COMP-533 Object-Oriented Software Development

Final

407 Express Toll Route Design

(40% of final grade)

November 13, 2012

Problem Statement

The 407 Express Toll Route is a highway that runs east-west just north of Toronto, and was one of the largest road construction projects in the history of Canada. The road uses a highly modern Electronic Toll Collection (ETC) system constructed by Raytheon.

The ETR technology allows motorists to pass through toll routes without stopping or even opening a window. To make this happen, each highway entry and exit point is equipped with a *gantry* (see figure 1).

Processing Registered Vehicles

The most cost-efficient way to pay for highway use is to open an account with the 407 ETR system. Accounts can be personal or linked to a company. In either case, billing information (name and address) is saved with the account. Once an account is created, vehicles can be registered with it. Registered vehicles require a small electronic tag, called a transponder (see figure 2), to be attached to the windshield behind the rear-view mirror. Transponders are leased for a small monthly fee. The registration includes the vehicle details.

The system automatically records the entry and exit of vehicles, and creates a transaction for each trip. This is done in the following way. When the vehicle enters the highway, it passes under the overhead gantry. The hardware devices of a gantry are shown in figure 3.

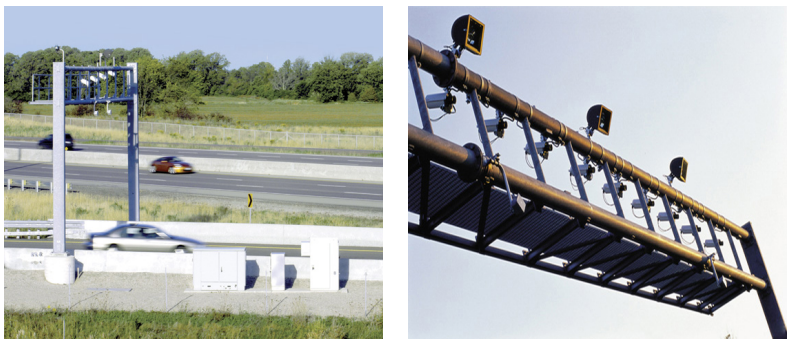

Figure 1: The 407 Entry / Exit Gantries

Figure 2: A Transponder

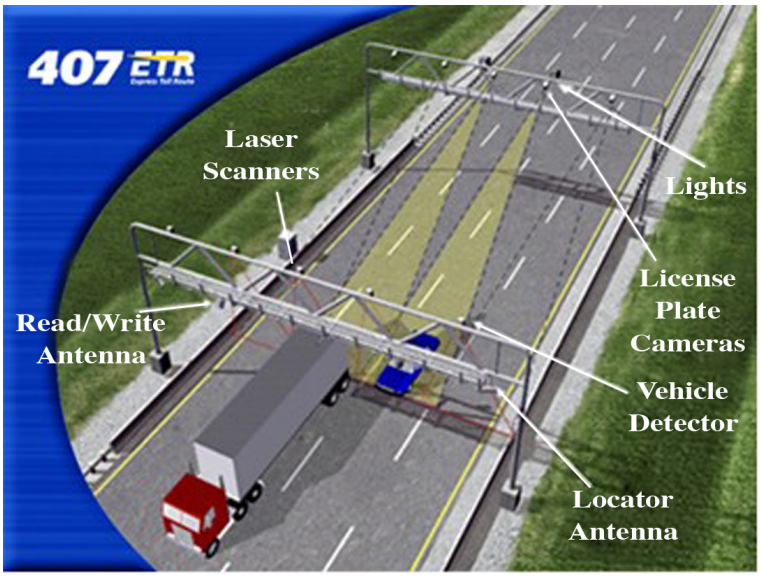

Figure 3: An Entry / Exit Gantry

The locator antennae determine if the vehicle is equipped with a transponder. Next, the read / write antennae read the account number from the transponder and the point of entry, time and date is recorded.

In addition, as a vehicle passes under the gantries, the system uses laser scanners to determine the class of vehicle (e.g. light vehicle, heavy single unit vehicle, heavy multiple unit vehicle). It does this by measuring the height, width and depth of each approaching vehicle. A check is made to verify that the class of vehicle corresponds to the one registered for this particular transponder.

The same process occurs when the vehicle exits the highway. The entry and exit data are then matched and the transponder account holder is debited. When the route is exited, the transponder gives a green signal followed by four short beeps to indicate a successful completion of the transaction.

Processing Unregistered Vehicles

Transponders are mandatory for heavy vehicles, i.e. vehicles with a gross weight of 5,000 kg. However, light vehicles can use the 407 ETR without registering.

When a motorist without a transponder enters the highway and passes under the two tolling gantries, the system triggers a set of digital cameras to take pictures of the rear number plate of the vehicle from different angles. In order to get good images, a set of lights are turned on before the images are taken. The lights automatically adjust their intensity to ensure the best conditions for taking an image of the number plate. At the same time, the laser scanners are activated to classify the vehicle and tell the toll collection system whether to charge for a passenger or commercial vehicle. The owner of the vehicle is identified by electronic access to government records.

The cameras and lights have been tested to ensure accuracy, even in blizzards and rainstorms. However, if the video correlation and image processing fails to determine the license plate with sufficient probability, a human operator has to look at the pictures to make the call.

Payment

Registered car owners, registered companies, and motorist that used the highway, receive an invoice in the mail at the end of the month containing the monthly rental fee for the registered transponders ($3 per transponder), if any, and the trips of all of the vehicles registered with their accounts. The price of each trip is calculated based on the time of day, distance traveled and type of vehicle. If the entry or exit time falls within peak hours (6am - 10am, 3pm - 7pm), the toll rate is 16.25 cents/km for light vehicles, 32.5 cents/km for heavy vehicles, and 48.75 cents/km for heavy multiple unit vehicles. Otherwise, the toll rate is 15.5 cents/km for light vehicles, 31 cents/km for heavy vehicles and 46.5 cents/km for heavy multiple unit vehicles.

