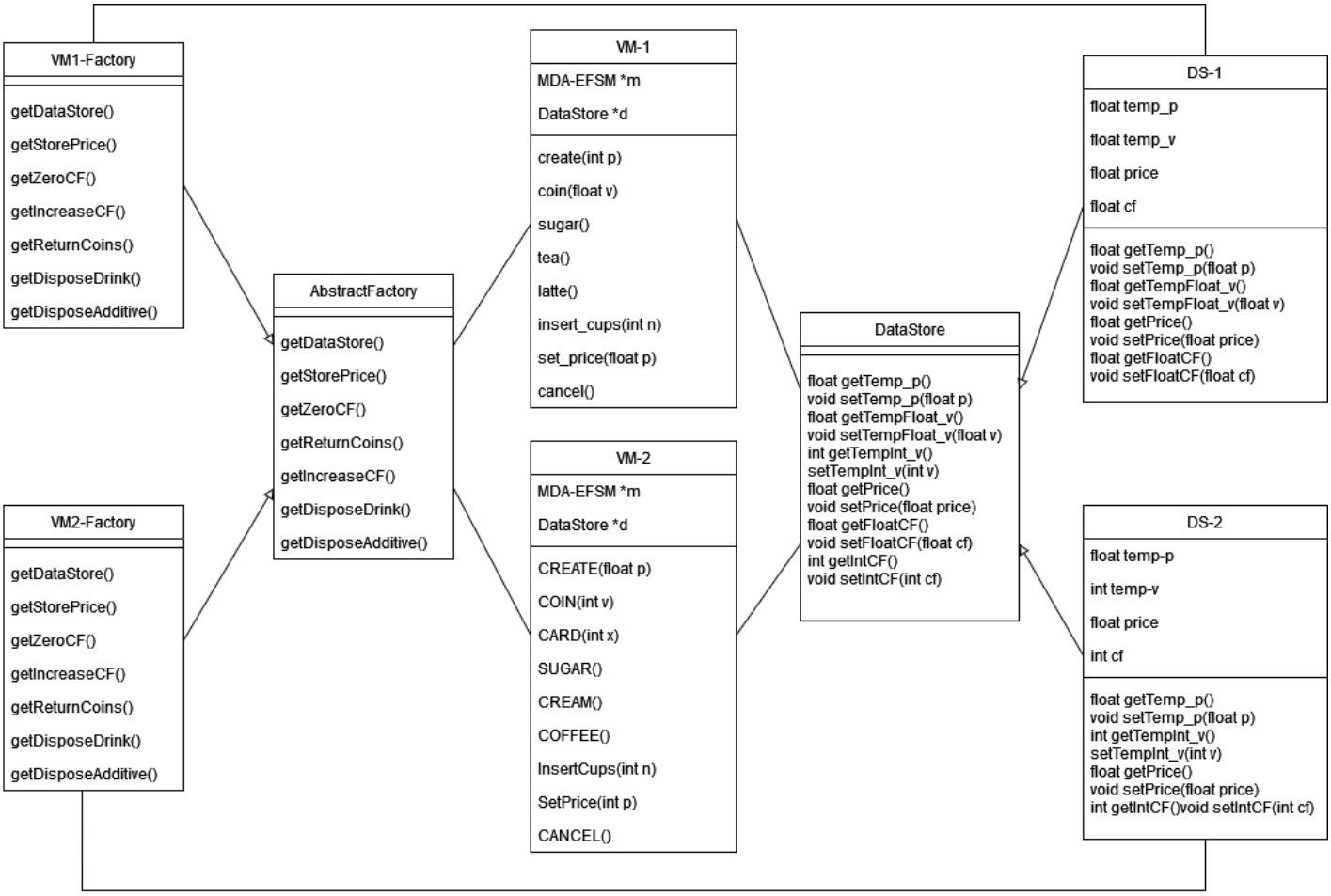
State Pattern



Strategy Pattern



Abstract Factory Pattern



Note: See overview diagram on page 7 for connection between meta action classes and VM1-Factory/VM2-Factory classes

Responsibilities

DRIVER

The Driver class is used to allow the user to access VM_1 and VM_2 classes.

Operations

main(String[] args)	Used to get user input to select operations in VM_1 and VM_2

INPUT PROCESSOR

VM_1

The VM_1 class supports the operations of Vending Machine 1 as defined by the project requirements.

Attributes

MDA_EFSM *m	Pointer to MDA_EFSM object
DataStore *d	Pointer to DataStore object

Operations

create(int p)	Creates a VM_1 object with an initial price p
coin(float v)	Indicates inserting v coins
sugar()	Used to select the sugar additive
tea()	Used to select tea
latte()	Used to select latte
insert_cups(int n)	Insert n cups
set_price(float p)	Set the price of a drink to p
cancel()	Cancel selection

VM_2

The VM_2 class supports the operations of Vending Machine 2 as defined by the project requirements.

Attributes

MDA_EFSM *m	Pointer to MDA_EFSM object
DataStore *d	Pointer to DataStore object

Operations

CREATE(float p)	Creates a VM_2 object with an initial price p
COIN(int v)	Indicates inserting v coins
CARD(int x)	Used to pay x amount with a card
SUGAR()	Used to select the sugar additive
CREAM()	Used to select the cream additive
COFFEE()	Used to select coffee
InsertCups(int n)	Insert n cups
SetPrice(int p)	Set the price of drink to p
CANCEL()	Cancel selection

MDA-EFSM

MDA_EFSM

The MDA_EFSM class supports meta events from VM_1 and VM_2, and functions as the context class in the state pattern.

Attributes

State *s	Pointer to the current state
State L[4]	List of states where L[1]=start, L[2]=no cups, L[3]=idle, L[4]=coin inserted
int k	Tracker for number of cups
int AL[]	List of additives

Operations

change_state(int state)	Used to change states: 0=Start, 1=NoCups, 2=Idle, 3=CoinsInserted
create()	Calls create() on the current state
coin(int p)	Calls coin(int p) on the current state
insert_cups(int n)	Calls insert_cups(int n) on the current state
card()	Calls card() on the current state
set_price()	Calls set_price() on the current state
dispose_drink(int d)	Calls dispose_drink(int d) on the current state
additive(int a)	Calls additive(int a) on the current state
cancel()	Calls cancel() on the current state
setK(int n)	Update int k
getK()	Return int k

OUTPUT PROCESSOR

OP

The OP class supports meta actions called by the MDA_EFSM class.

Attributes

AbstractFactory *af	Pointer to the AbstractFactory class
DataStore *d	Pointer to the DataStore class

// The remaining attributes are pointers to each respective meta action class.

