

MÉTODOS FORMAIS EM ENGENHARIA DE SOFTWARE

FORMAL MODELING OF A PARKING LOT MANAGEMENT SYSTEM IN VDM++

January 9, 2017

Ricardo Manuel Correia Magalhães

Mestrado Integrado em Engenharia Informática e de Computação

Faculdade de Engenharia da Universidade do Porto

Contents

1. Informal system description and list of requirements	2
1.1. Informal system description	2
1.2. List of requirements	2
2. Visual UML Model	3
2.1. Use Case Model	3
2.2. Class Model	3
3. Formal VDM++ Model	4
3.1. Card	4
3.2. Date	7
3.3. Group	8
3.4. MySystem	9
3.5. Operation	12
3.6. ParkingLot	14
4. Model Validation	17
4.1. MyTestCase	17
4.2. Test	18
5. Code Generation	21
6. Conclusions	22
7. References	23

1. INFORMAL SYSTEM DESCRIPTION AND LIST OF REQUIREMENTS

1.1. Informal system description

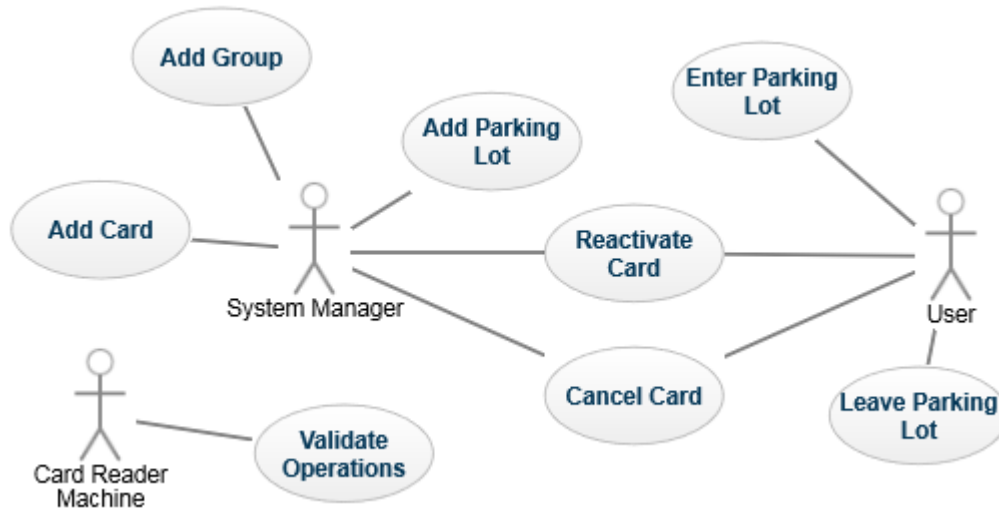
This system's goal is to maintain a set of parking lots. The users have a card with information (expiration date, plates, etc) and can enter or leave parking lots authorized for their group. The system is capable of creating every entity (card, parking, group) and doing operations, allowing or denying it. It can also list operations of a certain card and canceling or reactivating it.

1.2. List of requirements

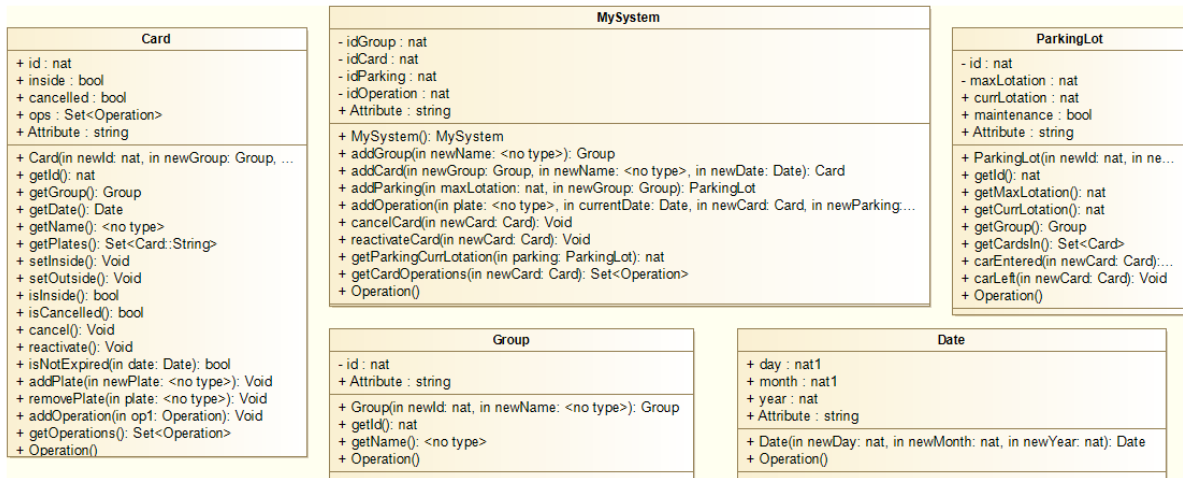
Id	Priority	Description
R1	Mandatory	Add groups to the system.
R2	Mandatory	Add a card, along with its owner's name and plates, with the belonging group.
R3	Mandatory	Add a parking lot to be used by a certain group.
R4	Mandatory	Add an operation (car entering or leaving).
R5	Mandatory	List a card's operations.
R6	Mandatory	Cancel and reactivate a card.
R6	Mandatory	Check information about current parking lot's location.