If a light vehicle uses the highway without a transponder, an additional video toll charge of $3.50 is applied per trip. Cheating motorists (for instance, motorists using a transponder with an unregistered vehicle, or heavy vehicles taking the highway without a transponder) are fined with $50.

Registered drivers can sign up for auto-payment by credit card, or simply pay each month by cheque. Refusal to pay invoices for 3 months results in plate denial, meaning that a debtor cannot renew the license plate of his cars or obtain a new license plate from the government until all tolls and fees have been paid.

Hardware / Software Decisions

The software to be developed has to interface with the hardware devices of the gantry. The development of the software running on the transponders is going to be outsourced to a different company, and hence does not have to be considered for this assignment.

Use Case Model

The use case model of the 407 ETR has already been established. The use cases and their relationships are summarized in figure 4. The individual descriptions of each use case are shown below.

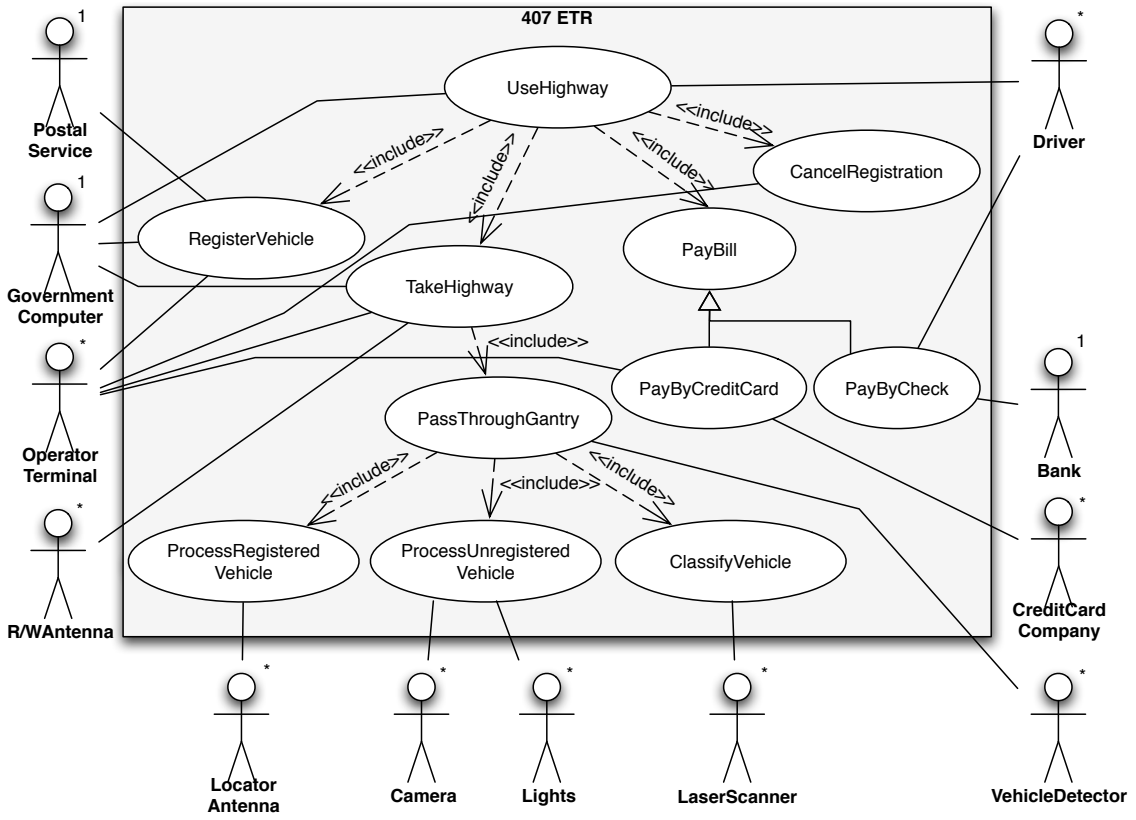

Figure 4: 407 ETR Use Case Diagram

UseHighway Use Case

Use Case: UseHighway
Scope: 407 ETR System
Level: Summary
Intention in Context: The intention of the Driver is to use the 407 ETR highway on a regular basis.
Multiplicity: One Driver can only drive one vehicle at a time on the highway. However, different Drivers can use the highway simultaneously.
Primary Actor: Driver
Secondary Actor: GovernmentComputer
Main Success Scenario:
1. Driver registers vehicle.
   Step 2-4 are repeated once a month as long as the vehicle is registered.
2. Driver takes highway.
   Step 2 can be repeated any number of times per month.
3. At the end of the month, System sends bill to Driver.
4. Driver pays bill.
5. Driver cancels registration.

Extensions:
1a. Driver uses highway without registering vehicle. Use case continues at step 2.
4a. Driver does not pay bill for 3 consecutive months.
4a.1. System informs GovernmentComputer of refusal to pay the bill. Use case continues at step 2.

RegisterVehicle Use Case

Use Case: RegisterVehicle
Scope: 407 ETR System

Level: User Goal

Intention in Context: The goal of the Driver is to register a vehicle with the system, which involves opening an account and linking a transponder to it.

Multiplicity: A driver registers his vehicles one at a time. However, the system should be able to handle multiple simultaneous registrations done by different drivers.

Primary Actor: Driver

Secondary Actor: OperatorTerminal, GovernmentComputer, PostalService

Main Success Scenario:

The Driver interacts with the System by calling an Operator.

