Complete Document

for

IIT Security System (IITSS)

By

Sunandita Patra

Group ID: 95, Roll number: 09CS3037

2nd Year Dual Degree Student

Department of Computer Science and Engineering

IIT Kharagpur

18th April, 2011

Software Requirements Specification

for

IIT Security System

Version 1.0

1st March, 2011

Table of Contents for the SRS

Revision History

Name	Date	Reason For Changes	Version
None	None	None	None

1. Introduction

1.1 Purpose

This document is the Software Requirement Specification for the IIT Security System version 1.0. IIT Security System is a software to automata entry, exit and monitoring of vehicles around the Campus, i.e. to control and monitor vehicular traffic in and out of the Campus.

1.2 Document Conventions

The following conventions have been used throughout this document:

1.2.1 Font: Calibri

1.2.2 Size 18 for main headings

1.2.3 Size 14 for sub-headings

1.2.4 Size 11 for the rest of the document

1.3 Intended Audience and Reading Suggestions

This document is aimed at helping the users (security head security officials of the institute), developers, project managers, testers, and documentation writers. It tries to explain the main objective to create an automated security system and also support the developers.

The user can come in terms with the main features of the software and use it accordingly.
The different readers are suggested to first go through the overview and then proceed accordingly as follows:

Security Official: Functional Requirements
Manager: Communication Interfaces
Developers: Functional as well as non-functional requirements
Documentation writers: sequentially from the beginning to end

1.4 Project Scope

IIT Security System is a software to control and monitor vehicles that move around in the Campus roads.
In Scope:

- Keeps record of all the vehicles belonging to the Campus residents.
- Allows the Campus vehicles in and out of the Main Gate.
- The record of outside vehicles that enter the Campus for some specified reasons is kept separately.
- Keeps track of vehicles that have disobeyed the traffic rules and have been involved in rash driving incidents.
- Provides data of vehicles present inside and outside the Campus to authorized personnel.
- Can disallow a vehicle to enter the Campus

Out of Scope:

- The system needs a monitoring official always and doesn't interact with the vehicle owner directly.
- Cannot check the validity of information that is being stored during registration.

- Cannot interpret the registration number of the vehicle that is trying to enter. Needs external support (camera or Keyboard input)

This software reduces the load on security officers to check whether a vehicle belongs to the Campus or not. It depends on the honesty and reliability of the security officer present at the Main Gate.

1.5 Definitions, Acronyms, and Abbreviations

IITSS: IIT Security System
IIT: Indian Institute of Technology
OS: Operating System

1.6 References

1.6.1 An Integrated approach to Software Engineering by Pankaj Jalote

1.6.2 Fundamentals of Software Engineering by Rajib Mall.

2. Overall Description

2.1 Product Perspective

IITSS performs a small part for a larger developed security system for the Campus. IITSS is a small system but is self-sufficient. It interacts with the camera and the security officer at the Main Gate to get information (registration number, colour, etc) about a vehicle. Thus, IITSS is a standalone system.

2.2 Product Features

IITSS is designed to support the security system of the Campus in keeping track of the road traffic and errant vehicles. In case of any indisciplinary activity, a vehicle can be barred from entering the Campus. Information can be obtained by authorized personnel about the vehicles present inside the Campus at a particular time and also enquire whether a vehicle is present in the Campus or not, i.e. the entry and exit time of a vehicle.

2.3 User Classes and Characteristics

The major user classes that are expected to use the software are as follows:

Security Staff

The officers who will be in-charge of the Main Gate will mainly use IITSS. They register the information about a particular vehicle after confirming it by their own means. And they also keep record of vehicles that have broken traffic rules and justifiable reasons given by the drivers. They can obtain information about a particular vehicle.

Camera

The camera at the main gate takes a picture of the vehicles and enters the registration number of the vehicle in the IITSS.

Dean of Campus Affairs

The Dean may check the software data when he or she wishes.

2.4 Operating Environment

OS: Windows XP or later
Software packages: Text reader/editor

2.5 Design and Implementation Constraints

Database management is not being used to store all the information related to the vehicles. Simple file-handling techniques are used, so the data may not be very much organized. The customer's organization will need to seek help from the developers' team in case of any problem with IITSS. The input from the camera is simulated by keyboard entry.

2.6 User Documentation

IITSS will have a 'Help' menu, which will assist the user in case of any query or doubts. Tutorials will be given for the Security Manager to conduct classes for his juniors.

2.7 Assumptions and Dependencies

It depends on the camera to give correct details of the vehicle that is entering.

3. System Features

3.1 Register a Campus Vehicle

3.1.1 Description and Priority

This feature helps to keep record of all the vehicles owned by the Campus residents. Only the vehicles that are registered are allowed to enter automatically.
Priority: High
Benefit: 4.7/5
Penalty: 4.9/5
Cost: 10%
Rating: 4.7/5

3.1.2 Stimulus/Response Sequences

3.1.2.1 Click on Register.

3.1.2.2 Fill the form in the new Dialog.

3.1.2.3 Fill in the details: Registration number, Type of vehicle (Scooter/Motorbike/car), Model of vehicle, colour, Owner Name, Owner Designation, Owner ID.

3.1.2.4 Click on Register.

3.1.2.5 The vehicle is now registered.

3.1.3 Functional Requirements

Dealing with invalid inputs:

REQ-1: **Blank field:** In case any field is left blank, message will appear to inform that all fields are necessary.

REQ-2: **Data redundancy:** In case the registration number matches with the registration number of a vehicle that is already registered, then the user will get a message about that and the registration number field will be cleared.

3.2 Allow vehicles to enter and exit

A. Non-registered vehicles

3.2.1 Description and Priority

This feature helps in keeping record of the entry and exit time of a non-registered vehicle (outside campus or campus vehicle which has not been registered yet). This checks that the vehicles that have been banned inside the Campus are not allowed to enter into the Campus.

Priority: High
Benefit: 4.9/5
Penalty: 4.9/5
Cost: 25%
Rating: 4.9/5

3.2.2 Stimulus/Response Sequences

3.2.2.1 Allowing entry

3.2.2.1.1 Enter the vehicle registration number.

3.2.2.1.2 Message will appear to inform that it is a non-Campus vehicle.

3.2.2.1.2.1 Entry is allowed. New form appears.

3.2.2.1.2.1.1 Enter the Purpose of visit.

3.2.2.1.2.1.2 Enter the name of the driver/owner (if present inside the car).

3.2.2.1.2.1.3 Enter the model, colour etc, i.e., other details regarding the vehicle.

3.2.2.1.2.1.4 Click on ‘Vehicle enters’.

3.2.2.1.2.2 Entry is not allowed. Message is displayed and vehicle is not allowed to enter.

3.2.2.2 Allowing exit

3.2.2.2.1 Enter the vehicle’s registration number.

3.2.2.2.2 Click on ‘vehicle exits’.

3.2.2.2.2.1 Vehicle is not delayed or reported. Vehicle exits.

3.2.2.2.2.2 Vehicle is delayed by atleast 8 hours.

3.2.2.2.2.2.1 Enter the reason for delay in the ‘Reason for delay’ field.

3.2.2.2.2.2.2 Click ‘Exit’.

3.2.2.2.2.3 Vehicle has been reported by some security officer.

3.2.2.2.2.3.1 The report description is displayed.

3.2.2.2.2.3.2 Enter the reason for breaking the rule given by the driver in the 'Reason' field.

3.2.2.2.2.3.2.1 The reason is **satisfactory**. Click on 'Satisfactory' to allow exit and now the vehicle can enter the Campus again as required.
3.2.2.2.2.3.2.2 The reason is **unsatisfactory**. Click on 'Unsatisfactory'. Vehicle exits but will be banned from any further entry in the Campus afterwards.

B. Registered Vehicles

3.2.1 Description and Priority

This helps in keeping record of the Campus vehicles activity, i.e., when it is going in or going out.

Priority : Medium
Benefit : 4.5/5
Penalty : 4.4/5
Cost : 25%
Rating : 4.4/5

3.2.2 Stimulus/Response Sequences

3.2.2.1 Allowing entry

3.2.2.1.1 Enter the vehicle registration number.

3.2.2.1.2 Click on Vehicle enters.

3.2.2.2 Allowing exit

3.2.2.2.1 Enter the vehicle registration number.

3.2.2.2.2 Vehicle exits.

3.2.2.2.3 In case the vehicle has been reported earlier, the report will be displayed. The driver will be asked to explain.

3.2.2.2.3.1 Enter the response given in the 'Reason provided' field.

3.2.2.2.3.2 The security officer-in-charge judges whether the response is satisfactory or not.
3.2.2.2.3.2.1 **Satisfactory response**: The response is stored.
3.2.2.2.3.2.2 **Unsatisfactory response**: Letter to be issued to dean of Campus affairs is generated.
3.2.2.2.3.2.3 The vehicle exits. The record of this incident is kept in the reports' database.

3.2.3 Functional Requirements

REQ-1: **Clock:** any time keeping device to give the current date and time, so that the entry time and exit time can be calculated.

REQ-2: **Wrong input:** In case a wrong registration number is put while exiting, message will appear to inform that the vehicle is not in the Campus at present.

REQ-3: **Blank field:** Until a response is entered for various 'reason' fields, vehicle is not allowed to exit.

3.3 Report errant vehicles

3.3.1 Description and Priority

If the vehicles disobeying the traffic rules are reported, then people will be more conscious of the rules and that ensures the safety of the Campus residents from accidents due to carelessness or rash driving.

Priority: High
Benefit: 4.9 5/5
Penalty: 4.95/5
Cost: 5%
Rating: 4.95/5

3.3.2 Stimulus/Response Sequences

3.3.2.1 Click on 'Report a vehicle'.

3.3.2.2 Enter the registration number and the description of the misbehavior.

3.3.2.3 Vehicle has been reported and action will be taken during its next exit.

3.3.3 Functional Requirements

REQ-1: **Communication:** Effective means of communication between the traffic officers present throughout the campus.

REQ-2: **Dealing with invalid inputs**:
A registration number is entered such that the vehicle with that registration number is not present in Campus at the moment. Report is ignored and not recorded.

3.4 View records

3.4.1 Description and Priority

The security officer can view the total number of vehicles that have been using the campus roads and also enquire about a particular vehicle.

Priority: Medium
Benefit: 4.6 /5
Penalty: 4.5/5
Cost: 25%
Rating: 4.5/5

3.4.2 Stimulus/Response Sequences

Click on 'View past records'.
Click on the following to view the particular information.

3.4.2.1 '**Total number of vehicles'**

- **Week:** Displays the total number of vehicles in the campus in the past one week.
- **Month:** Displays the total number of vehicles in the campus in the past one month.
- **Year:** Displays the total number of vehicles in the campus in the past one year.

3.4.2.2 **'Campus vehicles present outside'**: Displays the list of registered vehicles present outside the Campus along with their departure times.

3.4.2.3 **'Registered vehicles':** Displays the list of vehicles belonging to the Campus residents that have been registered.

3.4.2.4 To enquire about a particular vehicle, enter its registration number and the details about it will be displayed.

3.4.3 Functional Requirements

REQ-1: Text files handlers and editors.

4. External Interface Requirements

4.1 User Interfaces

There are mainly seven user interfaces. The first GUI will have the links to all other six interfaces.

4.1.1 **Welcome page.** It contains the field to enter the registration number of the vehicle that wants to enter the Campus and buttons to go to the following pages. The left pane contains the buttons and right pane contains the text field.

4.1.2 **Register a vehicle.** Enter the necessary information required to register a vehicle. Separate fields are present one after the other in different lines.

4.1.3 Allow **entry to a non-registered vehicle**. It contains fields to enter the necessary information.

4.1.4 **Exit** for a registered/non-registered vehicle that has broken traffic rules. Contains the necessary fields as specified in Section A 3.2.2.2 and B 3.2.2.2.

4.1.5 **Report** about a vehicle. Contains the necessary fields as specified earlier in Section 3.3.2.

4.1.6 Obtain past **records**. It contains buttons for various information (Specified in Section 3.4.2) that can be displayed and the displaying window in which the information is to be displayed in the right half.

4.1.7 **Error** messages, **warnings** and **notifications** are displayed using a message box.

4.2 Hardware Interfaces

4.2.1 Printer: To print the letter to be sent to the Dean of Campus affairs.

4.2.2 Camera.

4.3 Software Interfaces

4.3.1 Software will depend on the security features provided by the operating system and the language Java.
4.3.2 All the important data are stored in text files so a text editor/reader like Notepad or WordPad (any version) is must.