StorePrice *a1

ZeroCF *a2

IncreaseCF *a3

ReturnCoins *a4

DisposeDrink *a5

DisposeAdditive *a6

Operations

StorePrice()	Invoke StorePrice(d) on the AbstractFactory class
ZeroCF()	Invoke ZeroCF(d) on the AbstractFactory class
IncreaseCF()	Invoke IncreaseCF(d) on the AbstractFactory class
ReturnCoins()	Invoke ReturnCoins() on the AbstractFactory class

DisposeDrink(int drink)	Invoke DisposeDrink(drink) on the AbstractFactory class
DisposeAdditive(int AL[])	Invoke DisposeAdditive(a) on the AbstractFactory class

STATE PATTERN: DECENTRALIZED

The State class is an abstract class that has blank methods which will be overwritten in the specific state classes. The state classes are responsible for change of states.

Attributes

MDA_EFSM *m	Pointer to the MDA-EFSM class m
OP *op	Pointer to the output processor class op
String id	Used for outputting the current state during execution

Operations

The following are void methods that are supported by the parent class but do nothing until overwritten in the specific state classes:

- create(), coin(int p), insert_cups(int n), card(), set_price(), dispose_drink(int d), additive(int a), cancel()

Start

This is the class for the Start state.

Attributes

String id = "Start State"	Used for identifying current state

Operations

create()	Invokes StorePrice() on the OP class, changes state to NoCups, and resets number of cups to zero

NoCups

This is the class for the NoCups state.

Attributes

String id = "NoCups State"	Used for identifying current state

Operations

coin(int p)	Returns coins by invoking ReturnCoins() on OP
insert_cups(int n)	Checks for cups >0 and stores number of cups to k, then invokes ZeroCF() on OP and changes to Idle state; otherwise print error

Idle

This is the class for the Idle state.

Attributes

String id = "Idle State"	Used for identifying current state

Operations

coin(int p)	Invokes IncreaseCF() on OP to store coins then checks if there are

	sufficient coins (p==0 means insufficient, p==1 means sufficient). If p ==1, create a new additive list AL and change to CoinInserted state
set_price()	Invoke StorePrice() on OP
card()	Create a new additive list AL and change to CoinInserted state
insert_cups(int n)	Check if n>0 and store cups to K; otherwise print error message

CoinsInserted

This is the class for the CoinsInserted state.

Attributes

String id = "CoinsInserted State"	Used for identifying current state

Operations

coin(int p)	Returns extra coins by invoking ReturnCoins() on OP
cancel()	Invokes ReturnCoins() and ZeroCF() on OP, change to Idle state
dispose_drink(int d)	Invokes DisposeDrink() and DisposeAdditive() on OP to dispose drinks, then checks for sufficient cups. ● if k > 1: k = k-1, call ZeroCF(), change to Idle state ● else if k <= 1: call ZeroCF, change to NoCups state
additive(int a)	Checks additive selection: ● if a == 0, set a = 1 (select the additive) ● else a = 0 (deselect the additive)

STRATEGY PATTERN

DisposeAdditive

DisposeAdditive is the interface for selecting an additive and disposing it.

Operations

DisposeAdditive(int AL[])	Abstract method to choose additive and store to the additive list AL

DisposeAdditive1

DisposeAdditive1 is the class for selecting an additive on VM-1

Operations

DisposeAdditive(int AL[])	Method to choose additive and store to the additive list AL on VM-1

DisposeAdditive2

DisposeAdditive2 is the class for selecting an additive on VM-2

Operations

DisposeAdditive(int AL[])	Method to choose additive and store to the additive list AL on VM-2

DisposeDrink

DisposeDrink is the interface for disposing of a drink.

Operations

DisposeDrink(int drink)	Abstract method to choose a drink

DisposeDrink1

DisposeDrink1 is the class for disposing of a drink on VM-1

Operations

DisposeDrink1(int drink)	Method to choose a drink

DisposeDrink2

DisposeDrink2 is the class for disposing of a drink on VM-2

Operations

DisposeDrink2(int drink)	Method to choose a drink

IncreaseCF

IncreaseCF is the interface for increasing the total available coins

Attributes

DataStore *d	Pointer to the DataStore class

Operations

IncreaseCF(DataStore d)	Abstract method to increase coins

IncreaseCF1

IncreaseCF1 is the class for increasing the total available coins on VM-1

Attributes

DataStore *d	Pointer to the DataStore class

Operations

IncreaseCF1(DataStore d)	Method to increase coins

IncreaseCF2

IncreaseCF2 is the class for increasing the total available coins on VM-2

Attributes

DataStore *d	Pointer to the DataStore class

Operations

IncreaseCF2(DataStore d)	Method to increase coins

ReturnCoins

ReturnCoins is the interface for returning coins

Operations

ReturnCoins()	Abstract method to return coins

ReturnCoins1

ReturnCoins1 is the class for returning coins on VM-1

Operations

ReturnCoins1()	Method to return coins

ReturnCoins2

ReturnCoins2 is the class for returning coins on VM-2

Operations

ReturnCoins2()	Method to return coins

StorePrice

StorePrice is the interface for storing the price of a drink.

Attributes

DataStore *d	Pointer to the DataStore class

Operations

StorePrice(DataStore d)	Abstract method to store price of a drink

StorePrice1

StorePrice1 is the class for storing the price of a drink on VM-1

Attributes

DataStore *d	Pointer to the DataStore class

Operations

StorePrice(DataStore d)	Method to store price of a drink on VM-1

StorePrice2

StorePrice2 is the class for storing the price of a drink on VM-2

Attributes

DataStore *d	Pointer to the DataStore class

Operations

StorePrice(DataStore d)	Method to store price of a drink on VM-2

ZeroCF

ZeroCF is the interface for resetting total coins to zero

Attributes

DataStore *d	Pointer to the DataStore class

Operations

ZeroCF(DataStore d)	Abstract method to reset coins

ZeroCF1

ZeroCF1 is the class for resetting total coins to zero on VM-1

Attributes

DataStore *d	Pointer to the DataStore class

Operations

ZeroCF(DataStore d)	Method to reset coins to zero

ZeroCF2

ZeroCF2 is the class for resetting total coins to zero on VM-2

Attributes

DataStore *d	Pointer to the DataStore class

Operations

ZeroCF(DataStore d)	Method to reset coins to zero

ABSTRACT FACTORY PATTERN

AbstractFactory

AbstractFactory is an interface for the Abstract Factory pattern.

Operations

The following are abstract methods that are supported by the parent class and are defined in the concrete factory classes:

- DataStore getDataStore(); StorePrice getStorePrice(); ZeroCF getZeroCF(); IncreaseCF getIncreaseCF(); ReturnCoins getReturnCoins(); DisposeDrink getDisposeDrink(); DisposeAdditive getDisposeAdditive();

VM1_Factory

VM1_Factory is the concrete factory class for VM-1 and implements the AbstractFactory interface.