2. VISUAL UML MODEL

2.1. Use Case Model



2.2. Class Model



Class Name	Description
Card	Save card information and do operations related to it.
Group	Save group information and do operations related to it.
ParkingLot	Save parking lot information and do operations related to it.
Date	Auxiliar class to store date.
MySystem	Main class to do all operations

3. FORMAL VDM++ MODEL

3.1. Card

```
class Card

types
public String = seq of char;

instance variables
public id: nat;
public group: Group;
public username: String;
public plates: set of String;
public inside: bool;
public expirationDate: Date;
public cancelled: bool;
public ops: set of Operation;

operations

public Card: nat*Group*String*Date==> Card
  Card(newId,newGroup,newName,newDate) == (
    id := newId;
    group := newGroup;
    username := newName;
    plates := {};
    inside := false;
    expirationDate := newDate;
    cancelled := false;
    ops := {};
    return self;
  );

public getId: () ==> nat
  getId() == return self.id;

public getGroup: () ==> Group
  getGroup() == return self.group;
```

```
public getDate: () ==> Date
  getDate() == return self.expirationDate;

public getName: () ==> String
  getName() == return self.username;

public getPlates: () ==> set of String
  getPlates() == return self.plates;

public setInside: () ==> ()
  setInside() == inside := true;

public setOutside: () ==> ()
  setOutside() == inside := false;

public isInside: () ==> bool
  isInside() == return self.inside;

public isCancelled: () ==> bool
  isCancelled() == return self.cancelled;

public cancel: () ==> ()
  cancel() == cancelled := true;

public reactivate: () ==> ()
  reactivate() == cancelled := false;

public isNotExpired: (Date) ==> bool
  isNotExpired(date) == (
    if expirationDate.year > date.year or
```

```

    (expirationDate.year = date.year and expirationDate.month > date.month)
    or
    (expirationDate.year = date.year and expirationDate.month = date.month
     and expirationDate.day >= date.day)
  then return true
  else return false;
);

public addPlate: String ==> ()
  addPlate(newPlate) == (
    plates := plates union {newPlate}
  )
  pre newPlate not in set plates;

public removePlate: String ==> ()
  removePlate(plate) == (
    plates := plates \ {plate}
  )
  pre plate in set plates;

public addOperation: Operation ==> ()
  addOperation(op1) == (
    ops := ops union {op1}
  );

public getOperations: () ==> set of Operation
  getOperations() == return self.ops;

end Card

```

Function or operation	Line	Coverage	Calls
Card	17	100.0%	3
addOperation	86	100.0%	3
addPlate	74	100.0%	2

cancel	57	100.0%	1
getDate	36	100.0%	1
getGroup	33	100.0%	1
getId	30	100.0%	1
getName	39	100.0%	1
getOperations	91	100.0%	1
getPlates	42	100.0%	2
isCancelled	54	100.0%	3
isInside	51	100.0%	3
isNotExpired	63	56.0%	1
reactivate	60	100.0%	1
removePlate	80	100.0%	1
setInside	45	100.0%	1
setOutside	48	100.0%	1
Card.vdmpp		84.4%	27

3.2. Date

```

class Date

instance variables
public day: nat1;
public month: nat1;
public year: nat;

operations

public Date: nat*nat*nat ==> Date
  Date(newDay,newMonth,newYear) == (
    day := newDay;
    month := newMonth;
    year := newYear;
  )
  pre newDay < 32 and newMonth < 13;

end Date