4.4 Communications Interfaces

4.4.1 When a vehicle tries to enter the campus, the camera takes its picture and provides it to the IITSS database. The image is transferred through the USB port along with the registration number of the vehicle.

4.4.2 IITSS directs the gate to open or close through a digital control interface.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

5.1.1 Down time should be limited to a maximum of 5 minutes otherwise it will lead to a traffic jam as vehicles will be queued up inside as well as outside the Campus, i.e., on both the sides of the main gate.

5.1.2 Single user is supported (security officer).

5.1.3 Should run on a 500 MHz, 64 MB Machine.

5.1.4 95% of the responses should be within 1 second to ensure smooth traffic.

5.1.5 Data should not become corrupted in case of system crash or power failure.

5.2 Safety Requirements

IITSS is perfectly safe to use and it cannot to any harm to the other components of the system in which it is being used. It should be taken care that no files are stored in the folder containing all data related to IITSS to ensure that it is not modified or deleted.

5.3 Security Requirements

5.3.1 The files in which the information about the vehicles is stored should be secured against malicious deformations.
5.3.2 Since campus security is a critical operation, adequate safety against cyber attacks on the security software should be ensured.
5.3.3 The only person authorized to use IITSS is the security officer-in-charge of the main gate. He should provide a registration number/password to get access to important and secured data.

5.4 Software Quality Attributes

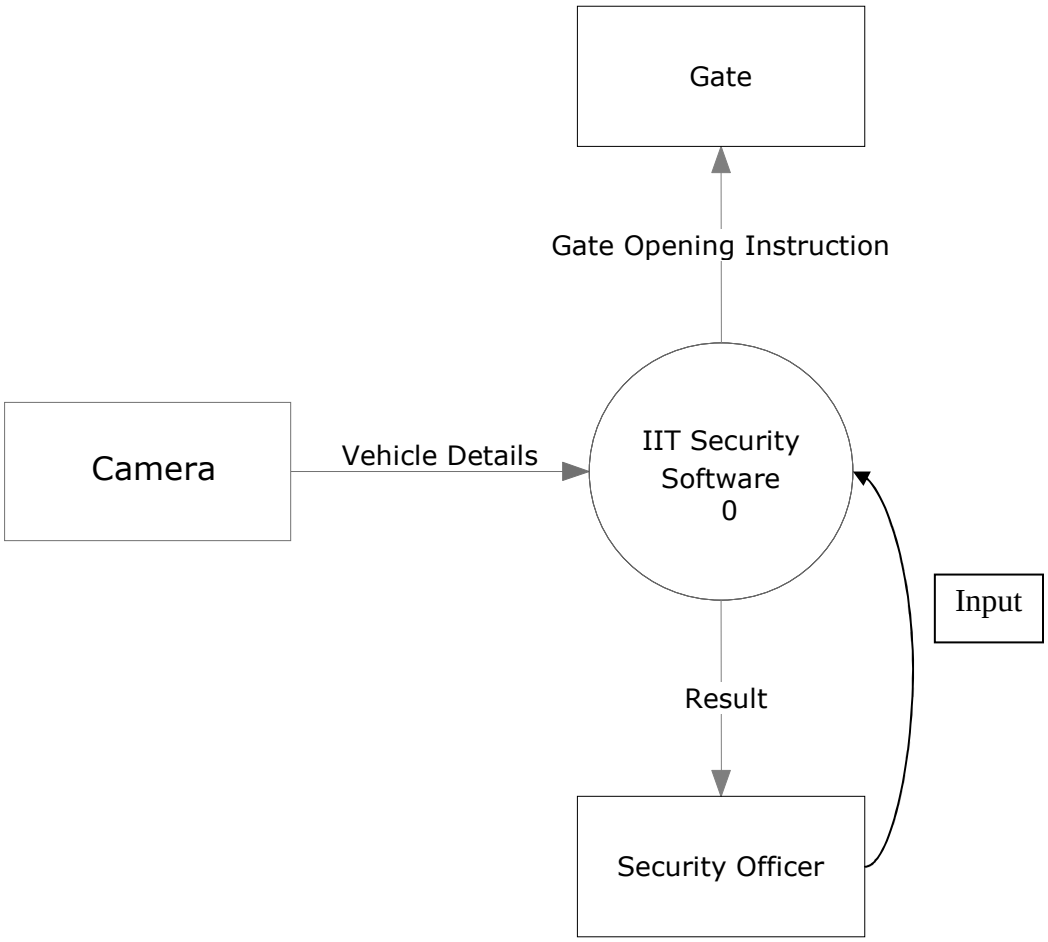
The expected quality of the software is graded (out of 5) as follows:

Adaptability	4.5
Availability	4.0
Correctness	4.8
Flexibility	4.6
Interoperability	4.7
Maintainability	4.6
Portability	4.9
Reliability	4.6
Reusability	4.8
Robustness	4.7
Testability	4.5
Usability	4.9

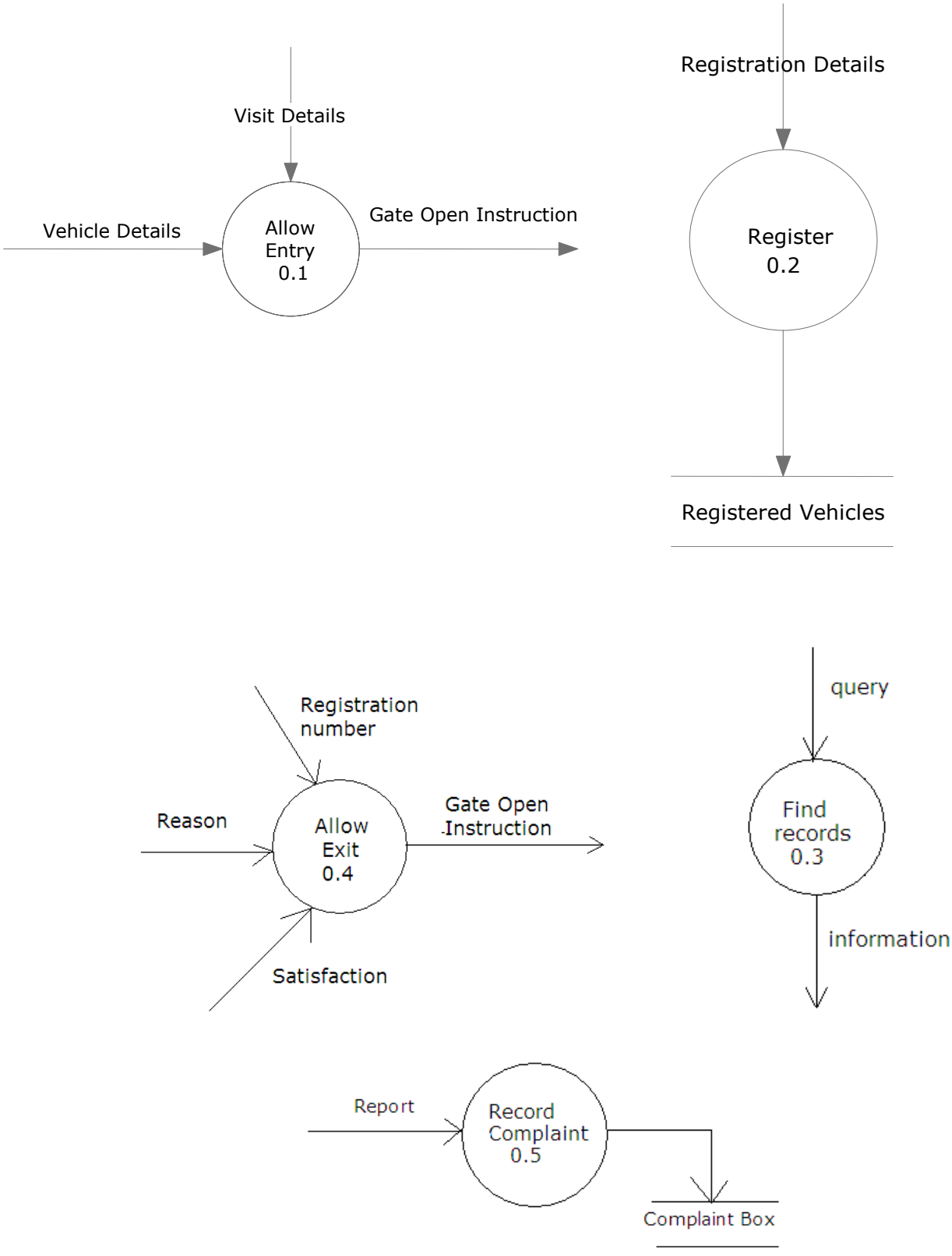
Analysis Model

Data Flow Diagram

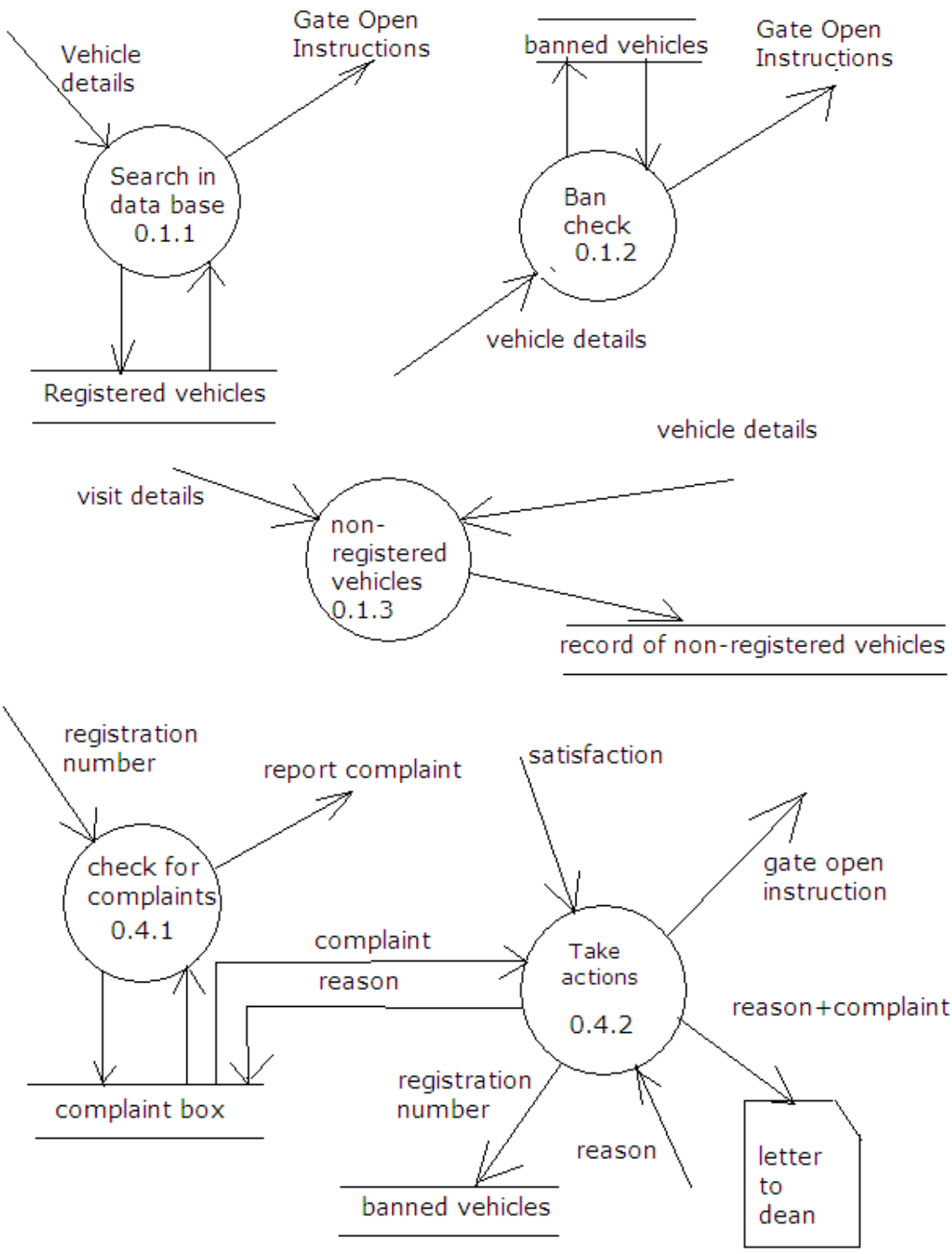
Context Diagram:



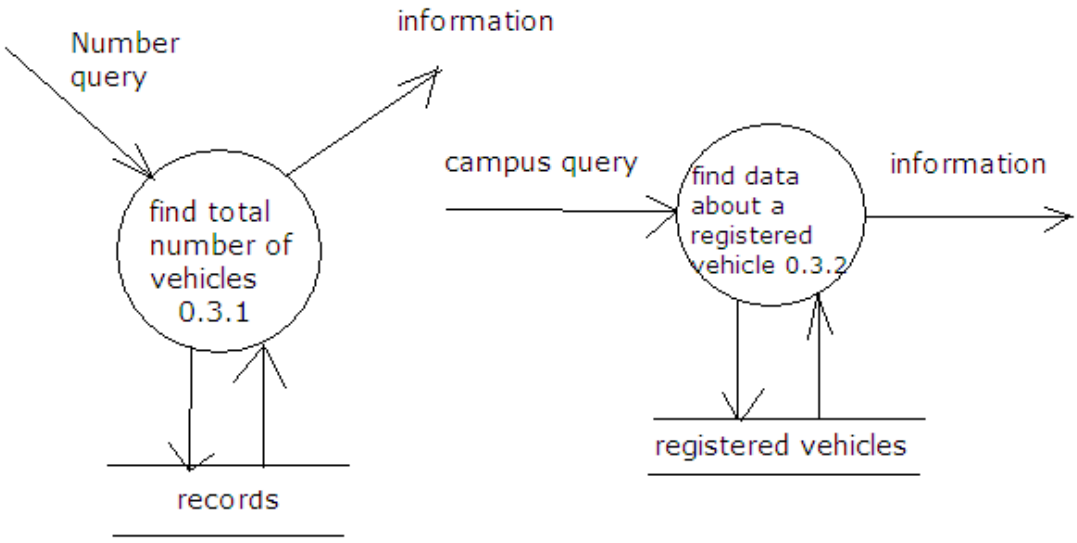
Level 1 DFD


Visit Details
Vehicle Details
Allow Entry 0.1
Gate Open Instruction
Registration Details
Register 0.2
Registered Vehicles
Registration number
Reason
Allow Exit 0.4
Gate Open Instruction
Satisfaction
query
Find records 0.3
information
Report
Record Complaint 0.5
Complaint Box

Level 2 DFD



Continued in the next page…


Number
query
information
find total
number of
vehicles
0.3.1
records
campus query
find data
about a
registered
vehicle 0.3.2
information
registered vehicles

Data Dictionary for the above DFD model

Vehicle Details: Registration number + colour + Model + Image} /* of the vehicle at the gate*/

/*names have their usual meanings*/

Gate open Instruction: {“Open gate”, “don’t open gate”}

Visit details: purpose of visit + name of the owner

Complaint: String /*report made by a security officer*/

Reason: String /*reason given by the driver for his misbehavior*/

Purpose of visit: String

Name of the owner: String

Satisfaction: {“Satisfied”, “Not Satisfied”}

Report Complaint: String

Report: String

Number query: {“Week”, “Month”, “Year”}

Campus query: [Registration number, name of the owner]

Information: [integer, String]

Query: [Campus query, Number query]

Registration details: Registration number + type + colour + model + owner name + owner ID + owner designation.

Input: Vehicle details + registration details + satisfaction + reason + query + report