Operations

DataStore getDataStore()	Returns a new DS_1 object
StorePrice getStorePrice()	Returns a new StorePrice1 object
ZeroCF getZeroCF()	Returns a new ZeroCF1 object
IncreaseCF getIncreaseCF()	Returns a new IncreaseCF1 object
ReturnCoins getReturnCoins()	Returns a new ReturnCoins1 object
DisposeDrink getDisposeDrink()	Returns a new DisposeDrink1 object
DisposeAdditive getDisposeAdditive()	Returns a new DisposeAdditive1 object

VM2_Factory

VM2_Factory is the concrete factory class for VM-2 and implements the AbstractFactory interface.

Operations

DataStore getDataStore()	Returns a new DS_2 object
StorePrice getStorePrice()	Returns a new StorePrice2 object
ZeroCF getZeroCF()	Returns a new ZeroCF2 object
IncreaseCF getIncreaseCF()	Returns a new IncreaseCF2 object
ReturnCoins getReturnCoins()	Returns a new ReturnCoins2 object
DisposeDrink getDisposeDrink()	Returns a new DisposeDrink2 object
DisposeAdditive getDisposeAdditive()	Returns a new DisposeAdditive2 object

DATA STORE

DataStore

DataStore is an abstract class that contains getters and setters for all possible attributes in DS1 and DS2.

Operations

The following are blank methods that will be overwritten in the specific datastore classes:

float getTemp_p()
void setTemp_p(float p)
float getTempFloat_v()
void setTempFloat_v(float v)
int getTempInt_v()
setTempInt_v(int v)
float getPrice()
void setPrice(float price)
float getFloatCF()
void setFloatCF(float cf)
int getIntCF()
void setIntCF(int cf)

DS_1

DS_1 is the DataStore class for VM-1.

Attributes

float temp_p, temp_v, price, cf; These are the data types supported by VM-1

Operations

The following are getters and setters for each attribute which will either return the current attribute value, or set the attribute to its respective parameter:

float getTemp_p()
void setTemp_p(float p)
float getTempFloat_v()
void setTempFloat_v(float v)
float getPrice()
void setPrice(float price)
float getFloatCF()
void setFloatCF(float cf)

DS_2

DS_2 is the DataStore class for VM-2.

Attributes

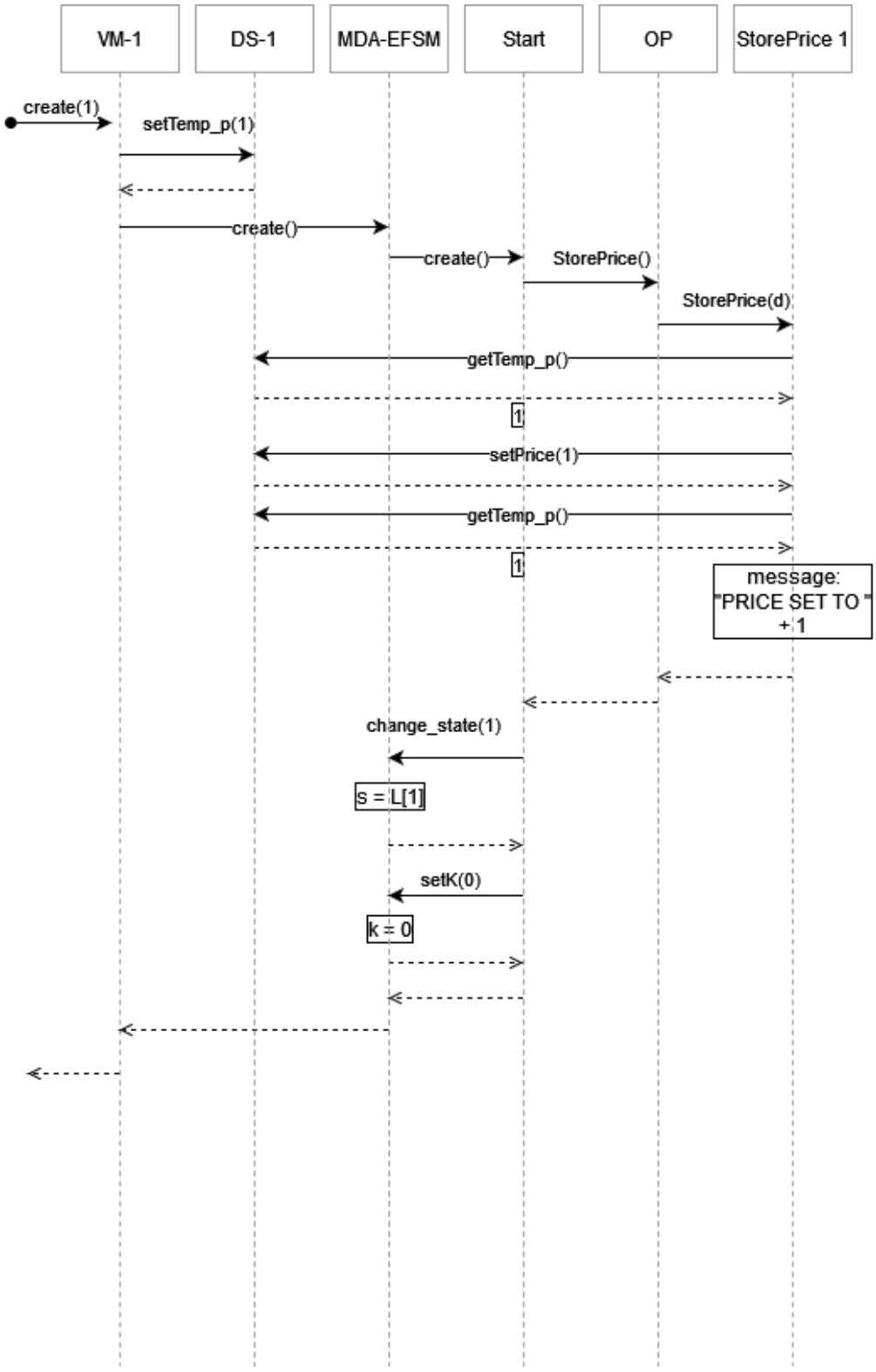
float temp_p, price;
int temp_v, cf; These are the data types supported by VM-2