1. Driver provides System with personal data and vehicle information.
2. System sends vehicle information to GovernmentComputer for verification.
3. GovernmentComputer notifies System that vehicle information is correct.
4. System acknowledges registration to the Driver.
5. System assigns a new transponder to the vehicle, and informs Postal Service to deliver the transponder to the Driver.
6. Driver installs and tests transponder.
7. Driver notifies the System of successful installation of the transponder.

Extensions:

3a. Exception{GovernmentComputerUnavailable}. Use case ends in failure.

4a. Driver does not have an account with the system yet. System opens a new account for the driver. Use case continues at step 4.

6a. Transponder installation and testing fails. Driver notifies System of the problem. Use case continues at step 5.

7a. Driver forgets to acknowledge installation and simply starts using the transponder on the highway. Use case ends in success.

TakeHighway Use Case

Use Case: TakeHighway

Scope: 407 ETR System

Level: User Goal

Intention in Context: The intention of the Driver is to drive a vehicle from one location to another by taking the 407 ETR highway.

Multiplicity: One Driver can only drive one vehicle at a time on the highway. However, different Drivers can take the highway simultaneously.

Primary Actor: Driver

Secondary Actor: RWAntenna, GovernmentComputer, OperatorTerminal

Main Success Scenario:

1. Driver enters highway, passing through gantry.
2. Driver exits highway, passing through gantry.
3. System retrieves the driver's vehicle record based on trip information*.
4. System determines the amount owed based on the trip information and adds the transaction to the vehicle's records.
5. System informs Driver by sending a signal to the RWAntenna of successful completion of transaction.

Extensions:

3a. Vehicle is unregistered and does not have a record yet.

3a.1. System sends licence plate information to GovernmentComputer.

3a.2. GovernmentComputer sends vehicle information and owner's address to System.

3a.2a. Exception{GovernmentComputerUnavailable}: use case ends in failure.

3a.3. System creates a new vehicle record. Use case continues at step 4.

3b. Vehicle is unregistered and licence plate is unrecognizeable.

3b.1. System displays pictures on OperatorTerminal.

3b.2. OperatorTerminal sends licence plate information to System. Use case continues at step 3.

3b.2a. Exception{OperatorTerminalFailure}: use case ends in failure.

4a. Exit unsuccessful.

4a.1a. If entry was successful, minimum trip charge is added to vehicle's records. Use case ends in success.

4a.1b. If entry was unsuccessful as well, use case ends in failure.

4b. Entry unsuccessful.
4b.1a. If exit was successful, minimum trip charge is added to vehicle's records. Use case continues at step 4.
5a. Vehicle is not registered. Use case ends in success.
5b. Exception{RWAntennaFailure}: use case ends in success.

*__Trip Information Details__: A complete trip information record includes entry and exit time and place, measured and obtained vehicle classification, transponder account or licence plate information or licence plate images.

PassThroughGantry Use Case

Use Case: PassThroughGantry
Scope: 407 ETR System
Level: Sub-Function
Intention in Context: The Driver passes through a entry or exit gantry as part of his trip.
Multiplicity: One Driver can only drive one vehicle at a time through a gantry. However, different Drivers can pass through the same or different gantries simultaneously.
Primary Actor: Driver
Main Success Scenario:
1. System processes registered vehicle.
2. System classifies vehicle.
   Steps 1 and 2 can be performed in any order or in parallel.
3. System records entry time and vehicle information for the trip.

Extensions:
2a. Processing registered vehicle fails.
2a.1. System processes unregistered vehicle. Use case continues at step 2.
4a. Both processing were unsuccessful. Use case ends in failure.
4b. Classification was unsuccessful. Use case ends in success.

ProcessRegisteredVehicle Use Case

Use Case: ProcessRegisteredVehicle
Scope: 407 ETR System
Level: Sub-Function
Intention in Context: The System communicates with the transponder to identify the approaching vehicle.
Multiplicity: The System must be able to process multiple registered vehicles simultaneously.
Primary Actor: N/A
Secondary Actor: LocatorAntenna, R/WAntenna
Main Success Scenario:
1. LocatorAntenna notifies System that it detected an approaching vehicle with transponder.
2. System asks R/WAntenna to obtain account information from transponder.
3. RWAntenna informs System of account information.
4. System records account information for the trip.
5. System turns on the Lights.
6. System triggers the Cameras.
7. Cameras send images to System.
8. System determines licence plate information based on images.

Extensions:
1a. The approaching vehicle does not have a transponder. Use case ends in failure.
1b. Exception{LocatorAntennaFailure}: use case ends in failure.
(2-3)a. Exception{RWAntennaFailure}: use case ends in failure.
3a. R/WAntenna is unable to obtain account information. Use case ends in failure.
6a. Exception{LightFailure}: use case continues at step 6.
7a. Exception{CameraFailure}: use case ends in success.

ProcessUnregisteredVehicle Use Case

Use Case: ProcessUnregisteredVehicle
Scope: 407 ETR System
Level: Sub-Function
Intention in Context: The System wants to identify the approaching vehicle using the licence plate information.
Multiplicity: The System must be able to process multiple unregistered vehicles simultaneously.
Primary Actor: N/A
Secondary Actors: VehicleDetector, Cameras, Lights
Main Success Scenario:
1. VehicleDetector notifies System that a vehicle is approaching.
2. System turns on the Lights.
3. System triggers the Cameras.
4. Cameras send images to System.
5. System determines licence plate based on images.

Extensions:
3a. Exception{LightFailure} Use case continues at step 2.
4a. Exception{CameraFailure} Use case ends in failure.
5a. Licence plate recognition fails.
5a.1. System displays pictures on an OperatorTerminal.
5a.2. Operator recognizes picture and sends licence plate information to System.