Result: information + complaint

/* Storage units names have their usual meanings*/

Registered vehicles: text file

Complaint box: text file

Records: text file

Record of non-registered vehicles: text file

Banned vehicles: text files; Letter to Dean: text file

Object oriented Analysis and Design

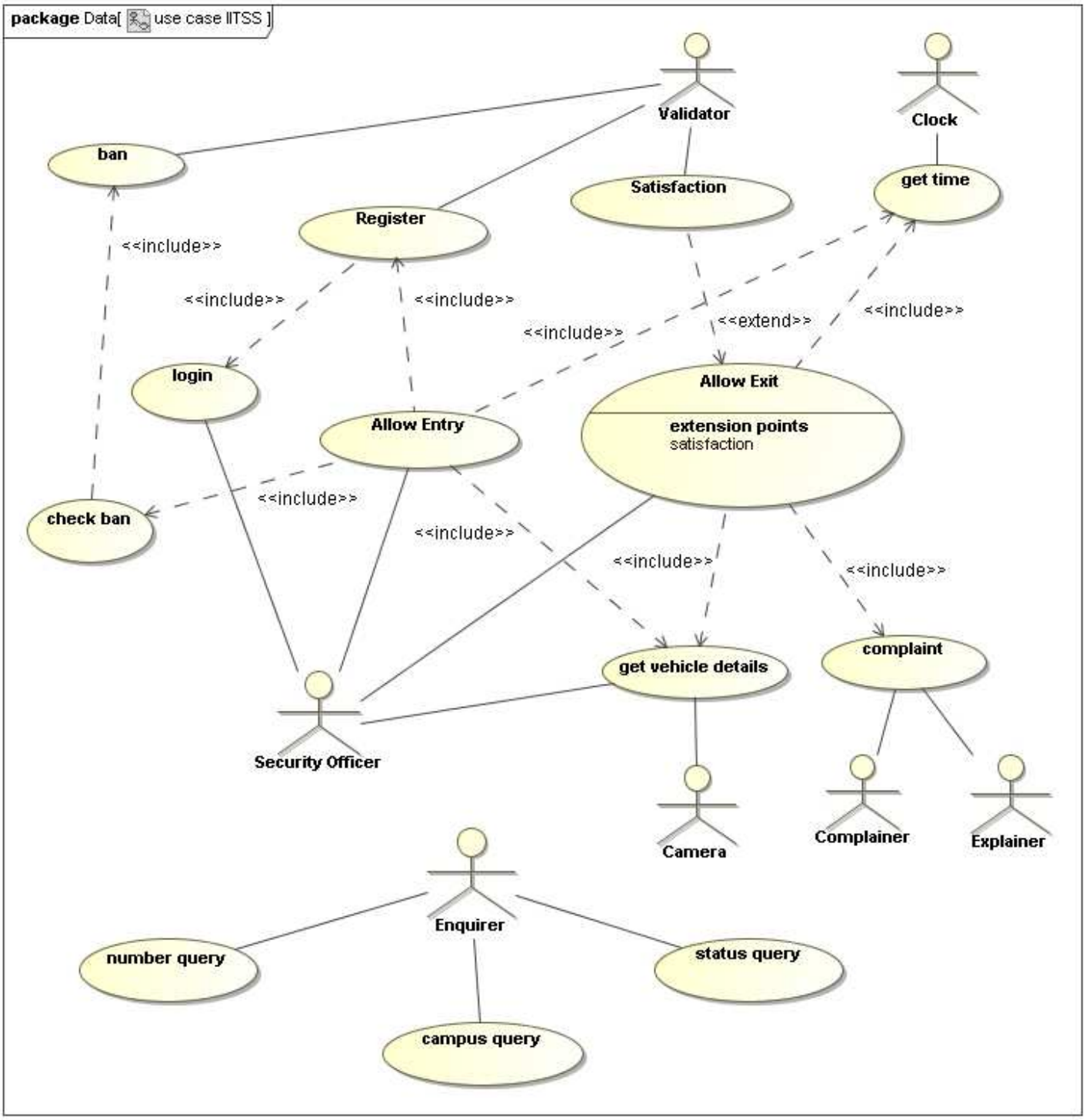
For

IIT Security System

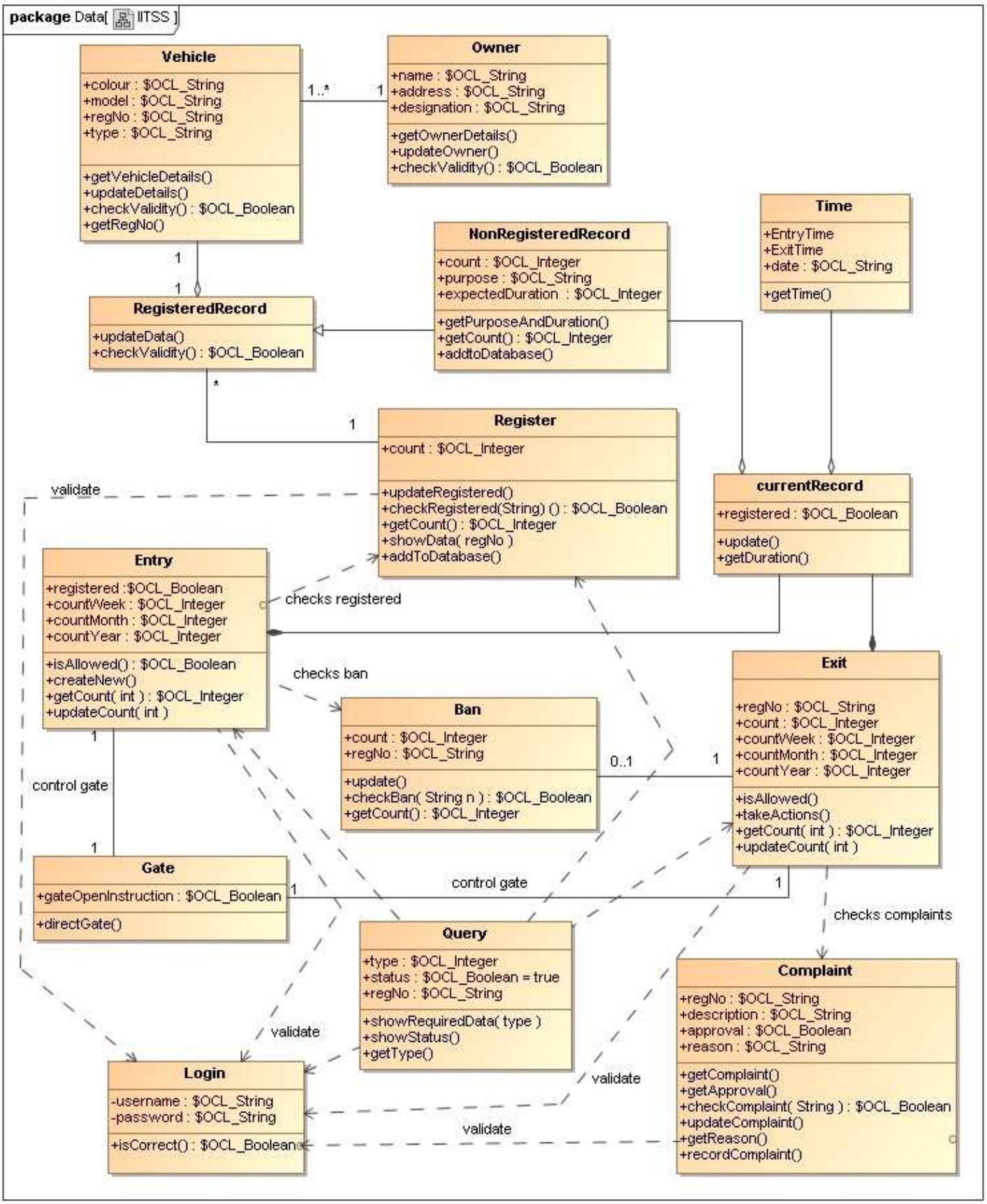
Version 1.0

30th March, 2011

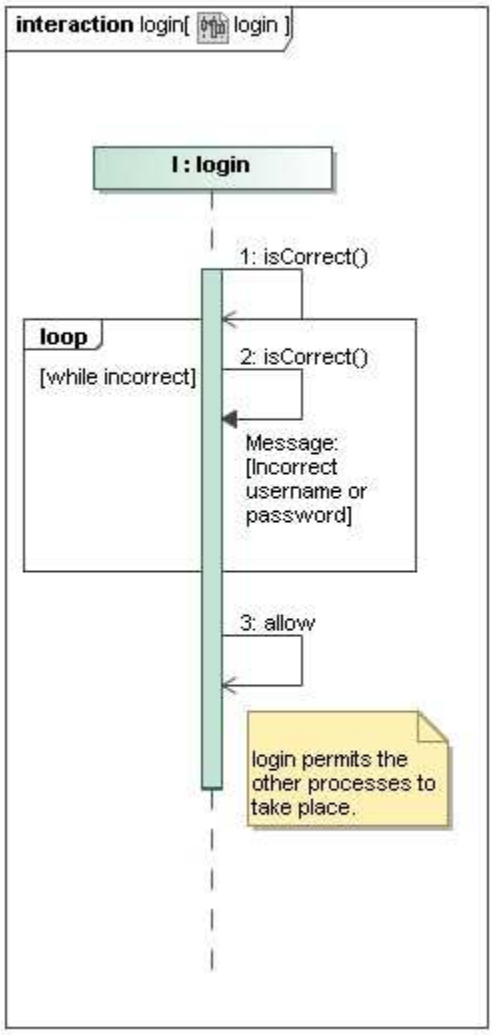
Use Case Diagram

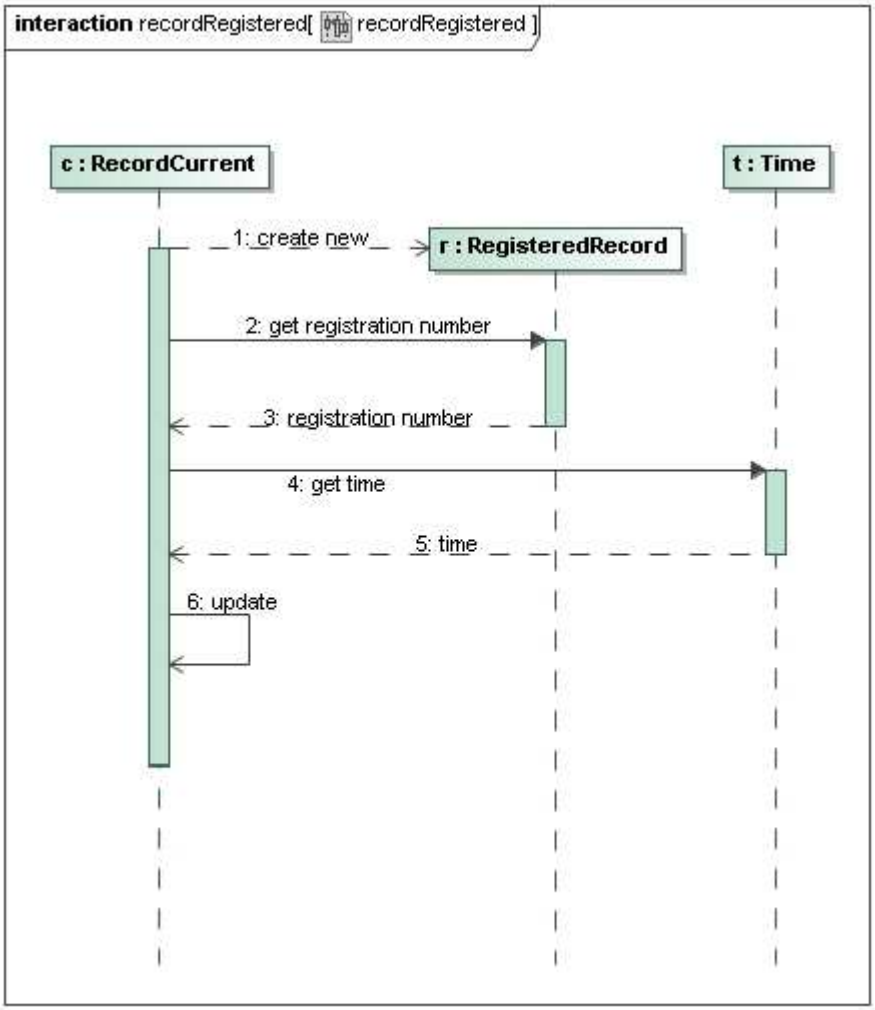


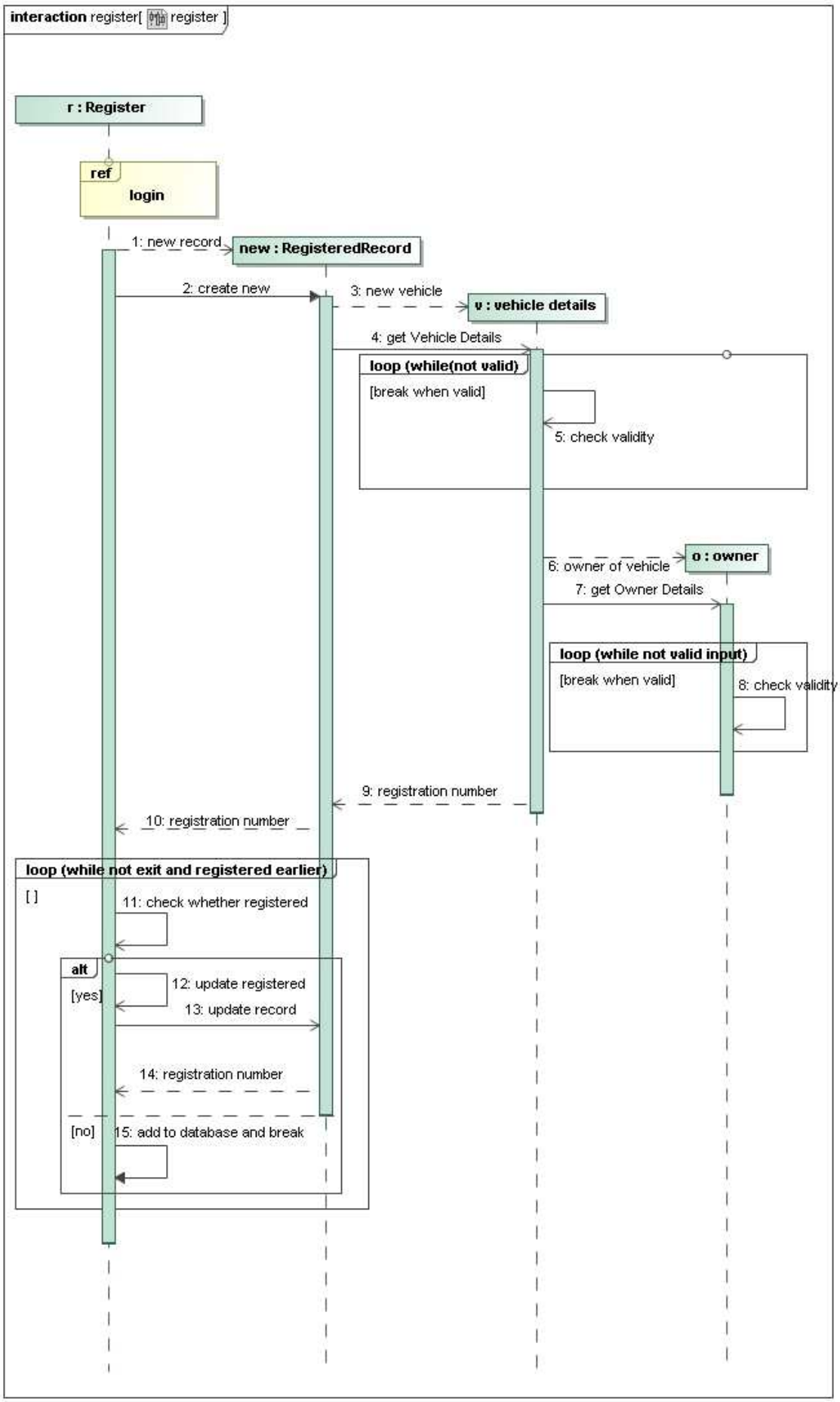
Class Diagram

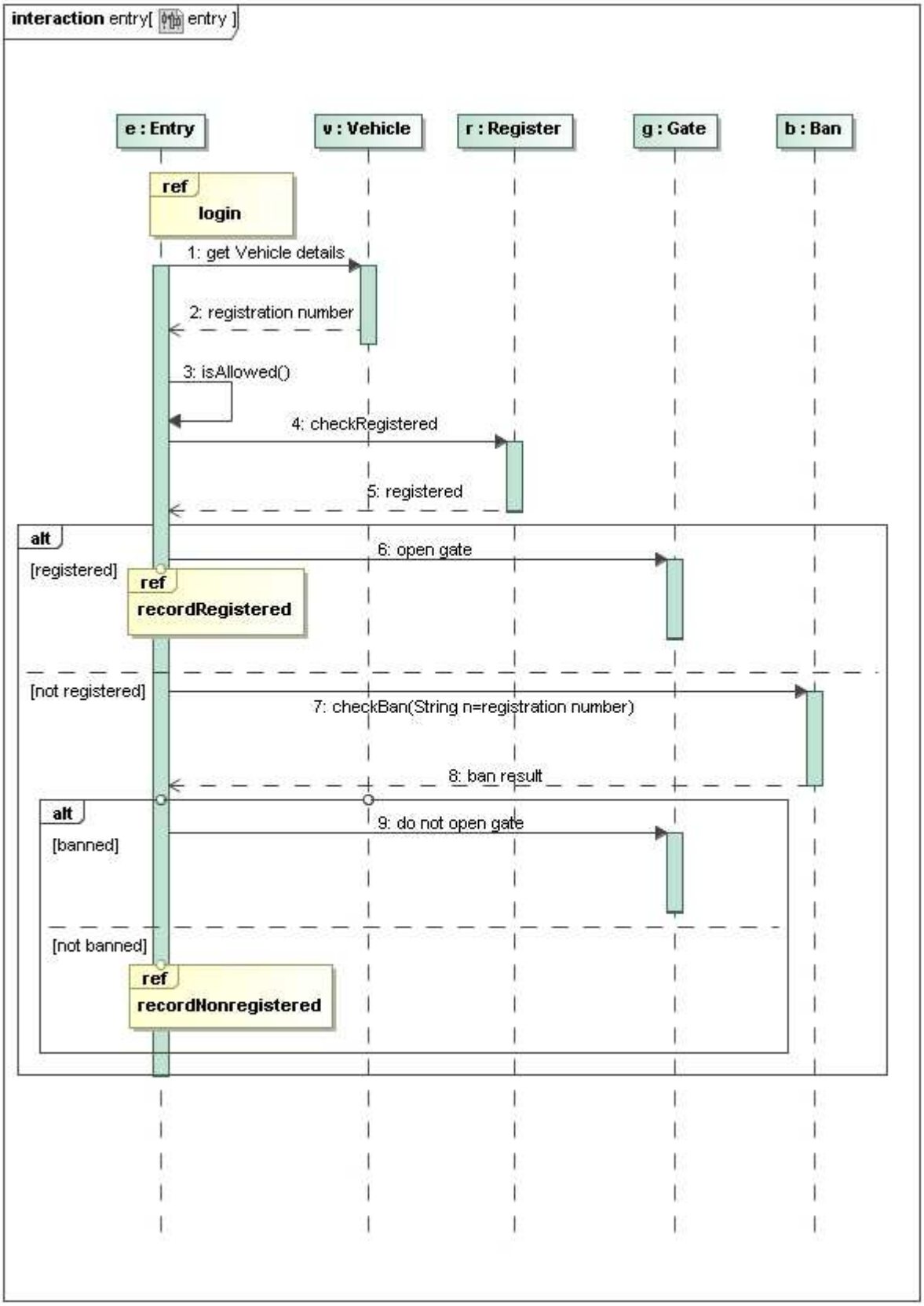


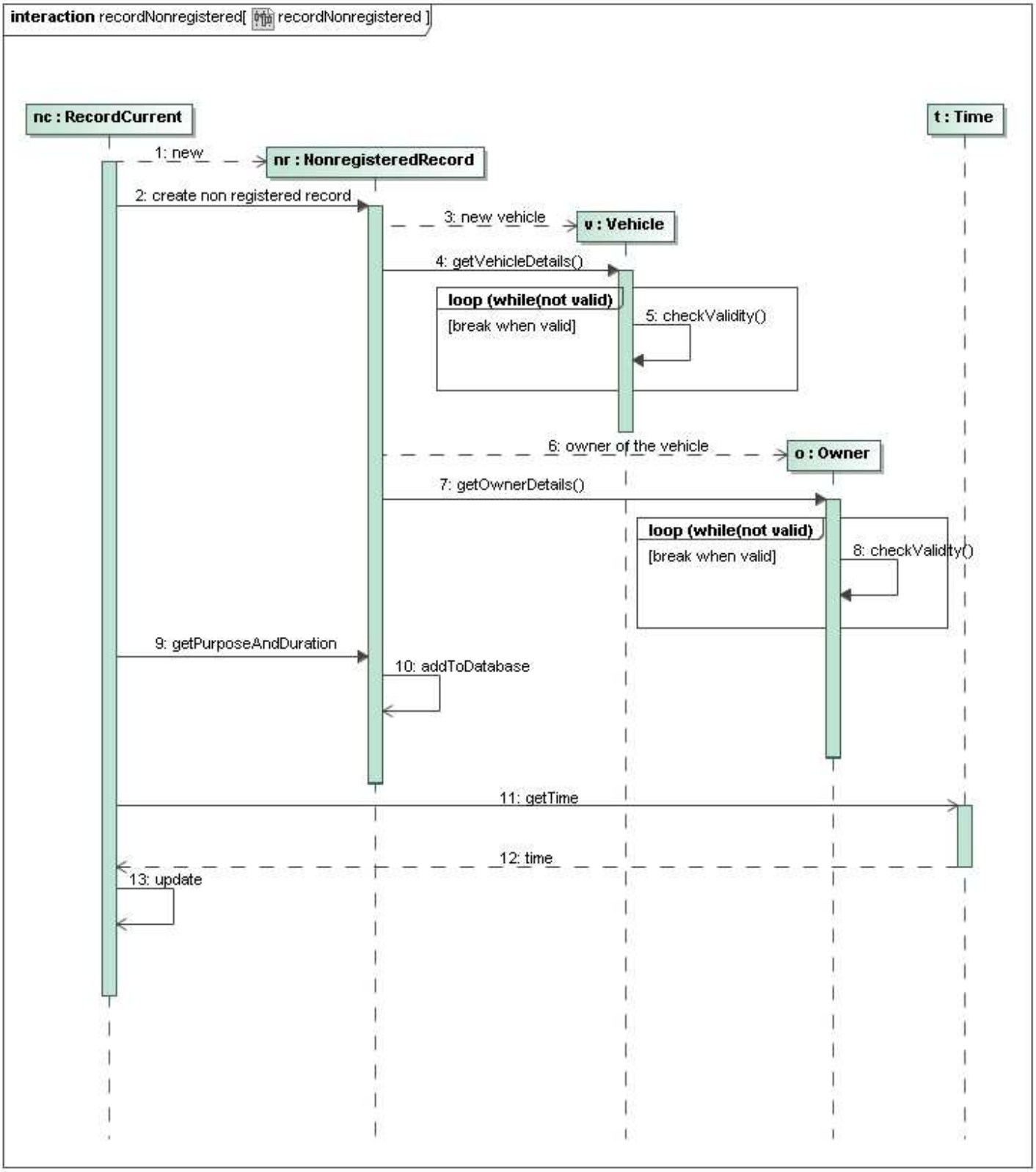
Sequence Diagrams