Operations

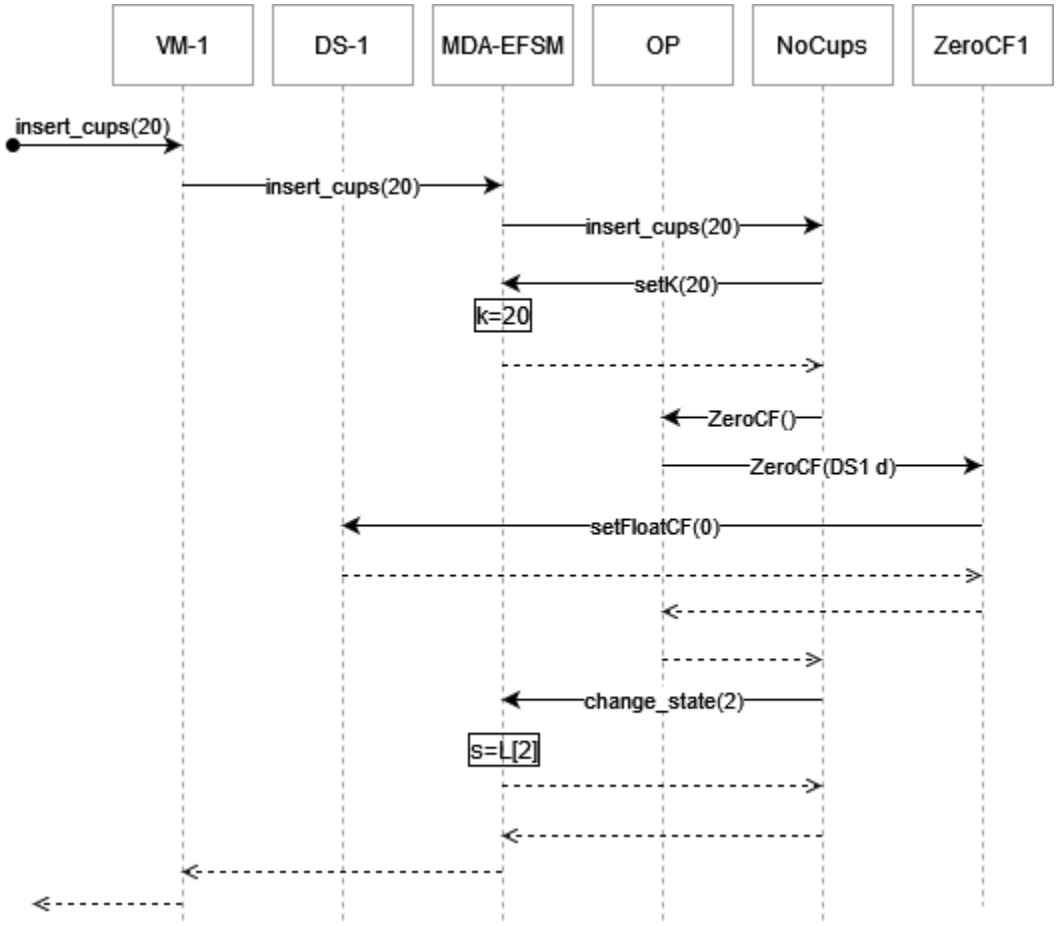
The following are getters and setters for each attribute which will either return the current attribute value, or set the attribute to its respective parameter:
float getTemp_p()
void setTemp_p(float p)
int getTempInt_v()
setTempInt_v(int v)
float getPrice()
void setPrice(float price)
int getIntCF()
void setIntCF(int cf)

Sequence Diagrams

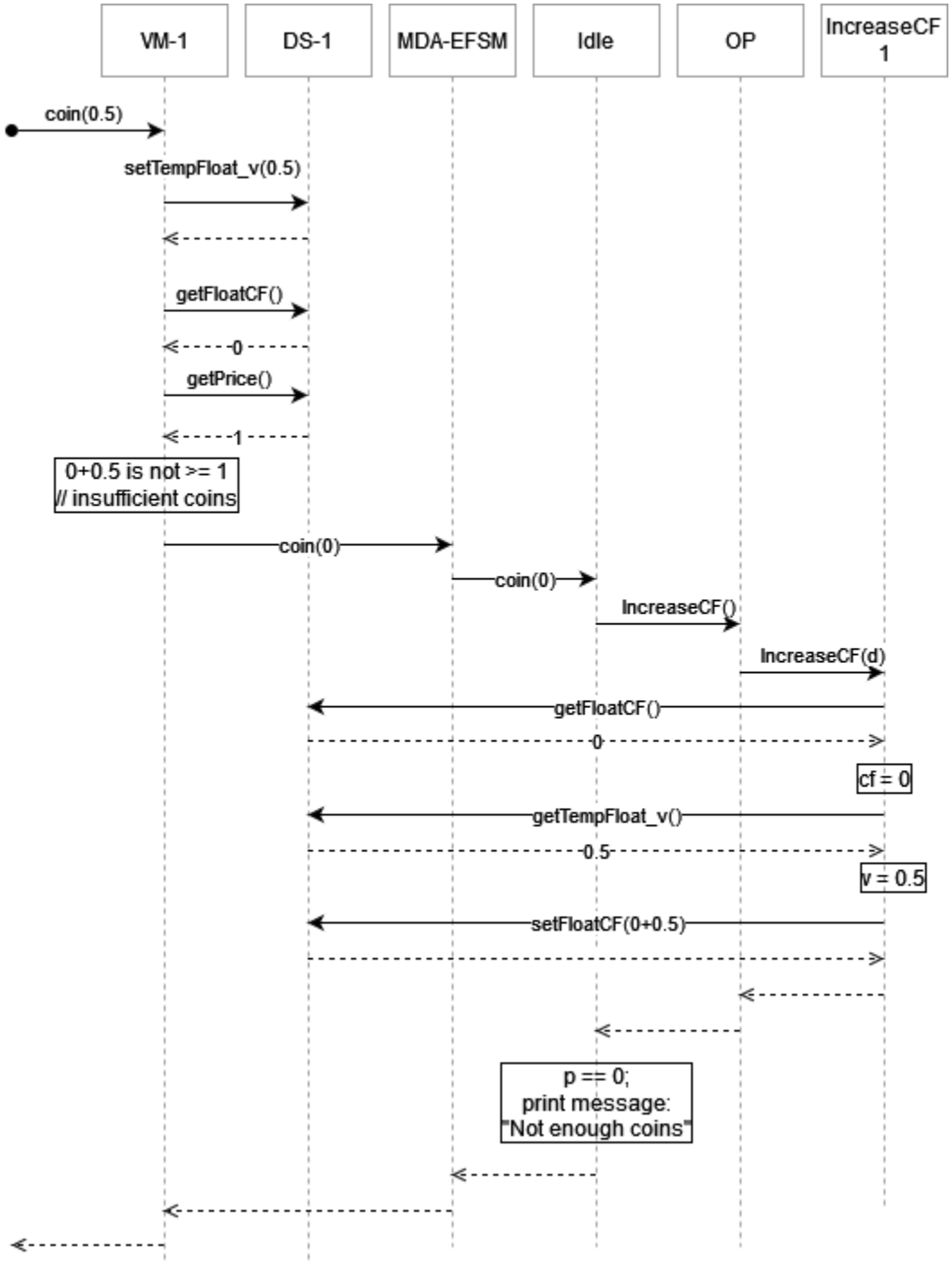
Scenario-I: VM_1 ***create(1), insert_cups(20), coin(0.5), coin(0.5), sugar(), latte()***
create(1)