```


Function or operation	Line	Coverage	Calls
Date	9	100.0%	5
Date.vdmpp		100.0%	5

3.3. Group

```

class Group

types
public String = seq of char;

instance variables
id: nat;
name: String;

operations

public Group: nat*String ==> Group
  Group(newId,newName) == (
    id := newId;
    name := newName;
    return self;
  );

public getId: () ==> nat
  getId() == return self.id;

public getName: () ==> String
  getName() == return self.name;

end Group

```

Function or operation	Line	Coverage	Calls
Group	12	100.0%	3

getId	19	100.0%	1
getName	22	100.0%	2
Group.vdmpp		100.0%	6

3.4. MySystem

```
class MySystem
types
public String = seq of char;
public Result = <Allow> | <Deny>;
public Type = <Enter> | <Leave>;

instance variables
idGroup: nat;
idCard: nat;
idParking: nat;
idOperation: nat;

groups: set of Group;
cards: set of Card;
parkings: set of ParkingLot;
ops: set of Operation;

operations

public MySystem: () ==> MySystem
MySystem() == (
  idGroup:=0;
  idCard:=0;
  idParking:=0;
  idOperation:=0;
  groups := {};
  cards := {};
  parkings := {};
  ops := {};
  return self;
);
```

```
public addGroup: String ==> Group
  addGroup(newName) == (
    dcl newGroup: Group := new Group(idGroup,newName);
    groups := groups union {newGroup};
    idGroup := idGroup + 1;
    return newGroup;
  );

public addCard: Group*String*Date==> Card
  addCard(newGroup,newName,newDate) == (
    dcl newCard: Card := new Card(idCard,newGroup,newName,newDate);
    cards := cards union {newCard};
    idCard := idCard+1;
    return newCard;
  )
  pre newGroup in set groups;

public addParking: nat*Group ==> ParkingLot
  addParking(maxLotation,newGroup) == (
    dcl newParking: ParkingLot := new ParkingLot(idParking,maxLotation,
      newGroup);
    parkings := parkings union {newParking};
    idParking := idParking + 1;
    return newParking;
  )
  pre newGroup in set groups;

public addOperation: String*Date*Card*ParkingLot*Type ==> Operation
  addOperation(plate,currentDate,newCard,newParking,newType) == (
```

```
    dcl newOp: Operation := new Operation(idOperation,currentDate,newCard,
        newParking,newType);
    ops := ops union {newOp};
    return newOp;
)
pre newCard in set cards and newParking in set parkings and plate in set
    newCard.plates;

public cancelCard: Card ==> ()
    cancelCard(newCard) == (
        newCard.cancel()
    )
    pre newCard in set cards
    post newCard.cancelled = true;

public reactivateCard: Card ==> ()
    reactivateCard(newCard) == (
        newCard.reactivate()
    )
    pre newCard in set cards
    post newCard.cancelled = false;

public getParkingCurrLotation: ParkingLot ==> nat
    getParkingCurrLotation(parking) == (
        return parking.getCurrLotation();
    )
    pre parking in set parkings;

public getCardOperations: Card ==> set of Operation
    getCardOperations(newCard) == (return newCard.getOperations())
```

```

pre newCard in set cards;

end MySystem

```

Function or operation	Line	Coverage	Calls
MySystem	19	100.0%	3
System	19	100.0%	3
addCard	40	100.0%	3
addGroup	32	100.0%	3
addOperation	58	100.0%	3
addParking	49	100.0%	2
cancelCard	66	100.0%	1
getCardOperations	86	100.0%	1
getParkingCurrLotation	80	100.0%	1
reactivateCard	73	100.0%	1
MySystem.vdmpp		100.0%	57

3.5. Operation

```

class Operation

types

public Result = <Allow> | <Deny>;
public Type = <Enter> | <Leave>;

instance variables

id: nat;
date: Date;
public cardUsed: Card;
parking: ParkingLot;
type: Type;
result: Result;

operations