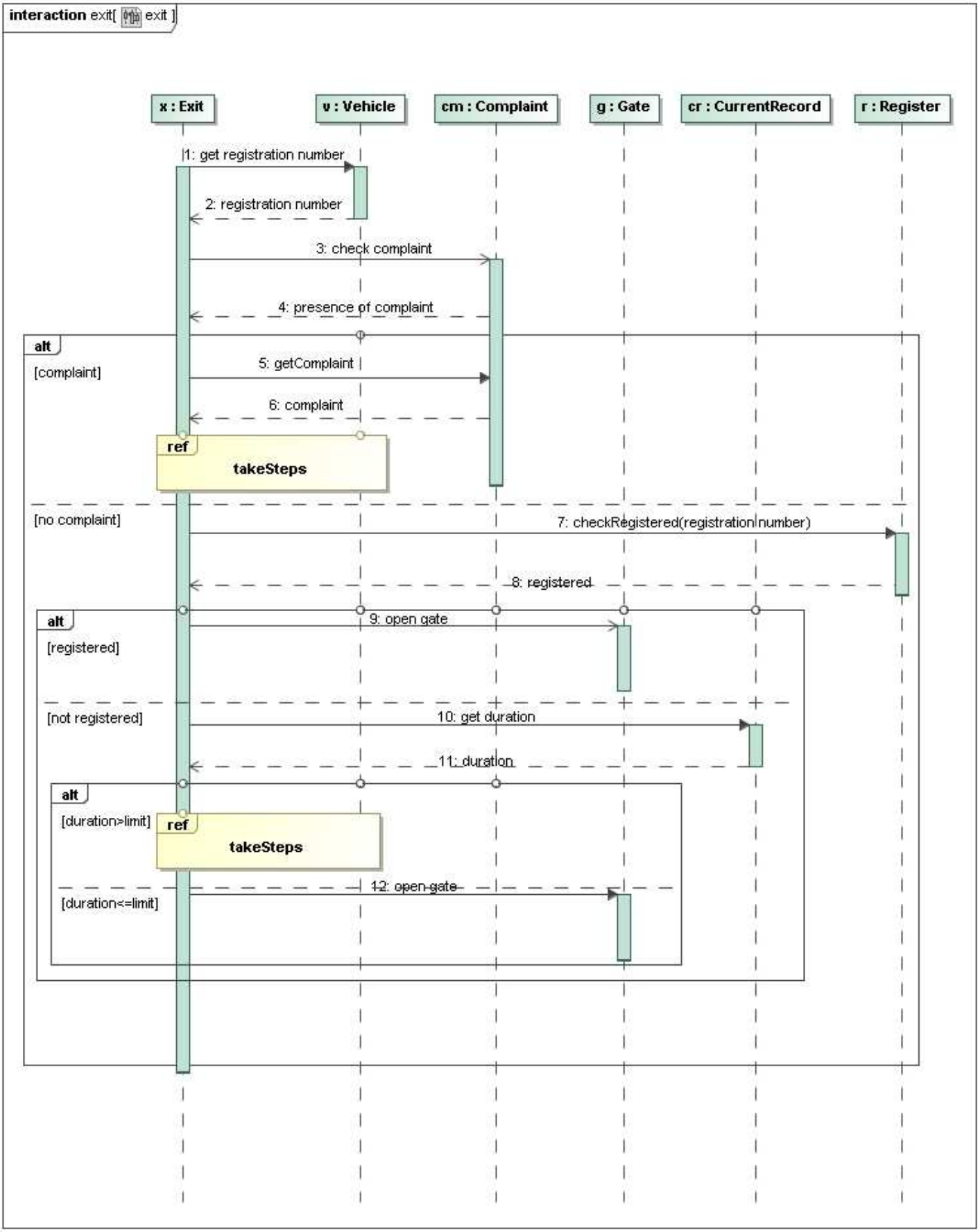


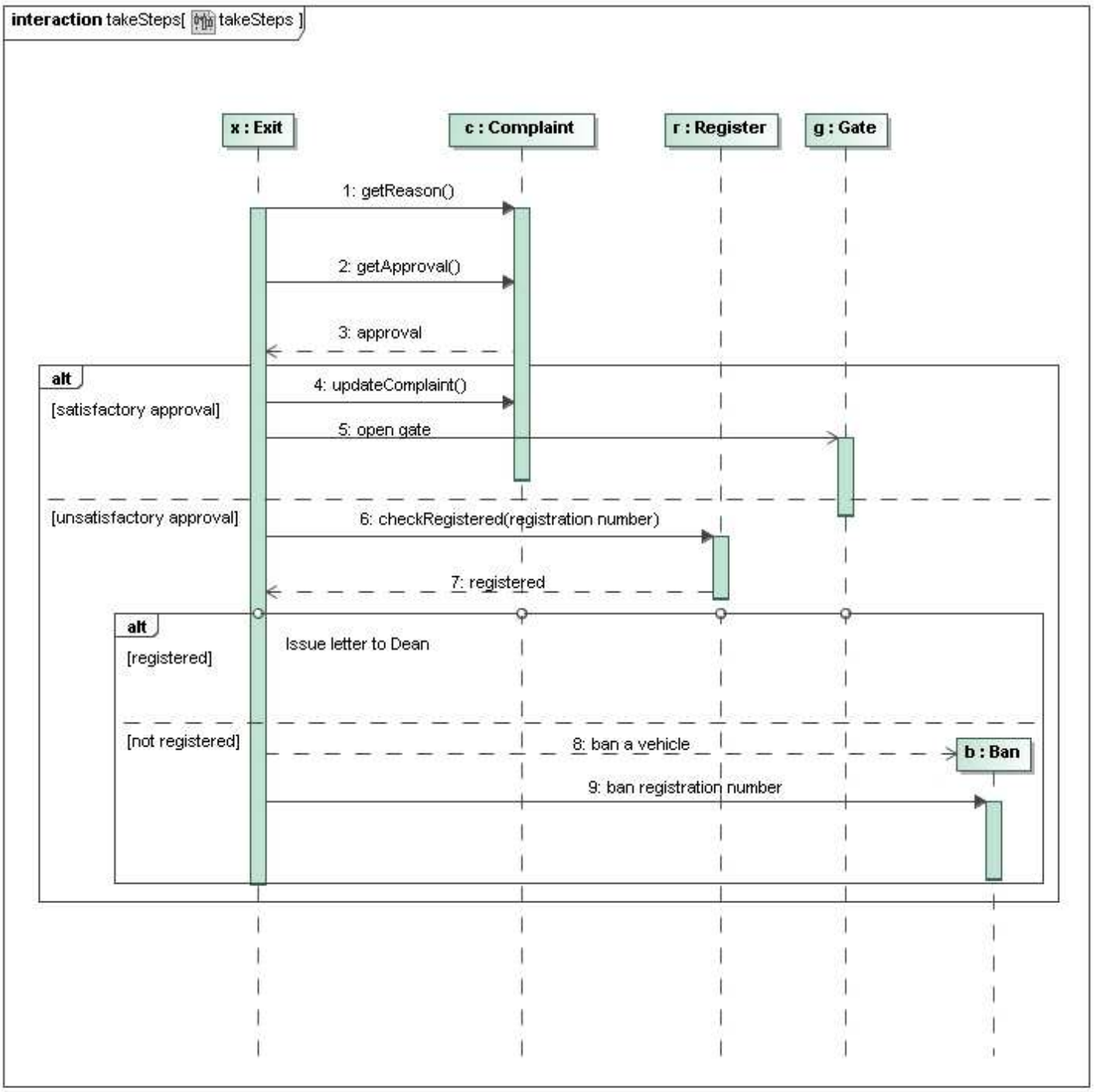


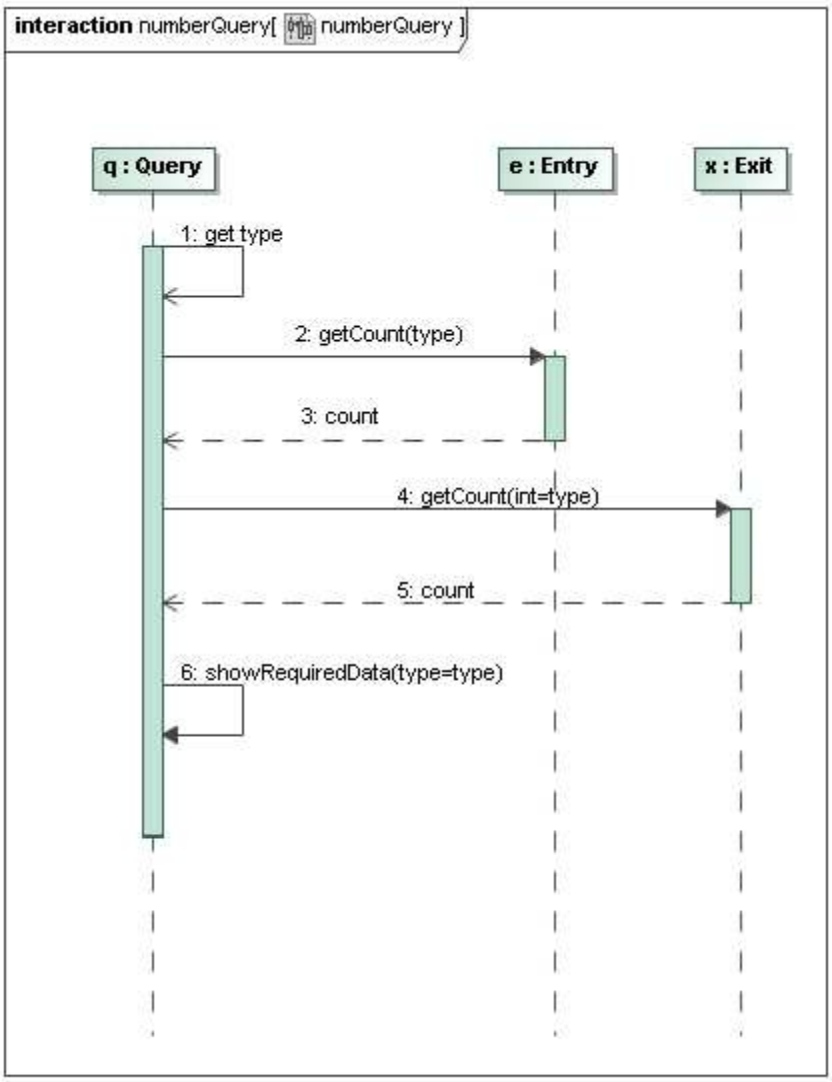

interaction register[register]
r : Register
ref
login
1: new record
new : RegisteredRecord
2: create new
3: new vehicle
v : vehicle details
4: get Vehicle Details
loop (while(not valid)
[break when valid]
5: check validity
6: owner of vehicle
o : owner
7: get Owner Details
loop (while not valid input)
[break when valid]
8: check validity
9: registration number
10: registration number
loop (while not exit and registered earlier)
[ ]
11: check whether registered
alt
[yes]
12: update registered
13: update record
14: registration number
[no]
15: add to database and break