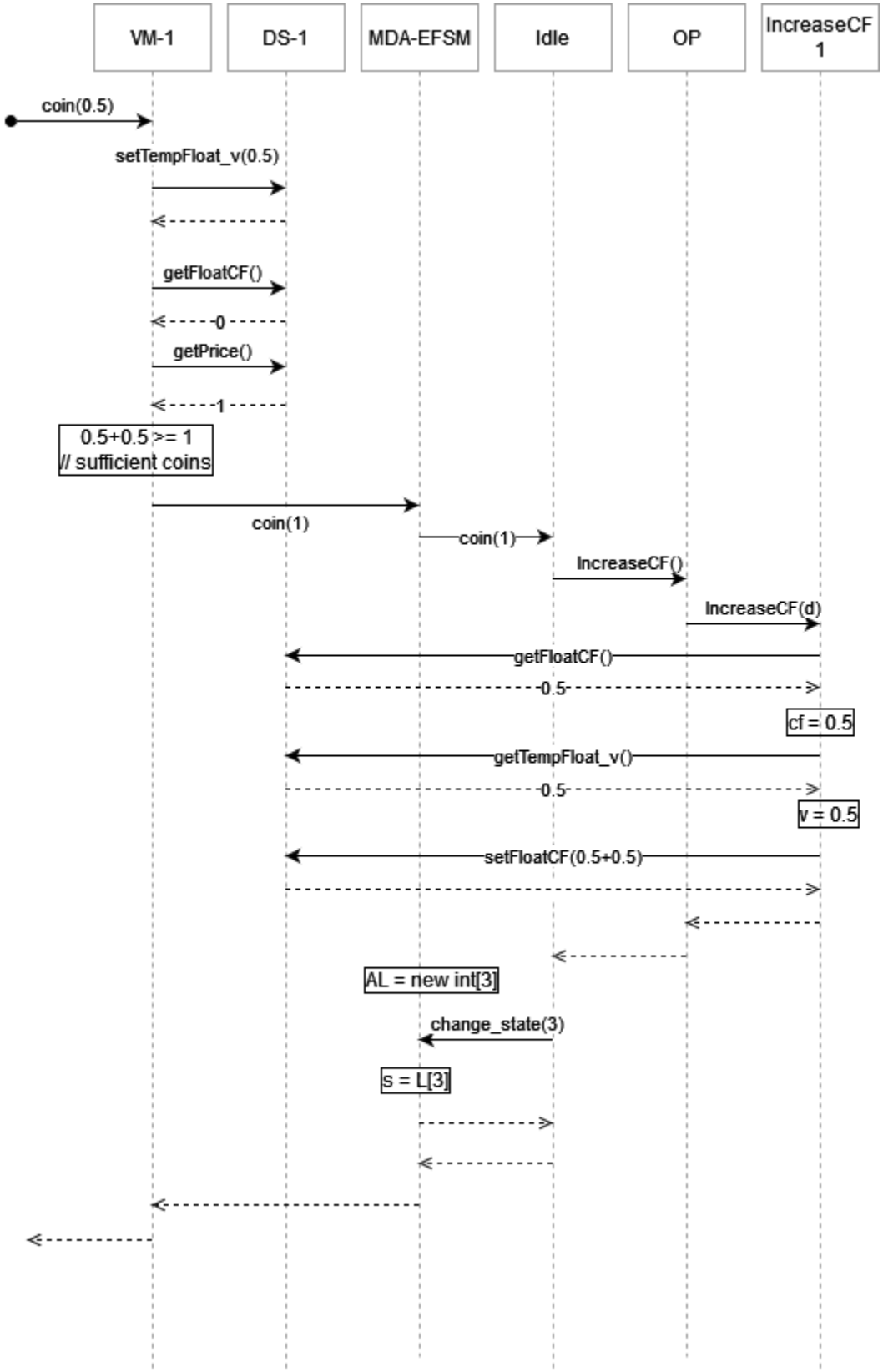
insert_cups(20)



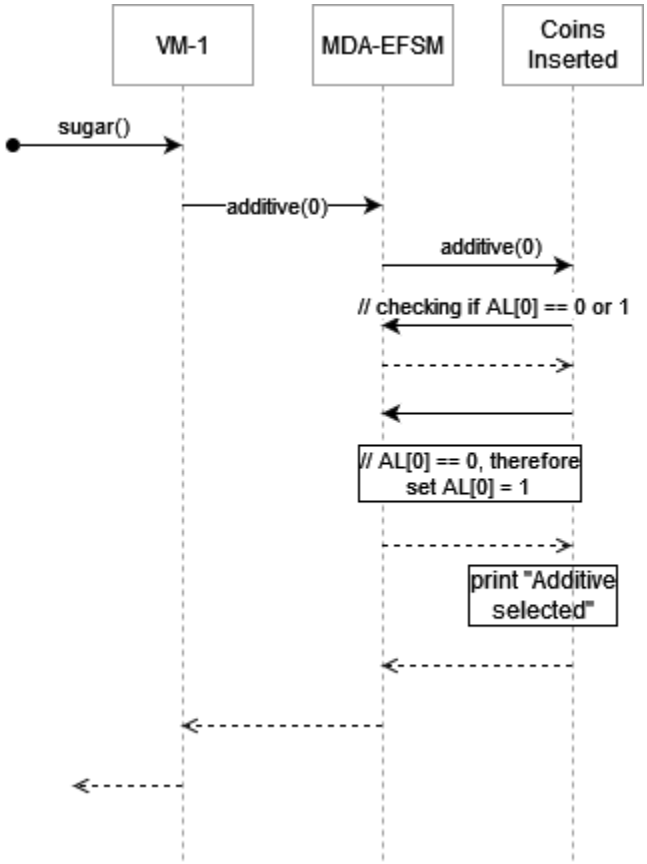
coin(0.5)


VM-1
DS-1
MDA-EFSM
Idle
OP
IncreaseCF 1
coin(0.5)
setTempFloat_v(0.5)
getFloatCF()
0
getPrice()
1
0+0.5 is not >= 1
// insufficient coins
coin(0)
coin(0)
IncreaseCF()
IncreaseCF(d)
getFloatCF()
0
cf = 0
getTempFloat_v()
0.5
v = 0.5
setFloatCF(0+0.5)
p == 0;
print message:
"Not enough coins"