ClassifyVehicle Use Case

Use Case: ClassifyVehicle
Scope: 407 ETR System
Level: Sub-Function
Intention in Context: The System wants to classify the approaching vehicle as light vehicle, heavy single unit vehicle, or heavy multiple unit vehicle.
Multiplicity: The System must be able to classify multiple vehicles simultaneously.
Primary Actor: N/A
Secondary Actor: LaserScanner
Main Success Scenario:
1. System activates LaserScanner.
2. LaserScanner sends vehicle dimensions to System.
3. System classifies vehicle and records classification in trip information.

Extensions:
2a. Exception{LaserScannerFailure} System records classification failure in trip information. Use case ends in failure.

PayByCreditCard Use Case

Use Case: PayByCreditCard
Scope: 407 ETR System
Level: User Goal
Intention in Context: The goal of the Driver is pay for his trip by credit card.
Multiplicity: Every driver pays for his trips once a month. The system must support concurrent payments of different drivers, be it by credit card or by check.
Primary Actor: Driver
Secondary Actor: OperatorTerminal, CreditCardCompany
Main Success Scenario:
Driver interacts with System by calling an Operator.
1. Operator provides System with credit card information.
2. System contacts CreditCardCompany to validate credit.
3. CreditCardCompany notifies System of successful validation.

4. System notifies Operator of success.
Extensions:
2a. Exception{CreditCardCompanyUnavailable}: System notifies Operator. Use case ends in failure.
3a. Exception{InsufficientCredit}: System notifies Operator. Use case ends in failure.

PayByCheck Use Case

Use Case: PayByCheck
Scope: 407 ETR System
Level: User Goal
Intention in Context: The goal of the Driver is pay for his trip by check.
Multiplicity: Every driver pays for his trips once a month. The system must support concurrent payments of different drivers, be it by credit card or by check.
Primary Actor: Driver
Secondary Actor: OperatorTerminal
Main Success Scenario:
Driver sends check to Operator.
1. Operator notifies System that check has been received.
Operator cashes check with Bank.
2. Bank notifies System that check has been cleared.
Extensions:
2a. Exception{BouncedCheck}: System notifies Driver. Use case ends in failure.

CancelRegistration Use Case

Use Case: CancelRegistration
Scope: 407 ETR System
Level: User Goal
Intention in Context: The goal of the Driver is to unregister a vehicle and potentially cancel his account with the 407 ETR system.
Multiplicity: A driver unregisters a vehicle one at a time. The system should be able to handle multiple concurrent unregistrations of different drivers.
Primary Actor: Driver
Secondary Actor: OperatorTerminal
Main Success Scenario:
Driver interacts with System by calling an Operator.
1. Operator notifies System that Driver wants to cancel his registration for a vehicle.
2. System marks vehicle registration as suspended and does not charge monthly fees anymore.
Driver sends transponder to Operator.
3. Operator notifies System that transponder has been received.
4. System cancels vehicle registration.
5. If Driver has no vehicles registered with the system anymore, System cancels driver account.
Extensions:
3a. Exception{TransponderUsed}: System reactivates registration. Use case ends in failure.

Architectural Decisions

After an initial analysis, the developing team determined that it would be very inefficient and unreliable to have one central computer handle the hardware of each gantry. They decided to assign a *gantry controller device* to each gantry. The device coordinates the LocatorAntenna, R/WAntenna, VehicleDetector, LaserScanner, Camera and Lights, collects all information for a passing vehicle, and then sends the completed trip information to the backend system.

As a result, the system development is now split into two systems: the *gantry controller system* and the *407 backend system*. You have been assigned to the team that develops the backend. An environment and concept model for the backend system are shown in figure 5 and figure 6.

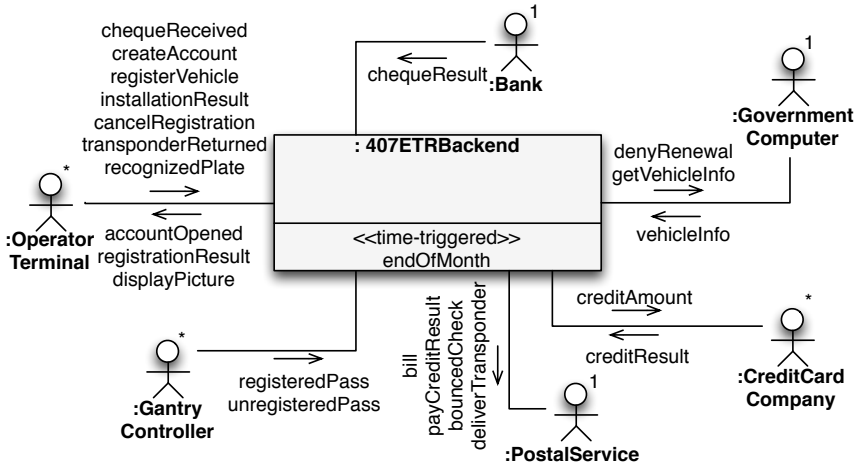

Figure 5: 407 ETR Backend Environment Model

Type Definitions

The environment model and concept model assume the existence of the following types:

- `Picture`: a type encoding a series of pictures taken by the licence plate cameras.
- `CreditCardNumber`: a type encoding a credit card number.
- `ChequeNumber`: a type enconding a cheque number.
- `LicencePlate`: a type encoding a licence plate number.
- `Time`: a type encoding time.
- `Location`: a type encoding a location on the highway.
- `type VehicleClass is enum {light, heavy_single, heavy_multiple, undetermined}`
- `type GantryKind is enum {entry, exit}`
- `type TransactionKind is enum {monthlyFee, registeredTrip, unregisteredTrip, fine}`
- `type Transaction is TupleType {kind: TransactionKind, description: String, amount: Integer}`

Input Messages

- `registeredPass(t: Transponder, when: Time, class: VehicleClass, pict: Picture)`
- `unregisteredPass(when: Time, class: VehicleClass, pict: Picture)`
- `vehicleInfo(lp: LicencePlate, class: VehicleClass, ownerName: String, ownerAddress: String)`: sent by *GovernmentComputer* to transmit vehicle information to System.
- `creditResult(c: CreditCardNumber, amount: Integer, outcome: Boolean)`: sent by *CreditCardCompany* to nofity System of credit request result. You can assume that the amount is equal to the amount that you requested.