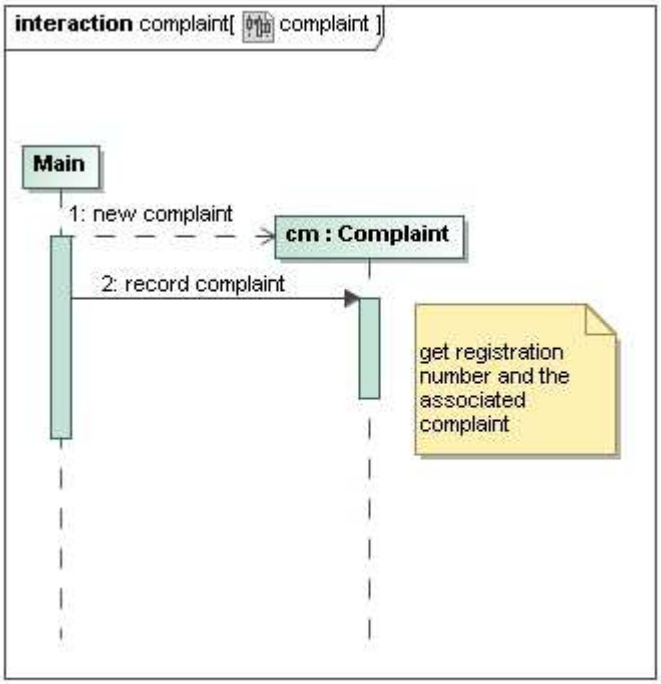

interaction entry[entry]
e : Entry
v : Vehicle
r : Register
g : Gate
b : Ban
ref
login
1: get Vehicle details
2: registration number
3: isAllowed()
4: checkRegistered
5: registered
alt
[registered]
6: open gate
ref
recordRegistered
[not registered]
7: checkBan(String n=registration number)
8: ban result
alt
[banned]
9: do not open gate
[not banned]
ref
recordNonregistered

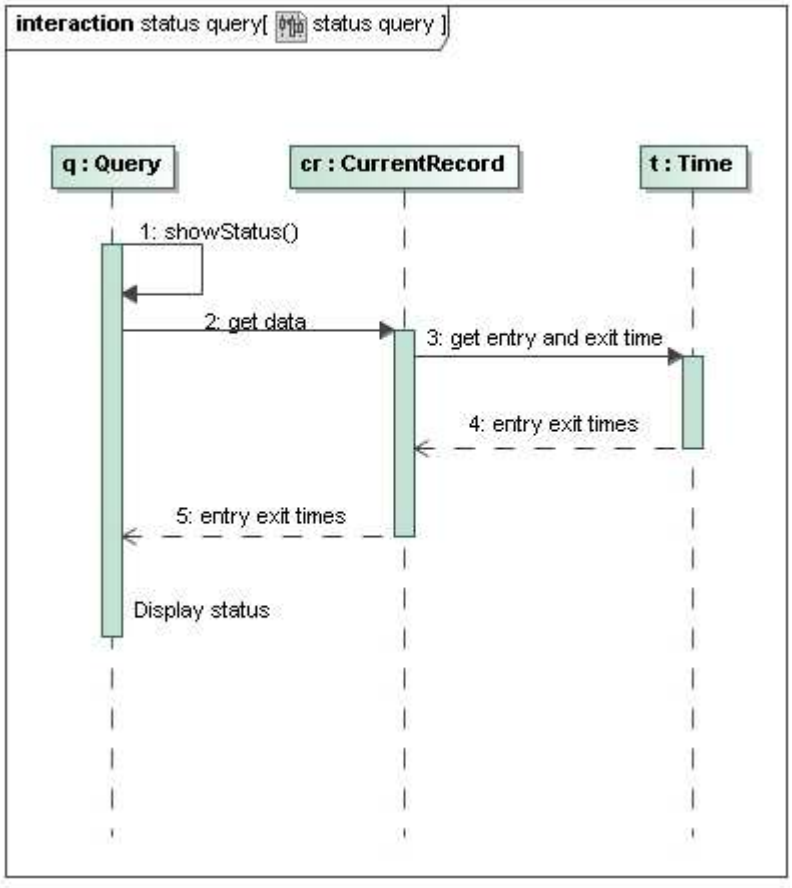

interaction recordNonregistered[recordNonregistered]
nc : RecordCurrent
t : Time
1: new
nr : NonregisteredRecord
2: create non registered record
3: new vehicle
v : Vehicle
4: getVehicleDetails()
loop (while(not valid)
[break when valid]
5: checkValidity()
6: owner of the vehicle
o : Owner
7: getOwnerDetails()
loop (while(not valid)
[break when valid]
8: checkValidity()
9: getPurposeAndDuration
10: addToDatabase
11: getTime
12: time
13: update

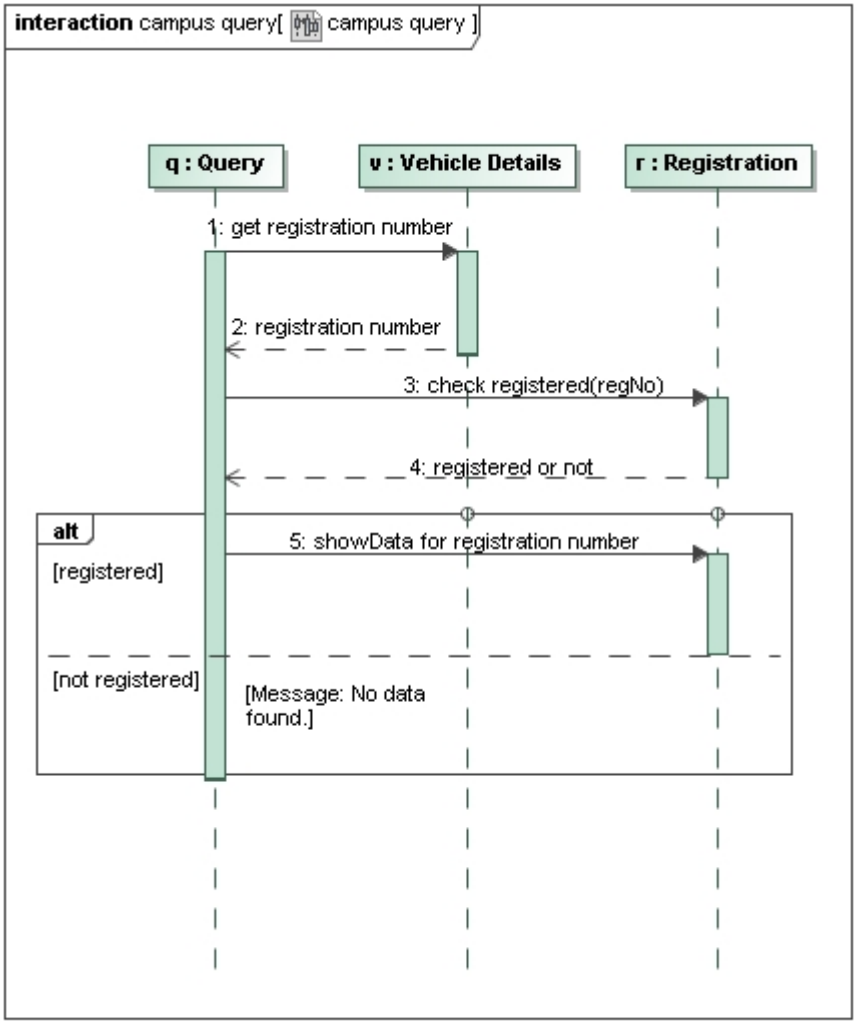

interaction exit[exit]
x : Exit
v : Vehicle
cm : Complaint
g : Gate
cr : CurrentRecord
r : Register
1: get registration number
2: registration number
3: check complaint
4: presence of complaint
alt
[complaint]
5: getComplaint
6: complaint
ref
takeSteps
[no complaint]
7: checkRegistered(registration number)
8: registered
alt
[registered]
9: open gate
[not registered]
10: get duration
11: duration
alt
[duration>limit]
ref
takeSteps
[duration<=limit]
12: open gate


interaction takeSteps[takeSteps]
x : Exit
c : Complaint
r : Register
g : Gate
1: getReason()
2: getApproval()
3: approval
alt
[satisfactory approval]
4: updateComplaint()
5: open gate
[unsatisfactory approval]
6: checkRegistered(registration number)
7: registered
alt
[registered]
Issue letter to Dean
[not registered]
8: ban a vehicle
b : Ban
9: ban registration number


interaction numberQuery[numberQuery]
q : Query
e : Entry
x : Exit
1: get type
2: getCount(type)
3: count
4: getCount(int=type)
5: count
6: showRequiredData(type=type)


interaction complaint[complaint]
Main
1: new complaint
cm : Complaint
2: record complaint
get registration number and the associated complaint


interaction status query[status query]
q : Query
cr : CurrentRecord
t : Time
1: showStatus()
2: get data
3: get entry and exit time
4: entry exit times
5: entry exit times
Display status


interaction campus query[campus query]
q : Query
v : Vehicle Details
r : Registration
1: get registration number
2: registration number
3: check registered(regNo)
4: registered or not
alt
[registered]
5: showData for registration number
[not registered]
[Message: No data found.]

Test Plan for IIT Security System

Login

1. Put wrong username or password.
2. Error message appears implies test successful.
3. Correct username and password, then ‘welcome’ window opens.

Code snippet:

```
private void loginActionPerformed(java.awt.event.ActionEvent evt) {

      userName = username.getText();

      passWord = password.getText();

      if(isCorrect())

      {

         String[] args = {"a"};

         Welcome.main(args);

      }

else    {

         JOptionPane.showMessageDialog(null,"Wrong username or password!");

      }

  }

  boolean isCorrect()

  {

  if(userName.equals("securityOfficer")&&passWord.equals("power"))    return true;

  else                                                                return false;

  }
```

Register

1. Check for some registration number of the wrong form.
2. Message should appear to inform invalidity.
3. Similarly for other fields, invalid entry is reported with message box
4. When valid input is given to all the fields, and the officer clicks on 'Register' button, the vehicle is registered.
5. Check registry by going to the query window and typing the registration number.
   Click on show details,
   First line should be vehicle is registered.
   Then the details follow.
6. The above implies test successful.