```

```

public Operation: nat*Date*Card*ParkingLot*Type ==> Operation
Operation(newId,newDate,newCard,newParking,newType) == (
  id := newId;
  date := newDate;
  cardUsed := newCard;
  parking := newParking;
  type := newType;

  if type = <Enter>
  then self.enter();

  if type = <Leave>
  then self.leave();

  cardUsed.addOperation(self);
  return self;
);

private enter: () ==> ()
enter() == (
  result := <Deny>;
  if cardUsed not in set parking.getCardsIn() and not cardUsed.isInside()
  then (
    result := <Allow>;
    parking.carEntered(cardUsed);
    cardUsed.setInside();
  )
)
pre (cardUsed.expirationDate.year > date.year or --Check if it has not
  expired
  (cardUsed.expirationDate.year = date.year and cardUsed.expirationDate.
    month > date.month) or
  (cardUsed.expirationDate.year = date.year and cardUsed.expirationDate.
    month = date.month
    and cardUsed.expirationDate.day >= date.day))
and not cardUsed.cancelled --Check if card is cancelled
and not parking.maintenance
and not parking.currLotation = 0; --Check if parking is on maintenance;

```

```

private leave: () ==> ()
  leave() == (
    result := <Deny>;
    if cardUsed in set parking.getCardsIn() and cardUsed.isInside()
    then (
      result := <Allow>;
      parking.carLeft(cardUsed);
      cardUsed.setOutside();
    )
  ); --I assumed not to check same things as in "enter" because the car has
      to go out even
      -- even if it is cancelled or expired, I guess

public getResult: () ==> Result
  getResult() == return self.result;

end Operation

```

Function or operation	Line	Coverage	Calls
Operation	19	92.0%	3
enter	37	72.6%	2
getResult	67	100.0%	3
leave	55	100.0%	1
Operation.vdmpp		81.3%	9

3.6. ParkingLot

```

class ParkingLot
types

instance variables
id: nat;
maxLotation: nat;
public currLotation: nat;
group: Group;

```

```
cardsIn: set of Card;
public maintenance: bool;

operations

public ParkingLot: nat*nat*Group ==> ParkingLot
ParkingLot (newId,newMaxLotation,newGroup) == (
  id := newId;
  maxLotation := newMaxLotation;
  currLotation := maxLotation;
  group := newGroup;
  cardsIn := {};
  maintenance := false;
  return self;
);

public getId: () ==> nat
getId() == return self.id;

public getMaxLotation: () ==> nat
getMaxLotation() == return self.maxLotation;

public getCurrLotation: () ==> nat
getCurrLotation() == return self.currLotation;

public getGroup: () ==> Group
getGroup() == return self.group;

public getCardsIn: () ==> set of Card
getCardsIn() == return self.cardsIn;

public carEntered: Card ==> ()
carEntered(newCard) == (
  currLotation := currLotation-1;
  cardsIn := cardsIn union {newCard};
```



```

);

public carLeft: Card ==> ()
  carLeft(newCard) == (
    currLotation := currLotation+1;
    cardsIn := cardsIn \ {newCard};
  )
  post currLotation <= maxLotation;

end ParkingLot

```

Function or operation	Line	Coverage	Calls
ParkingLot	14	100.0%	2
carEntered	40	100.0%	1
carLeft	46	100.0%	1
getCardsIn	37	100.0%	3
getCurrLotation	31	100.0%	2
getGroup	34	100.0%	1
getId	25	100.0%	1
getMaxLotation	28	100.0%	1
ParkingLot.vdmpp		100.0%	12

4. MODEL VALIDATION

4.1.MyTestCase

```
class MyTestCase
/*
  Superclass for test classes, simpler but more practical than VDMUnit`
  TestCase.
  For proper use, you have to do: New -> Add VDM Library -> IO.
  FEUP, MFES, 2015/16.
*/

operations

-- Simulates assertion checking by reducing it to pre-condition checking.
-- If 'arg' does not hold, a pre-condition violation will be signaled.

protected assertTrue: bool ==> ()
assertTrue(arg) ==
  return
pre arg;

-- Simulates assertion checking by reducing it to post-condition checking.
-- If values are not equal, prints a message in the console and generates
-- a post-conditions violation.

protected assertEquals: ? * ? ==> ()
assertEquals(expected, actual) ==
  if expected <> actual then (
    IO`print("Actual value (");
    IO`print(actual);
    IO`print(") different from expected (");
    IO`print(expected);
    IO`println(")\n")
  )
post expected = actual

end MyTestCase
```

Function or operation	Line	Coverage	Calls
assertEqual	20	38.8%	22
assertTrue	12	0.0%	0
MyTestCase.vdmpp		35.0%	22