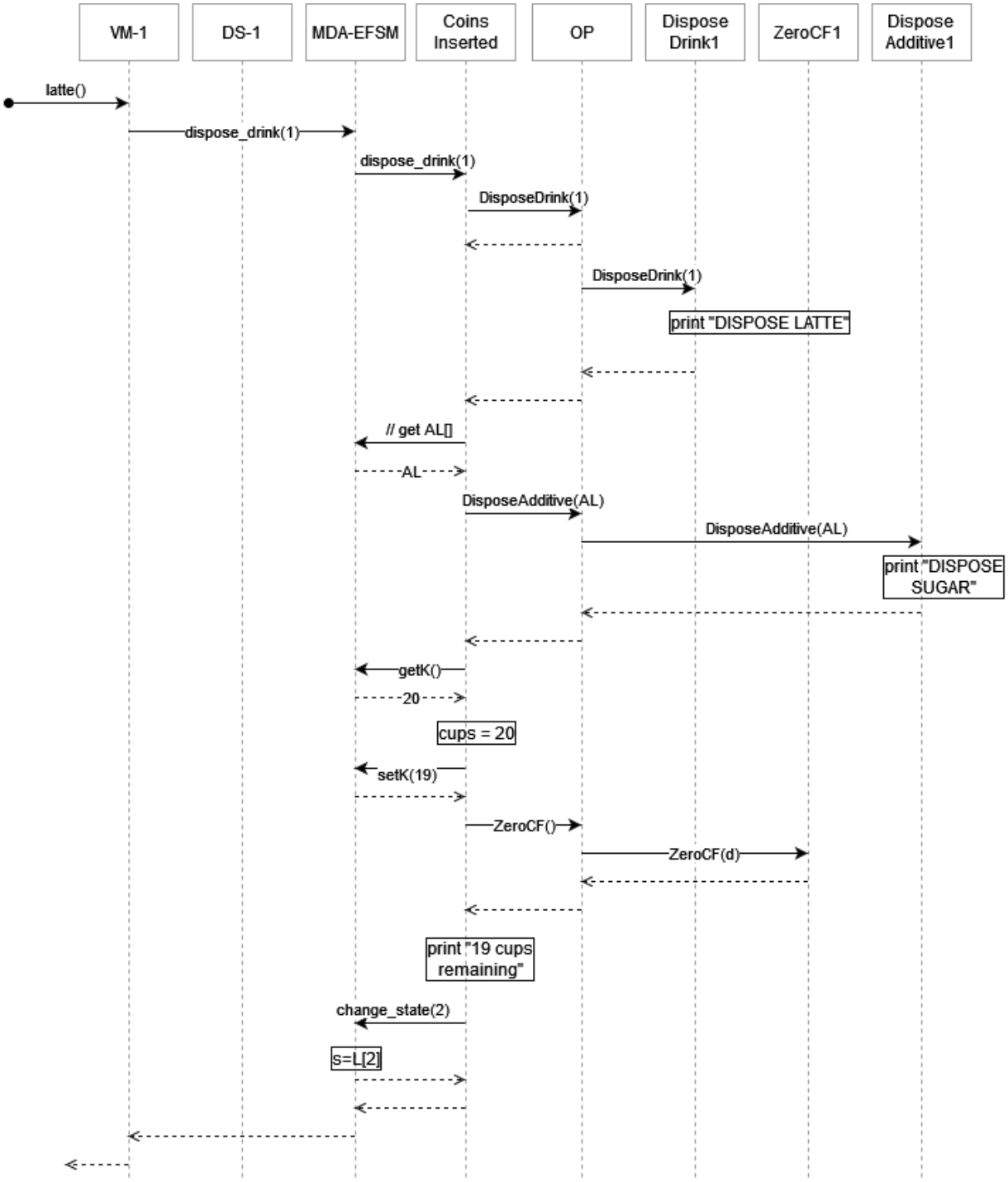
coin(0.5)


VM-1
DS-1
MDA-EFSM
Idle
OP
IncreaseCF 1
coin(0.5)
setTempFloat_v(0.5)
getFloatCF()
0
getPrice()
1
0.5+0.5 >= 1
// sufficient coins
coin(1)
coin(1)
IncreaseCF()
IncreaseCF(d)
getFloatCF()
0.5
cf = 0.5
getTempFloat_v()
0.5
v = 0.5
setFloatCF(0.5+0.5)
AL = new int[3]
change_state(3)
s = L[3]

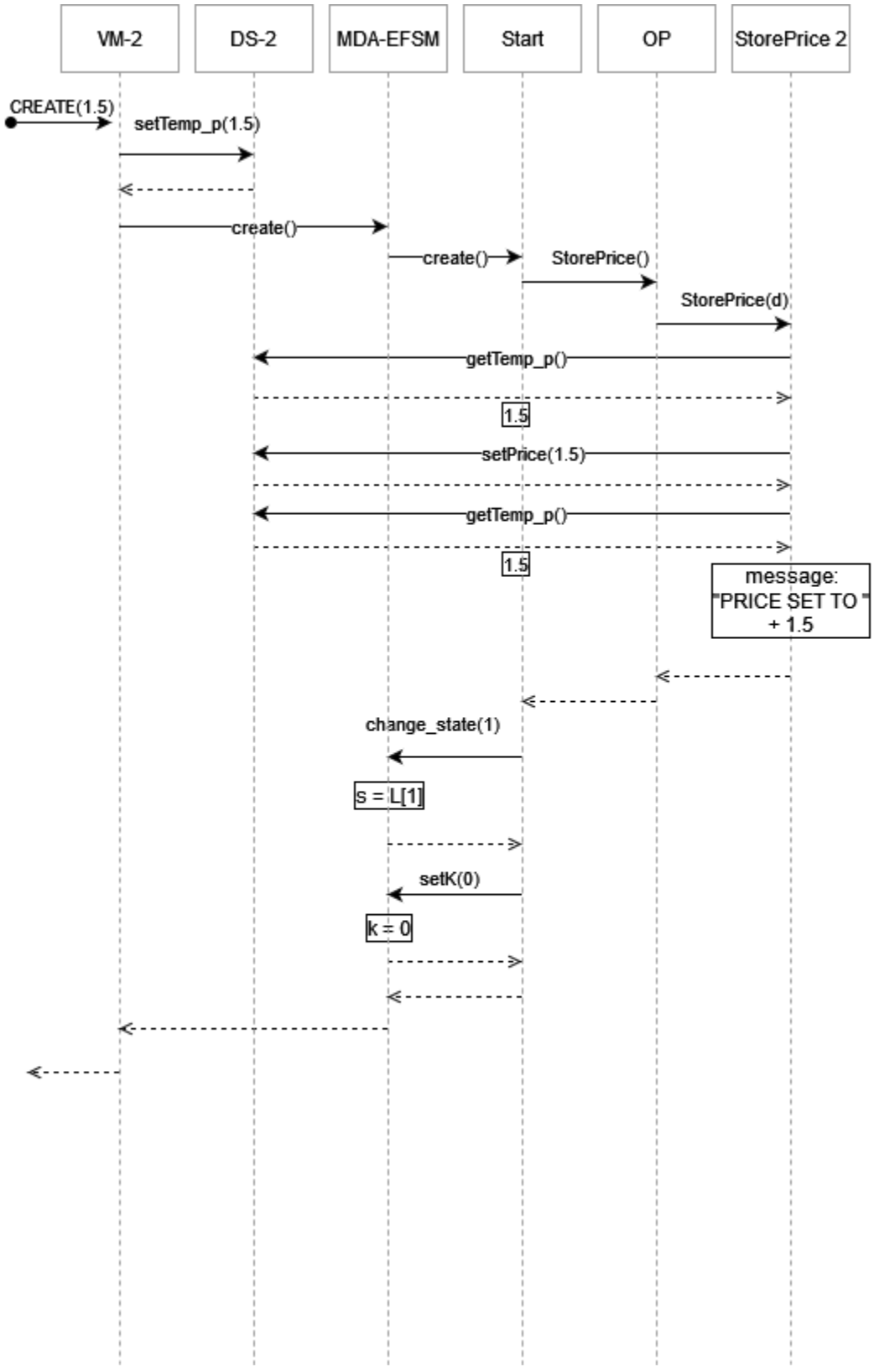
sugar()