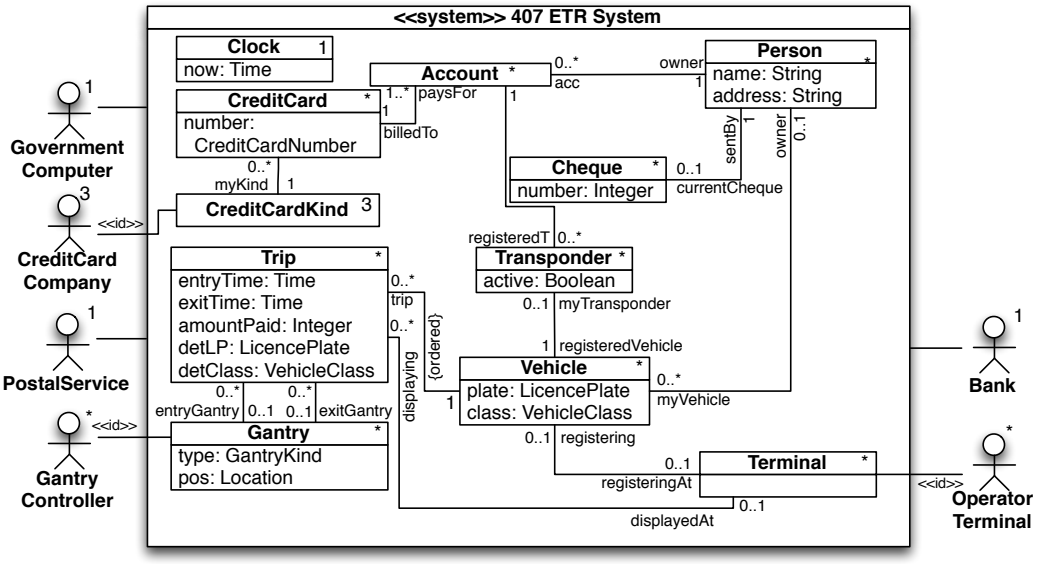

Figure 6: 407 ETR Backend Concept Model

- `chequeReceived(p: Person, cheque: ChequeNumber)`: sent by *Operator* when he receives a check from an unregistered driver.
- `chequeResult(cheque: ChequeNumber, amount: Integer, outcome: Boolean)`: sent by *Bank* when a cheque clears or is rejected. The value of amount does not necessarily cover all outstanding trips.
- `createAccount(name: String, address: String, cardnumber: CreditCardNumber)`: sent by *Operator* to create a new account for a person identified by a name and an address, and register for auto-payment by credit card. The person identified by the name and address could already be part of the conceptual system state, but does not have to be. Same for the credit card.
- `registerVehicle(a: Account, lp: LicencePlate, class: VehicleClass)` : sent by *Operator* when a driver wants to register a new car.
- `installationResult(t: Transponder, outcome: Boolean)`: sent by *Operator* to notify System of successful or unsuccessful installation of transponder.
- `cancelRegistration(t: Transponder)`: sent by *Operator* to notify System that a driver wants to return the transponder and cancel the registration of his vehicle.
- `transponderReturned(t: Transponder)`: sent by *Operator* to notify System of the return of a transponder.
- `recognizedPlate(t: Trip, lp: LicensePlate)`: sent by *Operator* after he recognized a licence plate image.
- `<<time-triggered>>endOfMonth()`: triggered at the end of the month to notify the System to send out the bills to all drivers. For accounts that are registered for auto-payment by credit card, the trips of the month are automatically billed to the credit card. In case a person has trips that are older than 3 months that he did not pay for, the government should be notified to deny future license plate renewals.

Output Messages

- `displayPicture(trip: Trip, pict: Picture)`: sent to *Operator* when a licence plate in unrecognizable.
- `getVehicleInfo(lp: LicencePlate)`: sent to the *GovernmentComputer* to obtain the vehicle information for the vehicle identified by the given licence plate.

- `bill(addressee: Person, transactions: Set(Transaction))`: sent to *PostalService* at the end of the month to inform driver about his transactions (trips, fines and monthly transponder lease).
- `bouncedCheque(addressee: Person, cheque: ChequeNumber)`: sent to *PostalService* to inform driver about a bounced check.
- `deliverTransponder(addressee: Person, t: Transponder)`: sent to *PostalService* in order to deliver a transponder to a registered driver.
- `accountOpened(a: Account)`: sent to *Operator* to notify driver of the opening of a new account.
- `registrationResult(outcome: Boolean)`: sent to *Operator* to notify driver of the success or failure of vehicle registration.
- `payCreditResult(addressee: Person, outcome: Boolean)`: sent to *PostalService* to notify driver of the success or failure of credit card payment.
- `creditAmount(c: CreditCardNumber, amount: Integer)`: sent to *CreditCardCompany* in order to check for the validity and solvency of a credit card.
- `denyRenewal(lp: LicencePlate)`: sent to the *GovernmentComputer* to block the licence plate renewal of the vehicle.

Backend Operation Model

The Operation Model assumes the existance of the following functions:

- `Location::distance(Location) : Distance` – a function that calculates the distance between two locations.
- `Time ''<'' Time : Boolean` – a function that can compare two Time values
- `Time::isPeakHour() : Boolean` – a function that returns true if the given Time falls within peak hours.
- `Picture::isRecognizeable() : Boolean` – a function that can determine if the licence plate number in the picture is recognizable.
- `Picture::getLicencePlate() : LicencePlate` – a function that uses image recognition techniques to extract a licence plate from the picture.
- `CreditCardNumber:getKind() : CreditCardKind` – a function that returns the kind of credit card for a given card number.