4.2. Test

```

class Test is subclass of MyTestCase

types
public Type = <Enter> | <Leave>;
public Result = <Allow> | <Deny>;

operations

public Test: () ==> Test
  Test() == (
    return self;
  );

--Test initializing and populating MySystem.

public testMySystem: () ==> ()
  testMySystem() == (
    dcl MySystem: MySystem := new MySystem();
    dcl group: Group := MySystem.addGroup("Students");
    dcl park: ParkingLot := MySystem.addParking(30,group);
    dcl date1: Date := new Date(31,12,2017);
    dcl card1: Card := MySystem.addCard(group,"John Doe",date1);

    assertEquals("Students",group.getName());
    assertEquals("Students",park.getGroup().getName());
    assertEquals(30,park.getMaxLotation());
    assertEquals("John Doe",card1.getName());
    assertEquals(group,card1.getGroup());
    assertEquals(date1,card1.getDate());

    assertEquals(0,group.getId());
  );

```

```
    assertEquals(0, park.getId());
    assertEquals(0, card1.getId());
};

public testCard: () ==> ()
testCard() == (
    dcl MySystem: MySystem := new MySystem();
    dcl group: Group := MySystem.addGroup("Students");
    dcl date1: Date := new Date(1,1,2015);
    dcl date2: Date := new Date(2,2,2017);
    dcl card1: Card := MySystem.addCard(group, "John Doe", date1);

    assertEquals(false, card1.isCancelled());
    assertEquals(false, card1.isInside());
    assertEquals(false, card1.isNotExpired(date2));

    MySystem.cancelCard(card1);
    assertEquals(true, card1.isCancelled());

    MySystem.reactivateCard(card1);
    assertEquals(false, card1.isCancelled());

    card1.addPlate("33-MM-22");
    assertEquals({"33-MM-22"}, card1.getPlates());
    card1.removePlate("33-MM-22");
    assertEquals({}, card1.getPlates());

);

public testOperation: () ==> ()
testOperation() == (
    dcl MySystem: MySystem := new MySystem();
    dcl group: Group := MySystem.addGroup("Students");
    dcl park1: ParkingLot := MySystem.addParking(30, group);
    dcl date1: Date := new Date(31,12,2017);
    dcl date2: Date := new Date(2,2,2017);
    dcl card1: Card := MySystem.addCard(group, "John Doe", date1);
    dcl op1: Operation;
```

```

dcl op2: Operation;
dcl op3: Operation;

card1.addPlate("33-MM-22");
op1 := MySystem.addOperation("33-MM-22", date2, card1, park1, <Enter>);
op2 := MySystem.addOperation("33-MM-22", date2, card1, park1, <Enter>);

assertEqual(<Allow>, op1.getResult());
assertEqual(<Deny>, op2.getResult());
assertEqual(29, MySystem.getParkingCurrLotation(park1));

op3 := MySystem.addOperation("33-MM-22", date2, card1, park1, <Leave>);
assertEqual(<Allow>, op1.getResult());
assertEqual(30, park1.getCurrLotation());

assertEqual({op1, op2, op3}, MySystem.getCardOperations(card1));

);
end Test

```

Function or operation	Line	Coverage	Calls
Test	8	100.0%	1
testCard	34	100.0%	1
testMySystem	14	100.0%	1
testOperation	61	100.0%	1
testSystem	14	100.0%	1
Test.vdmpp		100.0%	5

5. CODE GENERATION

After successfully generating Java code, a console interface was built to test all the features. One of the things to change was 'seq of char' to 'String' in order to use the 'Scanner' class in Java. The interface is easy and minimalist, with menus navigation in lists with the option chosen by the user. Below are some examples.

```
*****
***MENU***
*****

1. Add Group
2. Add Card
3. Add Parking Lot
4. Register Operation
5. Get Card Operations
6. Cancel Card
7. Reactivate Card

Option: 1
```

```
*****
***ADD CARD***
*****

CHOOSE A GROUP FROM BELOW:

0. Students
Group ID: 0

-----
Name: Ricardo
Day of Expiration Date: 1
Month of Expiration Date: 1
Year of Expiration Date: 2018
Add a Plate: 55-MM-FF

Press 1 to add another one; else to add the card as it is: 0
Card 'Ricardo' added!
Enter 0 for back menu...
```

6. CONCLUSIONS

The system does most of what is required, except renovating the expiration date of the card. Some more conditions and invariants could be added. Also, the class 'Date' should try to verify months with different number of days. However, the usage of VDM++ on this particular project contributed to my knowledge of the language and understanding the usefulness of it.

7. REFERENCES

1. Class notes available in Moodle.
2. Overture IDE Guide.
3. \LaTeX documentation to write report.