VM-1
MDA-EFSM
Coins Inserted
sugar()
additive(0)
additive(0)
// checking if AL[0] == 0 or 1
// AL[0] == 0, therefore set AL[0] = 1
print "Additive selected"

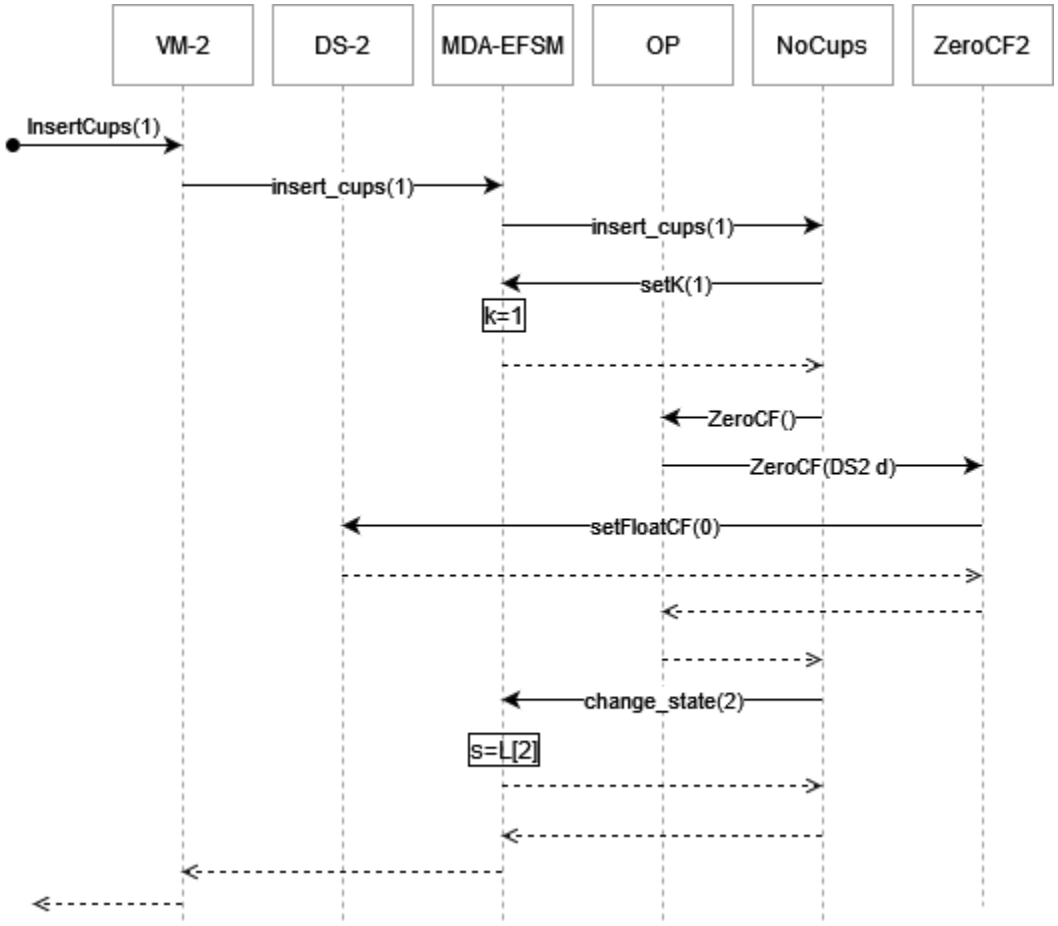
latte()


VM-1
DS-1
MDA-EFSM
Coins Inserted
OP
Dispose Drink1
ZeroCF1
Dispose Additive1
latte()
dispose_drink(1)
dispose_drink(1)
DisposeDrink(1)
DisposeDrink(1)
print "DISPOSE LATTE"
// get AL[]
AL
DisposeAdditive(AL)
DisposeAdditive(AL)
print "DISPOSE SUGAR"
getK()
20
cups = 20
setK(19)
ZeroCF()
ZeroCF(d)
print "19 cups remaining"
change_state(2)
s=L[2]

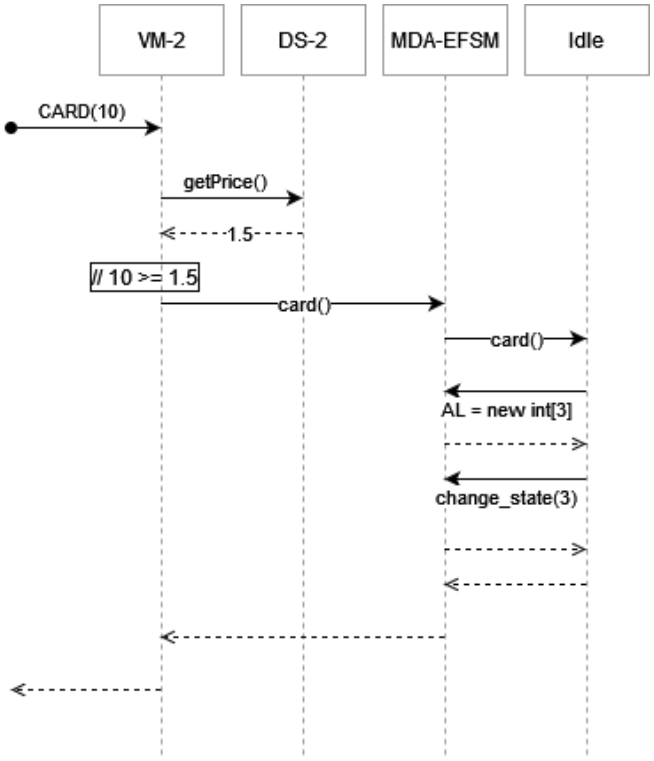
Scenario-II: VM_2 ***CREATE(1.5), InsertCups(1), CARD(10), CREAM(), COFFEE()***
CREATE(1.5)