Operation: 407Backend::createAccount(name: String, address: String);
Scope: Person, Account;
Messages: OperatorTerminal::{AccountOpened}
New:
newAccount: Account;
newPerson: Person;
Alias: p: Person = **self**.person**@pre**→**any**(p | p.name = name **and** p.address = address);
Post:
newAccount.**oclIsNew**() &
if p.**oclIsUndefined**() **then**
newPerson.**oclIsNew**(name => name, address => address) &
newPerson.acc = newAccount
else
p.acc→**includes**(newAccount)
endif &
let card: CreditCard = **self**.creditCard→**any**(number = cardnumber) **in**
if card.**oclIsUndefined**() then
newCard.**oclIsNew**(number => cardnumber) &
newCard.myKind = newCard.getKind() &
newAccount.billedTo = newCard
else
newAccount.billedTo = card
endif
endlet &
sender^accountOpened(newAccount)

Operation: 407Backend::registerVehicle(a: Account, lp: LicencePlate, class: VehicleClass);
Scope: Account, Vehicle, Terminal, Transponder;
Messages: GovernmentComputer::{GetVehicleInfo}
New:
newVehicle: Vehicle;
newTransponder: Transponder;
Pre: **self**.vehicle**@pre**→**select**(plate = lp)→**isEmpty**();
Post:
newVehicle.**oclIsNew**((plate => lp, class => class) &
newVehicle.registeringAt = **sender**.terminal &
newVehicle.owner = a.owner &
newTransponder.**oclIsNew**() &
newVehicle.myTransponder = newTransponder &
a.registeredT→**includes**(newTransponder) & **self**.governmentComputer^getVehicleInfo(lp)

Operation: 407Backend::installationResult(t: Transponder, outcome: Boolean);
Scope: Transponder, Vehicle;
Messages: PostalService::{DeliverTransponder}
New: newTransponder: Transponder;
Post:
if outcome **then**
t.active
else
newTransponder.**oclIsNew**() &
t.registeredVehicle**@pre**.myTransponder = newTransponder &
t.account**@pre**.registeredT→**includes**(newTransponder) &
self.transponder→**excludes**(t) &
self.postalService^deliverTransponder(t.account**@pre**.owner, newTransponder)

endif

Operation: 407Backend::cancelRegistration(t: Transponder);
Scope: Transponder;
Post:
not t.active

Operation: 407Backend::transponderReturned(t: Transponder);
Scope: Transponder;
Pre: **not** t.active
Post:
self.transponder→**excludes**(t)

Predicate: 407Backend::TripCreated (v: Vehicle, newTrip: Trip, when: Time, where: Gantry, class: VehicleClass, kind:);
Body:
newTrip.**oclIsNew**() &
newTrip.detClass = class &
if where.type = GantryKind::entry **then**
newTrip.entryTime = when &
newTrip.entryGantry = where
else
newTrip.exitTime = when &
newTrip.exitGantry = where
endif &
v.trip = v.trip**@pre**→**append**(newTrip)

Operation: 407Backend::registeredPass(t: Transponder, when: Time, class: VehicleClass, pict: Picture);
Scope: Gantry, Vehicle, Trip, Transponder;
Alias:
myGantry: Gantry = **sender**.gantry;
v: Vehicle = t.registeredVehicle;
New: newTrip: Trip;
Post:
t.active &
if myGantry.type = GantryKind:entry **then**
self.TripCreated(v, newTrip, when, myGantry, class) &
if pict.isRecognizable() **then**
newTrip.detLP = pict.getLicencePlate()
else
– if we can't recognize the licence plate, then don't bother showing it to an operator
newTrip.detLP = v.plate
endif
else
– it is an exit
let myLastTrip: Trip = v.trip→**last**() in
if myLastTrip.exitGantry→**isEmpty**() **then**
myLastTrip.exitTime = when &
myLastTrip.exitGantry = myGantry
else
– entry was not registered because of a failure
self.TripCreated(v, newTrip, when, myGantry, class)
endif
endlet

```
    endif

Operation: 407Backend::unregisteredPass(when: Time, class: VehicleClass, pict: Picture);
    Scope: Gantry, Vehicle, Trip, Terminal;
    Messages: OperatorTerminal::{DisplayPicture}, GovernmentComputer::{GetVehicleInfo}
    Alias:
     myGantry: Gantry = sender.gantry;
    New:
      newTrip: Trip;
      newVehicle: Vehicle;
    Post:
     if myGantry.type = GantryKind:entry then
      newTrip.oclIsNew() &
      newTrip.entryTime = when &
      newTrip.detClass = class &
      newTrip.entryGantry = myGantry &
      if pict.isRecognizable() then
       newTrip.detLP = pict.getLicencePlate() &
       let v: Vehicle = self.vehicle→select(plate = lp) in
        if v.oclIsUndefined() then
         newVehicle.oclIsNew() &
         newVehicle.trip = Sequence{newTrip} &
         self.governmentComputer^getVehicleInfo(pict.getLicencePlate())
        else
         v.trip = v.trip@pre→append(newTrip)
        endif
       endlet
      else
       – license plate is not recognizable, need to display picture on a terminal
       let chosenTerminal: Terminal = self.terminal→any(t | t.registering→isEmpty()) in
        chosenTerminal^displayPicture(newTrip, pict) &
        newTrip.displayedAt = chosenTerminal
       endlet
      endif
     else
      – it is an exit
      if pict.isRecognizable() then
       let v: Vehicle = self.vehicle→select(plate = pict.getLicencePlate()) in
        if v.oclIsUndefined() then
         newVehicle.oclIsNew() &
         newTrip.oclIsNew() &
         newTrip.exitTime = when &
         newTrip.detClass = class &
         newTrip.exitGantry = myGantry &
         newVehicle.trip = Sequence{newTrip} &
         self.governmentComputer^getVehicleInfo(pict.getLicencePlate())
        else
         let myTrip: Trip = v.trip→last()) in
          if myTrip.exitGantry.oclIsUndefined() then
           myTrip.exitTime = when &
           myTrip.exitGantry = myGantry
          else
           self.TripCreated(v, newTrip, when, myGantry, class) &
          endif
```

```
        endlet
      endif
    endlet
  else
    – picture unrecognizable
    newTrip.oclIsNew() &
    newTrip.exitTime = when &
    newTrip.exitGantry = myGantry &
    let chosenTerminal: Terminal = self.terminal→any(t | t.registering→isEmpty()) in
      chosenTerminal^displayPicture(newTrip, pict) &
      newTrip.displayedAt = chosenTerminal
    endlet
  endif
endif
```