Code snippets:

```
private void registerBActionPerformed(java.awt.event.ActionEvent evt) {

      setDetails();

      if(r.checkValidity()==false)

      {

         JOptionPane.showMessageDialog(null, "Invalid input");

      }

    else{

         if(checkRegistered(r.v.regNo)==true)

         {

            JOptionPane.showMessageDialog(null, "Already registered.");

         }

         else addToDataBase();

      }

  }
```

```
 int getCount()
{
   try{
      File c = new File("registeredCount.txt");
      FileReader fr = new FileReader(c);
      BufferedReader br = new BufferedReader(fr);
      count = Integer.parseInt(br.readLine());
      fr.close();
   }
   catch(IOException e)
   {
      System.out.println("File Handling Error.");
   }
   return count;
}
boolean checkRegistered(String w)
{
   boolean match=false;
   String record;
   int c = getCount();
   try
   {
      File f = new File("register.txt");
      FileReader fr = new FileReader(f);
```

```
        BufferedReader br = new BufferedReader(fr);
        record = br.readLine();
        int i =1;
        while(i<=c)
        {
           System.out.println("\n"+record);
           if(record.equals(w))
           {
              match=true;
           }
        record=br.readLine();
        i++;
        }
      fr.close();
      }
      catch(IOException e)
      {
         System.out.println("File handling error.");
      }
     return (match);
   }
void addToDataBase()
   {
    String eol = System.getProperty("line.separator");
```

```
try
{
   File f = new File(r.v.regNo+".txt");
   FileWriter w = new FileWriter(f);
   w.append(r.v.regNo);
   w.append(eol+r.v.type);
   w.append(eol+r.v.model);
   w.append(eol+r.v.colour);
   w.append(eol+r.v.Owner.name);
   w.append(eol+r.v.Owner.designation);
   w.append(eol+r.v.Owner.address);
   w.flush();

   File f2 = new File(r.v.regNo+"status.txt");
   FileWriter w2 = new FileWriter(f2);
   w2.write("true");
   w2.close();

File c = new File("registeredCount.txt");
FileReader fr = new FileReader(c);
BufferedReader br = new BufferedReader(fr);
count = Integer.parseInt(br.readLine());
count++;
fr.close();
```

```
        w = new FileWriter(c);
        w.write(""+count);
        w.close();

        File rg = new File("register.txt");
        FileWriter wr = new FileWriter(rg,true);
        wr.append(r.v.regNo+eol);
        wr.flush();
        }
      catch(IOException e)
      {
      System.out.println("\n File handling error.");
      }
  }
```

Entry

1. Enter the registration number of a banned vehicle.
2. Message should appear to inform that vehicle is banned from entry.
3. Enter the registration number of a non-registered vehicle.
4. Details for the vehicles will be asked.
5. Enter the registration number of a registered vehicle.
6. Vehicle is allowed to enter. Gate opens.
7. Go to Query and check status. Status should be ‘vehicle is inside’.

Code snippets:

```
boolean isAllowed(String registrationNumber)
  {
     Register r = new Register();
```

```
if(r.checkRegistered(registrationNumber)==true)
{
   currentRecord cR = new currentRecord();
   cR.registered=true;
   cR.regNo=registrationNumber;
   cR.update();
   updateStatus(registrationNumber,true);
   Gate.directGate(true);
   return (true);
}
else {
   Ban b = new Ban();
   if(b.checkBan(registrationNumber)==true)
   {
      updateStatus(registrationNumber,false);
      Gate.directGate(false);
      return (false);
   }
   else
   {
      createNew(registrationNumber);
      return (true);
   }
   }
```

```
    }

    void createNew(String regNo)
    {
        currentRecord r = new currentRecord();
        String[] str ={regNo};
        r.registered = false;
        r.main(str);
    }
    void updateStatus(String regNo, boolean status)
    {
        File f = new File(regNo+"status.txt");
        try
        {
            FileWriter w = new FileWriter(f);
            w.write(""+status);
            w.close();
        }
        catch(IOException e)
        {
            System.out.println("Error while updating status.");
        }
    }
}
```

Test code snippets for entry (using JUnit 4):

```
public void testIsAllowed() {
        System.out.println("isAllowed");
        String registrationNumber = "MH12S1100";
        Entry instance = new Entry();
        boolean expResult = false;
        boolean result = instance.isAllowed(registrationNumber);
        assertEquals(expResult, result);
}
public void testCreateNew() {
        System.out.println("createNew");
        String regNo = "QW11A1111";
        Entry instance = new Entry();
        instance.createNew(regNo);


public void testUpdateStatus() {
        System.out.println("updateStatus");
        String regNo = "AS11A1111";
        boolean status = false;
        Entry instance = new Entry();
        instance.updateStatus(regNo, status);
        File f = new File("AS11A1111status.txt");
        try{
```

```
    BufferedReader br = new BufferedReader(new FileReader (f));
    if(br.readLine().equals("false"))
    {
       System.out.println("successful");


    }
 else{
       System.out.println("unsuccessful");
 }
    }
    catch(IOException e)
    {
    }
  }
}
```

Exit

1. Enter the registration number of a vehicle that has a complaint or is delayed (in case it is not registered).
2. You will be informed about the complaint.
3. Enter the possible reason and the approval of the security officer.
4. If the reason is declared valid by the officer, vehicle is allowed to exit.
5. If the reason is not satisfactory,
   i. Vehicle is registered : Message: letter is issued to the dean of campus affairs
   ii. Vehicle is not registered: Message: Vehicle is banned from further entry.
6. Go to Query and check status of the vehicle. It should be that ‘vehicle is outside’ and the time and date of the last exit should also be mentioned.

Code snippets:

```
boolean isAllowed()
  {
     boolean complaintPresent;
     Complaint c = new Complaint();
     complaintPresent = c.checkComplaint(regNo);
     c.regNo = regNo;
     if(complaintPresent == true)
     {
        c.getComplaint();
        JOptionPane.showMessageDialog(null,"Your vehicle has the following complaint.\n"+c.description+"\n Enter the explanation.");
        takeSteps();
        update();
        return(true);
     }
     else {
        Register r = new Register();
        registered = r.checkRegistered(regNo);
        if(registered == true)
        {
           update();
           updateStatus(regNo, false);
           Gate.directGate(true);
```

```
            return (true);
        }
        else
        {
            long d = Time.getDuration(regNo);
            if(d>=8)
            {
                JOptionPane.showMessageDialog(null,"Your vehicle is delayed.\n Enter the explanation.");
                takeSteps();
                update();
                return (true);
            }
             else {
                 update();
                 updateStatus(regNo, false);
                 Gate.directGate(true);
                 return (true);
             }
        }
    }
}
void takeSteps()
{
     Complaint c = new Complaint();
```

```
    c.regNo = regNo;
    String[] str = {regNo};
    ReasonsInterface.main(str);
}

void update()
{
     Register r = new Register();
    registered = r.checkRegistered(regNo);
    String eol = System.getProperty("line.separator");
    t = new Time();

    try
    {
       File f = new File("exitRecord.txt");
       FileWriter w = new FileWriter(f,true);
       w.append(regNo+eol+registered+eol);
       w.append(""+t.getTime()+eol+t.milliTime()+eol);
       w.flush();
       w.close();
    }
    catch(IOException e)
    {
       System.out.println("\n File handling error in current exit record.\n");
```

```
        }
    }
      void updateStatus(String regNo, boolean status)
    {
       File f = new File(regNo+"status.txt");
       try
       {
          FileWriter w = new FileWriter(f);
          w.write(""+status);
          w.close();
       }
       catch(IOException e)
       {
          System.out.println("Error while updating status.");
       }
    }
}
```

Code snippets for banning a vehicle or checking whether a vehicle is banned or not:

```
void update(String reg)
       {
          if(checkBan(reg)==true)
          {
             JOptionPane.showMessageDialog(null,"Already banned.");
          }
```

```
else
        {
        updateCount();
        String eol = System.getProperty("line.separator");
        try
        {
        FileWriter w = new FileWriter(new File("banned.txt"),true);
        w.append(reg+eol);
        w.flush();
        w.close();
        }
        catch(IOException e)
        {
           System.out.println("File writing error while banning.");
        }
        }
    }
  boolean checkBan(String reg)
  {
     boolean result = false;
    try
        {
        FileReader w = new FileReader(new File("banned.txt"));
        BufferedReader br = new BufferedReader(w);
```

```
        regNo = br.readLine();
        getCount();
        int i = 1;
        while(i<=count)
        {
           if(regNo.equals(reg))
           {
              result = true;
              return(result);
           }
           regNo=br.readLine();
           i++;
        }
        br.close();
        }
        catch(IOException e)
        {
           System.out.println("File handling error while checking ban.");
        }
     return (result);
}
void updateCount()
{
   try
```

```
      {
         File c = new File("banCount.txt");
         FileReader fr = new FileReader(c);
         BufferedReader br = new BufferedReader(fr);
         count = Integer.parseInt(br.readLine());
         count++;
         fr.close();
         FileWriter w = new FileWriter(c);
         w.write(""+count);
         w.close();
      }
      catch(IOException e)
      {
         System.out.println("\n Error while updating banned vehicle count.");
      }
   }
int getCount()
   {
      try{
         File c = new File("banCount.txt");
         FileReader fr = new FileReader(c);
         BufferedReader br = new BufferedReader(fr);
         count = Integer.parseInt(br.readLine());
         fr.close();
```

```
    }
    catch(IOException e)
    {
       System.out.println("File Handling Error.");
    }
    return count;
  }
```

Complaint

1. On the welcome page, click on Complaint box.
2. Enter an invalid registration number.
3. Message appears: Invalid registration number.
4. Enter a valid registration number (of the errant vehicle).
5. Enter the description of the complaint.
6. Click on complaint button.
7. Now go back to the welcome window and try to exit the vehicle with that particular registration number.
8. Complaint description should appear.

Code snippets:

```
private void complaintBActionPerformed(java.awt.event.ActionEvent evt) {
        regNo=regNoF.getText();
        description=complaintF.getText();
        if(!regNo.matches("[A-Z][A-Z]\\d\\d[A-Z]\\d\\d\\d\\d"))
           {
           JOptionPane.showMessageDialog(null, "Invalid registration number. Please check.");
           }
        else recordComplaint();    }
 void getComplaint()
```

```
    {
       try
       {
       File f = new File(regNo+"Com.txt");
       BufferedReader br = new BufferedReader(new FileReader(f));
       description = br.readLine();
       br.close();
       }
       catch(IOException e)
       {
          System.out.println("Error while getting complaint from file.");
       }
    }
void recordComplaint()
    {
    String eol = System.getProperty("line.separator");
    try
    {
       File f=new File("complaint.txt");
       FileWriter w = new FileWriter(f,true);
       w.append(regNo+eol);
       w.flush();
       w.close();
       f=new File(regNo+"Com.txt");
```

```
    w = new FileWriter(f);
    w.append(description);
    w.flush();
    w.close();

    File c = new File("complaintCount.txt");
    FileReader fr = new FileReader(c);
    BufferedReader br = new BufferedReader(fr);
    int count = Integer.parseInt(br.readLine());
    count++;

    fr.close();
    w = new FileWriter(c);
    w.write(""+count);
    w.close();
}
catch(IOException e)
{
    System.out.println("File handling error.");
}
}
boolean checkComplaint(String x)
{
    try{
```

```
File f = new File("complaint.txt");
FileReader fr = new FileReader(f);
BufferedReader br =  new BufferedReader(fr);
String number = br.readLine();
int i=1;

BufferedReader br2 = new BufferedReader(new FileReader(new File("complaintCount.txt")));
int count = Integer.parseInt(br2.readLine());
br2.close();

while(i<=count)
{
   if(number.equals(x))
      return (true);
   number=br.readLine();
   i++;
}
return (false);
}
catch(IOException e)
{
   System.out.println("File Handling error while checking complaint.");
}
return false;
```

```
}

void removeComplaint()
{
   String eol = System.getProperty("line.separator");
   try
   {
   File f2 = new File("complaintCount.txt");
   BufferedReader br2 = new BufferedReader(new FileReader(f2));
   int count = Integer.parseInt(br2.readLine());
   br2.close();

   if(count == 0) return;

   File f = new File("complaint.txt");
   String[] s = new String[count];
   BufferedReader br = new BufferedReader(new FileReader(f));
   int i = 0;
   String x=br.readLine();
   int count2=count;

   while(i<count)
   {
      if(x.equals(regNo))
```

```
    {
        count2--;
        break;
    }
     else
    {
        s[i]=x;
        i++;
        x=br.readLine();
     }
}
count = count2;
br.close();

FileWriter w = new FileWriter(f);

for(i=0;i<count;i++)
{
w.append(s[i]+eol);
}
w.flush();
w.close();

w = new FileWriter(f2);
```

```
        w.write(""+count);
        w.close();
        }
        catch(IOException e)
        {
           System.out.println("Error while removing complaint.");
        }
    }
```

Code snippets for testing the Complaint methods (using JUnit 4)

```
public void testGetComplaint() {
        System.out.println("getComplaint");

        Complaint instance = new Complaint();
        instance.regNo ="QW12Q1111";
        instance.getComplaint();
        assertEquals(instance.description,"accident");
    }
public void testRecordComplaint() {
        System.out.println("recordComplaint");
        Complaint instance = new Complaint();
        instance.regNo="WB12S1001";
        instance.description="rash driving";
        instance.recordComplaint();
        File f = new File("WB12S1001Com.txt");
```

```
    if(f.exists())
    {
       System.out.println("\n record complaint successful.");
    }
else{
      System.out.println("\n record complaint not successful.");
}   }
  public void testCheckComplaint() {
    System.out.println("checkComplaint");
    String x = "";
    Complaint instance = new Complaint();
    boolean expResult = false;
    boolean result = instance.checkComplaint(x);
    System.out.println("The test result of CheckComplaint:");
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
  }
  public void testRemoveComplaint() {
    System.out.println("removeComplaint");
    Complaint instance = new Complaint();

    instance.regNo="WB12S1001";
    instance.description="test";
```

```
instance.recordComplaint();

instance.removeComplaint();

if(instance.checkComplaint("WB12S1001")==false)

{

   System.out.println("test successful");

}

else {

   System.out.println("test failure");

 }  }
```

Query

1. Enter the registration number and click on show details or check status (already done while testing other cases).
2. Click on the various other buttons to get the number of registered vehicles, number of vehicles that has entered/exited the campus in the last 24 hours.