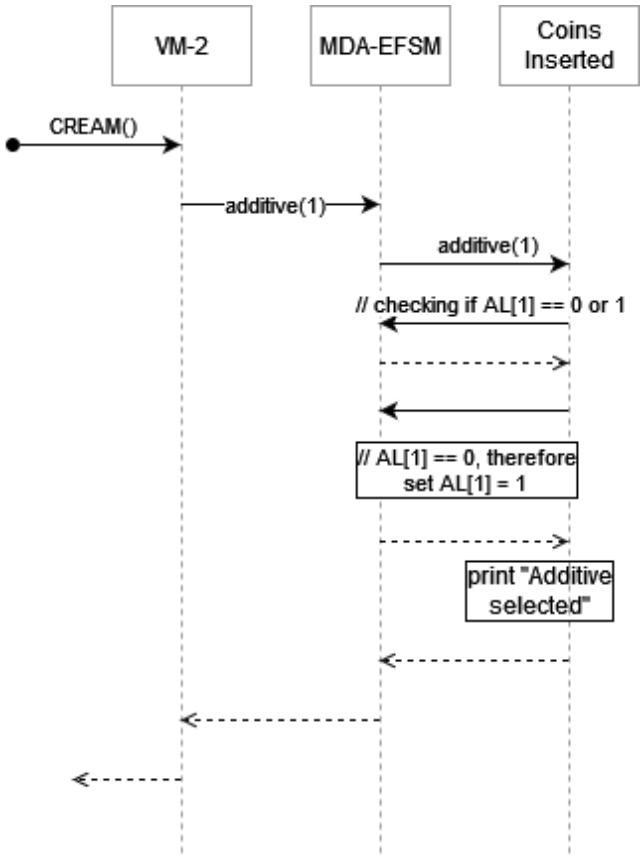
InsertCups(1)



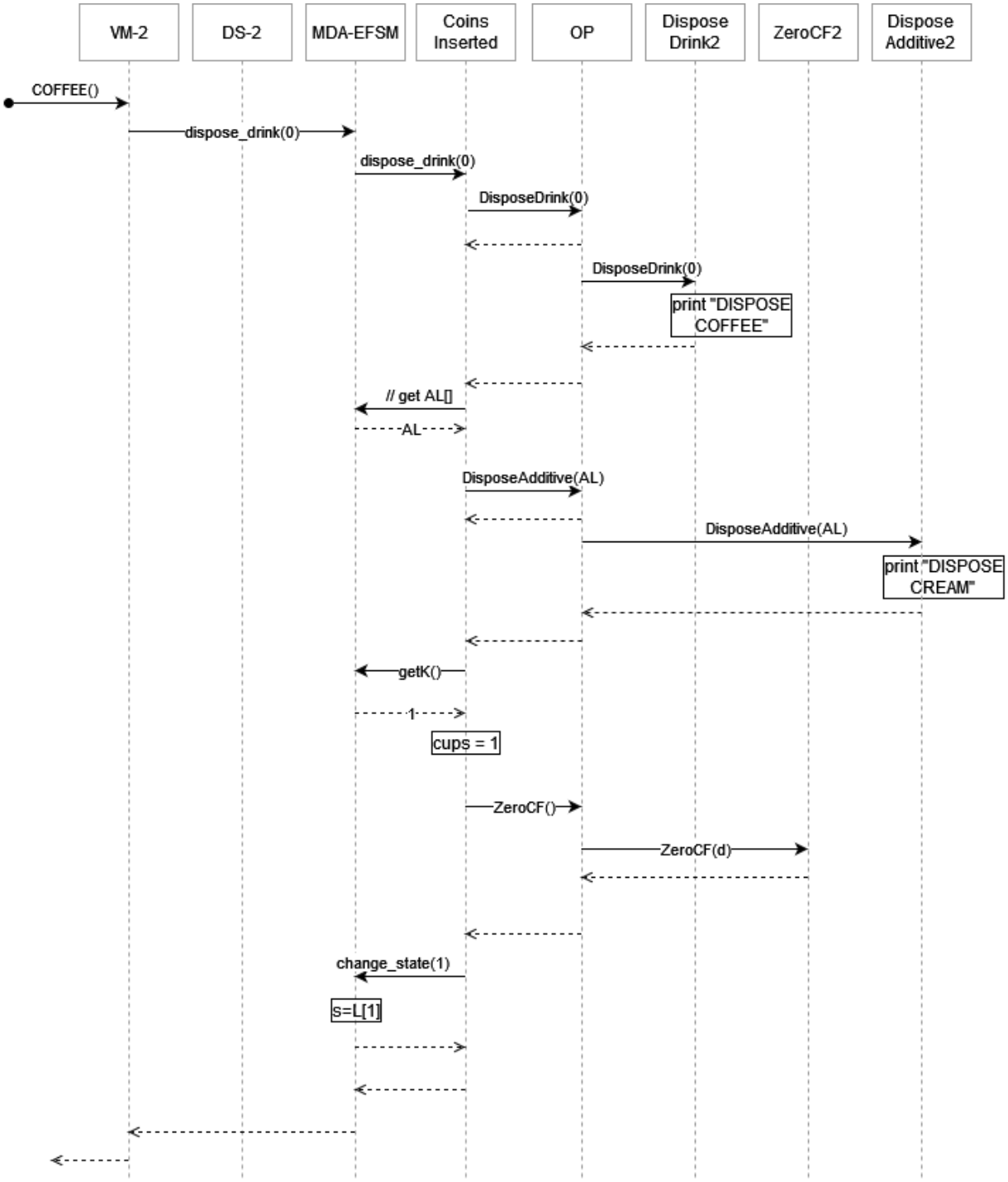
CARD(10)



CREAM()



COFFEE()


VM-2
DS-2
MDA-EFSM
Coins Inserted
OP
Dispose Drink2
ZeroCF2
Dispose Additive2
COFFEE()
dispose_drink(0)
dispose_drink(0)
DisposeDrink(0)
DisposeDrink(0)
print "DISPOSE COFFEE"
// get AL[]
AL
DisposeAdditive(AL)
DisposeAdditive(AL)
print "DISPOSE CREAM"
getK()
1
cups = 1
ZeroCF()
ZeroCF(d)
change_state(1)
s=L[1]