Operation: 407Backend::vehicleInfo(lp: LicencePlate, class: VehicleClass, ownerName: String, ownerAddress: String);

Scope: Vehicle, Person, Transponder, Account, Terminal;
Messages: PostalService::{DeliverTransponder}, OperatorTerminal::{RegistrationResult}
Alias: registeringVehicle: Vehicle = **self**.terminal.registering**@pre**→**any**(plate = lp);
New: newPerson: Person;
Post:

```
if registeringVehicle.oclIsUndefined() then
  – vehicleInfo for an unregistered vehicle that takes the highway for the first time
  let v: Vehicle = self.vehicle→any(plate = lp),
    owner: Person = self.person→any(name = ownerName and address = ownerAddress) in
    v.class = class &
    if owner.oclIsUndefined() then
      newPerson.oclIsNew(name => ownerName, address => ownerAddress} &
      newPerson.vehicle = Set{v}
    else
      owner.vehicle→includes(v)
    endif
  endlet
else
  – vehicleInfo is for a vehicle that is currently being processed for registration by an operator
  let success: Boolean = registeringVehicle.class = class and
    registeringVehicle.owner.name = ownerName and
    registeringVehicle.owner.address = ownerAddress in
    if not success then
      – not specified in the use cases what should happen in this case!
    else
      self.postalService^deliverTransponder(registeringVehicle.owner)
    endif
    registeringVehicle.registeringAt@pre.operatorTerminal^registrationResult(success) &
    registeringVehicle.registeringAt→isEmpty()
  endlet
endif
```

```
Operation: 407Backend::recognizedPlate(tr: Trip, lp: LicencePlate);
  Scope: Gantry, Vehicle, Trip, Terminal;
  Messages: GovernmentComputer::{GetVehicleInfo}
  Alias:
   myTerminal: Terminal = sender.terminal;
   v: Vehicle = self.vehicle→select(plate = lp);
  New: newVehicle: Vehicle;
  Post:
   if v.oclIsUndefined() then
     newVehicle.oclIsNew() &
     newVehicle.trip = Sequence{tr} &
     newVehicle.trip.detLP = lp &
     self.governmentComputer^getVehicleInfo(lp)
   elseif tr.exitGantry→isEmpty() then
     v.trip = v.trip@pre→append(tr)
   elseif v.trip→last().exitGantry@pre→isEmpty() then
     v.trip→last().exitGantry = tr.exitGantry &
     v.trip→last().exitTime = tr.exitTime &
     self.trip→excludes(tr)
   else
     v.trip = v.trip@pre→append(tr)
   endif &
   myTerminal.displaying→excludes(tr)

Context: Trip
  Alias:
    peak: Boolean = self.entryTime.isPeakHour() or self.exitTime.isPeakHour();
    travelledDistance: Distance =
      if t.entryGantry→isEmpty() or t.exitGantry→isEmpty() then
        minimumDistance
      else
        t.entryGantry.pos.distance(t.exitGantry.pos)
      endif
  Def: costForDistance() : Integer =
   if self.detClass = VehicleClass::light then
     if peak then travelledDistance * 16.25
     else travelledDistance * 15.5
     endif
   elseif self.detClass = VehicleClass::heavy_single then
     if peak then travelledDistance * 32.5
     else travelledDistance * 31.5
     endif
   else
     if peak then travelledDistance * 48.75
     else travelledDistance * 46.5
     endif
   endif

Context: Trip
  Def: videoCharge() : Integer =
   if self.vehicle.myTransponder→isEmpty() then 350
   else 0
   endif
```

Context: Trip
Def: cheatingFine() : Integer =
if ((**self**.vehicle.myTransponder→**notEmpty**() **and** (**self**.vehicle.plate <> **self**.detLP)) **or**
((**self**.detClass <> VehicleClass::single) **and** (**self**.vehicle.myTransponder→**isEmpty**()) **then**
5000
else 0
endif

Context: Trip
Def: cost() : Integer = **self**.costForDistance() + **self**.videoCharge() + **self**.cheatingFine();