Code snippets:

```
private void detailsBActionPerformed(java.awt.event.ActionEvent evt) {

    regNo = regNoF.getText();

    File f = new File(regNo+".txt");

    if(f.exists())

    {

       Register r = new Register();

       try{

         BufferedReader br = new BufferedReader(new FileReader(f));

         String s="";

         br.readLine();
```

```
 if(r.checkRegistered(regNo)==true)
 {
   s=s.concat("Vehicle is Registered.\n");
   s=s.concat("Type: "+br.readLine()+"\n");
   s=s.concat("model: "+br.readLine()+"\n");
   s=s.concat("colour: "+br.readLine()+"\n");
   s=s.concat("Owner's name: "+br.readLine()+"\n");
   s=s.concat("Owner's designation: "+br.readLine()+"\n");
   s=s.concat("Owner's address: "+br.readLine()+"\n");
   resultF.setText(s);
 }
else{
   s=s.concat("Vehicle is not registered.\n");
   s=s.concat("Type: "+br.readLine()+"\n");
   s=s.concat("model: "+br.readLine()+"\n");
   s=s.concat("colour: "+br.readLine()+"\n");
   s=s.concat("Owner's name: "+br.readLine()+"\n");
   s=s.concat("Owner's designation: "+br.readLine()+"\n");
   s=s.concat("Owner's address: "+br.readLine()+"\n");
   s=s.concat("Purpose of visit: "+br.readLine()+"\n");
   s=s.concat("Duration: "+br.readLine()+"\n");
   resultF.setText(s);
 }
}
```

```
        catch(IOException e)
        {
            System.out.println("Error which querying about details.");
        }
    }
     else{
            resultF.setText("No vehicle with the given registration number exists.");
     }
}

private void regCountBActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try{
        File f = new File("registeredCount.txt");
        BufferedReader br = new BufferedReader(new FileReader(f));
        resultF.setText("Number of registered vehicles = "+br.readLine());
    }
    catch(IOException e)
    {
        System.out.println("File handling error while obtaining registered count.");
    }
}
```

```
private void statusBActionPerformed(java.awt.event.ActionEvent evt) {
    String date="";
    regNo = regNoF.getText();
    File f = new File(regNo+"status.txt");
    if(f.exists()==true){
    try
    {
    BufferedReader br = new BufferedReader(new FileReader(f));
    String status = br.readLine();
    if(status.equals("true"))
    {
       resultF.setText("Vehicle is inside.");
    }
     else if(status.equals("false"))
     {
        String s="";
       s=s.concat("Vehicle is outside.");
       File f2 = new File("ExitRecord.txt");
       BufferedReader br2 = new BufferedReader(new FileReader(f2));
       String x = br2.readLine();
       while(x!=null)
       {
          if(x.equals(regNo))
          {
```

```
            br2.readLine();
            date = "\n [Date Time] of last exit = ["+br2.readLine()+"]";
            br2.readLine();
            x = br2.readLine();
        }
        else{
            br2.readLine();
            br2.readLine();
            br2.readLine();
            x = br2.readLine();
        }
      }
      s = s.concat(date);
      resultF.setText(s);
   }
   }
   catch(IOException e)
   {
      System.out.println("File handling error while getting count.");
   }}
   else {
      resultF.setText("No data available.");
   }
}
```

```
private void entryActionPerformed(java.awt.event.ActionEvent evt) {
   long t,count =0;
   long c =Time.milliTime();
   File f2 = new File("record.txt");
   try{
      BufferedReader br2 = new BufferedReader(new FileReader(f2));
      String x = br2.readLine();
      while(x!=null)
      {
            br2.readLine();
            br2.readLine();
            t = Long.parseLong(br2.readLine());
            if((c-t)/(24*3600000)<24) count++;
            x = br2.readLine();
      }
      resultF.setText("Number of vehicles that has entered the Campus in the past 24 hours = "+count);
   }
      catch(IOException e)
       {
       System.out.println("query error");
      }
}
private void exitActionPerformed(java.awt.event.ActionEvent evt) {
   long t,count =0;
```

```
    long c =Time.milliTime();
    File f2 = new File("exitRecord.txt");
    try{
       BufferedReader br2 = new BufferedReader(new FileReader(f2));
       String x = br2.readLine();
       while(x!=null)
       {
            br2.readLine();
            br2.readLine();
            t = Long.parseLong(br2.readLine());
            if((c-t)/(24*3600000)<24) count++;
            x = br2.readLine();
       }
       resultF.setText("Number of vehicles gone out of the Campus in the past 24 hours = "+count);
    }
       catch(IOException e)
       {
          System.out.println("query error");
       }
}
```

Software Project Management Process (SPMP) Document

for

IIT Security System

(IITSS)

Version 1.0

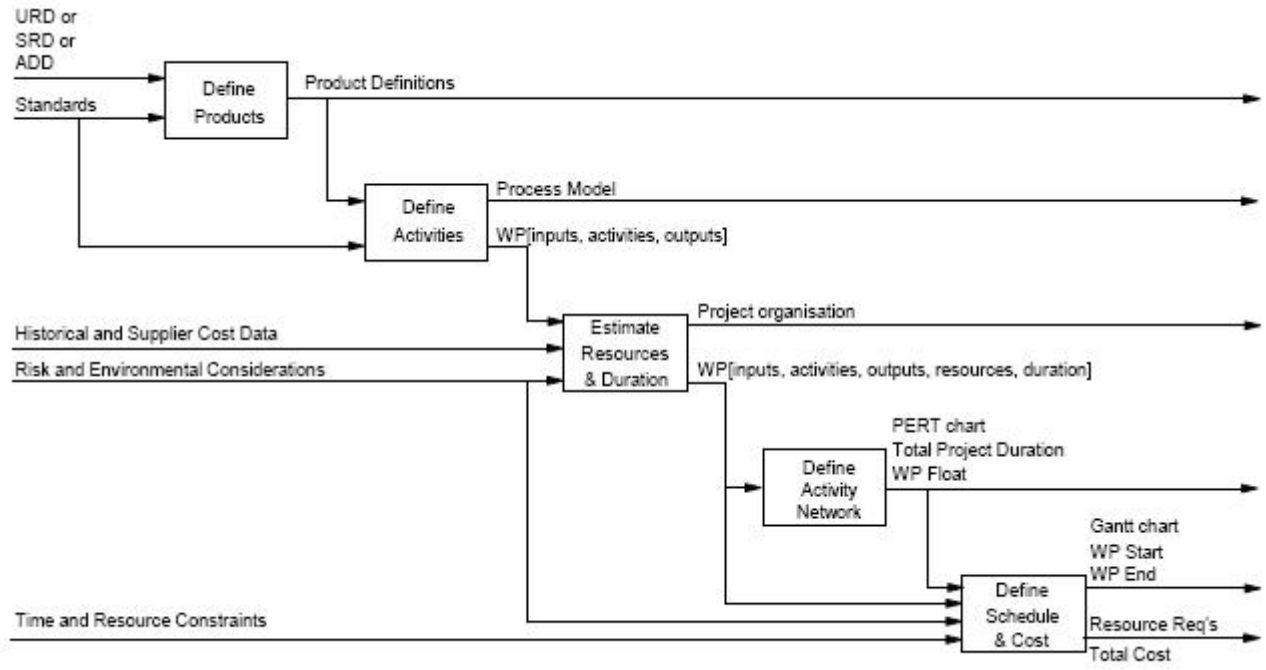
18th April, 2011

Introduction

Project Management provides an integrated framework for project organization, planning and control which is designed to:

- ensure the timely and cost-effective production of all the end-products
- maintain acceptable standards of quality
- fulfill the requirements of the enterprise which has invested in the project

Project planning chart



Project Estimates

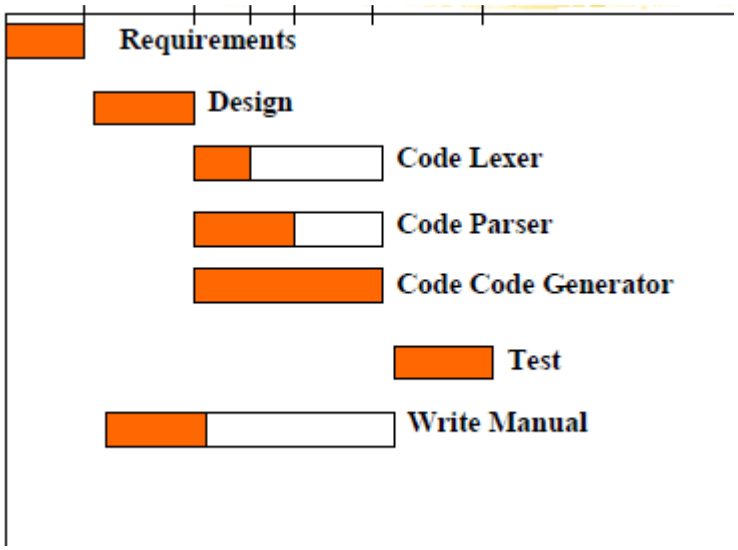
Function	Efforts required by a single person
SRS documentation	2 weeks
DFD preparation	1 week
Function oriented design	1 week
Object oriented analysis and designing	3 weeks
Coding	1 week
Testing	1 week

Project Resources Plan

The resources necessary to carry out the project can be tabulated as follows.

Persons involved (human resources)	Role	Software Requirements	End product
Requirement specifier	prepares the SRS document	MS Word + pdf converter	SRS
Data flow model developer	makes the Data flow diagram	SmartDraw + MS Word + pdf converter	DFD
Function oriented model designer	makes the top level architecture design and the function oriented design	SmartDraw + MS Word + pdf converter	FOD (detailed design)
Object oriented designer	Makes the use case diagram, class diagram and sequence/ collaboration diagrams.	MagicDraw + MS Word + pdf converter	OOAD (Detailed design)
Coder	does the coding of the software using netbeans	Netbeans IDE 6.9.1, Notepad	Source Code
Tester	Tests that the software is giving correct result as expected from the SRS	Netbeans IDE 6.9.1 with JUnit 4 installed	Correct source code
Project Manager	Manages the work of the above people and maintains coordination between the development team members	Task manager	Complete Software(ready for use)

Gantt Chart



Schedule

Item	To be Completed by
Software Project Management Plan (this document)	April 20, 2011
Software Requirements Specifications	March 6, 2011
Data Flow diagram	March 6, 2011
Function oriented design	March 15, 2011
Object oriented design (detailed design)	April 5, 2011
Source Code	April 17, 2011
Testing	April 17, 2011
Updating all the above documents	April 20, 2011

Risk Management Plan

1. Deficiency in the knowledge and understanding of the problem and its solution: This indicates that the developer does not have the complete understanding of the problem. This will affect the quality of the project in terms of requirements of the product and their fulfillment, which is not desirable. Building a prototype for the project model and doing an extensive literature search can overcome this. This will help the developer in delivering an efficient and quality product.

2. "Lack of Skills and knowledge of tools needed for statistical analysis", which means that the developer does not have knowledge about the tools and knowledge of working on statistical analysis. In this case, the developer is expected to update his / her knowledge of tools available for this purpose and decide the one that will be used in the project and master it.

-End of IITSS Document-