Operation: 407Backend::endOfMonth();
Scope: Account, Person, Vehicle, Trip, Cheque, CreditCard, CreditCardKind;
Messages: PostalService::{Bill}, GovernmentComputer::{DenyRenewal}, CreditCardCompany::{creditAmount}
Post:
– the bills are sent to registered accounts
self.account→**forAll**(a |
let contents: Set(Transaction) **in**
contents→**includes**(Tuple{TransactionKind::monthlyFee, 'Monthly Transponder Fees',
300 * a.registeredT→**select**(active)→**size**()}) &
a.registeredT.registeredVehicle.trip→**select**(t | t.cost() > t.amountPaid)→**forAll**(t |
contents→**includes**(Tuple{TransactionKind::registeredTrip, 'Trip x to y', t.cost() - t.amountPaid})) &
self.postalService^bill(a.owner, contents)
endlet) &
– the bills are sent to unregistered vehicle owners
self.vehicle→**forAll**(v | v.myTransponder→**isEmpty**() **implies**
let contents: Set(Transaction) **in**
v.trip→**select**(t | t.cost() > t.amountPaid)→**forAll**(t |
contents→**includes**(Tuple{TransactionKind::unregisteredTrip, 'Trip x to y', t.cost() - t.amoutPaid})) &
self.postalService^bill(v.owner, contents)
endlet) &
– the credit cards associated with accounts are charged
let amountToBeCharged = **self**.account.registeredT→**select**(active)→**size**() * 300 +
self.account→**forAll**(a | a.registeredT.registeredVehicle.trip→**select**(t | t.cost() > t.amountPaid)
→**collect**(cost() - amountPaid)→**sum**() **in**
a.billedTo.myKind.creditCardCompany^creditAmount(a.billedTo.number, amount)
endlet &
– licence plate renewals are denied for customers that are overdue
self.vehicle→**forAll**(v | v.trip→**select**(t | (t.exitTime < self.clock.now - 3 * Time::months or
t.entryTime < self.clock.now - 3 * Time::months)→**notEmpty**() **and**
v.owner.currentCheque→**isEmpty**())
implies self.governmentComputer^denyRenewal(v.plate))

Operation: 407Backend::creditResult(c: CreditCardNumber, amount: Integer, outcome: Boolean);
Scope: CreditCard, CreditCardKind, Account, Person, Vehicle, Trip;
Alias: payer: Person = **self**.creditCard→**select**(card | card.number = c).paysFor.owner→**any**()
Messages: Operator::{PayCreditResult}
Post:
if outcome **then**
– we assume that, if successful, the amount is always the total amount requested
self.creditCard→**select**(card | card.number = c).paysFor.registeredT.registeredVehicle.trip→**forAll**(t |
t.amountPaid = t.cost())
endif &
self.postalSercive^payCreditResult(payer, outcome)

Operation: 407Backend::chequeReceived(p: Person, cheque: ChequeNumber);
Scope: Person, Cheque;
New: newCheque: Cheque;
Post:
newCheque.**oclIsNew**() &
newCheque.number = cheque &
p.currentCheque = newCheque

Operation: 407Backend::chequeResult(cheque: ChequeNumber, amount: Integer, outcome: Boolean);
Scope: Trip, Person, Vehicle, Cheque;
Alias:
payer: Person = **self**.cheque→**any**(ch | ch.number = cheque).sentBy**@pre**;
trips: Set(Trip) = payer.myVehicle.trip;
Messages: PostalService::{BouncedCheque}
Post:
if outcome **then**
trips.amountPaid→**sum**() = trips.amountPaid**@pre**→**sum**() + amount &
trips→**select**(t | t.amountPaid = t.cost()).exitTime→**max**() < trips→**select**(t |
t.amountPaid < t.cost()).exitTime→**min**() &
let oldestNotCompletelyPaidTrip = trips→**any**(t | t.amountPaid < t.cost() **and**
t.exitTime < trips→**select**(amountPaid < cost().exitTime→**min**()) **in**
oldestNotCompletelyPaidTrip.cost() - oldestNotCompletelyPaidTrip.amountPaid >
trips→**select**(t | t.amountPaid < t.cost()).amountPaid→**sum**())
endlet
else
self.postalService^bouncedCheque(payer, cheque)
endif &
payer.currentCheque→**isEmpty**()

Design

In this part you are to elaborate a partial design for the 407 ETR Backend System. Some decisions have been made already:

- When communicating with the environment, transponders are identified by unique serial numbers encoded as integers.
- When communicating with the environment, accounts are identified by unique account numbers encoded as integers.
- When communicating with the environment, drivers are identified with their name and address, encoded as Strings.
- Licence plates are encoded as Strings.
- There is a design class *Picture* that provides the methods `isRecognizable() : Boolean` and `getLicencePlate() : String`. A specialized team will implement these methods using image recognition algorithms, and therefore you can simply assume they exist. The *Picture* class is serializeable, and therefore instances of *Picture* can be sent over the network (for instance, as a parameter to registeredPass and unregisteredPass).
- All attributes of classes are to be made private. If their values are to be available to other classes, then appropriate *getter* or *setter* methods have to be defined.

Task 1: Interaction Model

Establish an *Interaction Model* for the following system operations: `registerVehicle`, `registeredPass`, `unregisteredPass`, `vehicleInfo`, `recognizedPlate`, `chequeReceived`, `chequeResult`, `endOfMonth`. (To keep the size of the design reasonable, you do not have to design `createAccount`. You can simply assume that this operation sets up accounts in a way that is convenient for your other operations. Also, you do not have to handle credit cards, which means that you don't need to design `creditResult`, or handle the auto-payment by credit card in `endOfMonth`).

Justify the choice of the controller object for each operation.

Task 2: Dependency Model

Establish a *Dependency Model* that depicts all the dependencies among the design classes discovered in task 1.

Task 3: Design Class Model

Establish a *Design Class Model* that shows the (private) attributes and methods of all design classes used in task 1 and 2.

Hand-In

Please hand in a paper copy of your solution until Wednesday November 21st. You can hand in electronically by sending an email to Omar.Alam@mail.mcgill.ca with the title "COMP-533 Final of yournames" and cc me as well (Joerg.Kienzle@mcgill.ca). If you don't get an acknowledgment for your email, send us another email (without attachment, but putting the handin somewhere where we can download it).

Remember that you are allowed to work in groups of 2, but not with a person you worked with for a previous assignment. If you work with someone, please hand in a single copy with both names.