Technical University of Denmark

02291 System Integration

Toll System

Group 26

Authors:	*Student IDs:*
Cristina Ailoaei	s213804
Dinesh Kumar Khadka	s220073
Eleni Gaitaniou	s212361
Johannes Vaht Steinitz	s205534
Nikolaos Karavasilis	s213685
Roza Hasso	s213558
Tamas Paulik	s212569

Saturday 28th October, 2023

Contents

1 Introduction **5**
1.1 Methodology . . . 5

2 Requirements **6**
2.1 Domain Analysis . . . 6
2.1.1 Glossary . . . 6
2.1.2 Domain class diagram . . . 9
2.1.3 Activity diagrams . . . 10
2.2 Functional Requirements . . . 10
2.2.1 Use Cases . . . 11
2.2.2 Detailed Use Cases . . . 13
2.3 Non-Functional Requirements . . . 16

3 Acceptance Tests **18**
3.1 Check-in with Credit Card . . . 18
3.2 Check-out with Credit Card . . . 18
3.3 Buy Toll Tag . . . 19
3.4 Check-in with Toll Tag . . . 19
3.5 Check-out with Toll Tag . . . 20
3.6 Pay Overdue Bill . . . 20
3.7 Change Rate . . . 21

4 Design **22**
4.1 Component Design . . . 22
4.2 Class Design . . . 25
4.2.1 Toll Lane . . . 26
4.2.2 Toll Station . . . 27
4.2.3 Enterprise . . . 29
4.3 OCL Constraints . . . 30
4.4 Behaviour Design . . . 31
4.4.1 Toll Lane Computer Component . . . 31
4.4.2 Toll Tag Component . . . 32
4.4.3 Station Server Component . . . 33
4.4.4 Enterprise Server Component . . . 34
4.4.5 Enterprise Client Component . . . 35

5 Validation **36**
5.1 Use Case Realisation . . . 36
5.1.1 Check in with Credit Card . . . 36
5.1.2 Check out with Credit Card . . . 37
5.1.3 Buy Toll Tag . . . 38

5.1.4 Check in with Toll Tag 39
5.1.5 Check out with Toll Tag 40
5.1.6 Pay Overdue Bill 41
5.1.7 Change Rate 43

6 Conclusion **45**
6.1 Experience with the project 45

7 Work Division **46**

List of Figures

1 Domain Class Diagram . . . 9
2 Activity Diagram . . . 10
3 Use Case Diagrams . . . 12
4 Component Diagram . . . 22
5 Component Diagram with interfaces - Toll Lane . . . 24
6 Component Diagram with interfaces - Toll Station . . . 24
7 Component Diagram with interfaces - Enterprise . . . 25
8 Detailed class diagram - holistic view . . . 25
9 Detailed Class Diagram: Shared Library . . . 26
10 Detailed Class Diagram: Toll Lane . . . 27
11 Detailed Class Diagram: Toll Lane . . . 28
12 Detailed Class Diagram: Enterprise . . . 29
13 Toll Lane Computer Component (LSM) . . . 32
14 Toll Tag Component (LSM) . . . 33
15 Station Server Component (LSM) . . . 34
16 Enterprise Server Component (LSM) . . . 35
17 Enterprise Client Component (LSM) . . . 35
18 Sequence diagram: Check-in with Credit Card . . . 37
19 Sequence diagram: Check-out with Credit Card . . . 38
20 Sequence diagram: Buy Toll Tag . . . 39
21 Sequence diagram: Check-in with Toll Tag . . . 40
22 Sequence diagram: Check-out with Toll Tag . . . 41
23 Sequence diagram: Pay Overdue Bill . . . 43
24 Sequence diagram: Change Rate . . . 44

List of Tables

1 *Check-In with credit card* detailed use case . . . 13
2 *Check-Out with credit card* detailed use case . . . 14
3 *Check-In with Toll Tag* detailed use case . . . 14
4 *Check-Out with Toll Tag* detailed use case . . . 15
5 *Change Rates* detailed use case . . . 15
6 *Buy Toll Tag* detailed use case . . . 16
7 *Pay Overdue Bill* detailed use case . . . 16
8 Fit Table for *Check-in with Credit Card(Main Scenario)* . . . 18
9 Fit Table for *Check-in with Credit Card(Alternative Scenario)* . . . 18
10 Fit Table for *Check-out with Credit Card(Main Scenario)* . . . 18
11 Fit Table for *Check-out with Credit Card(Main Scenario)* . . . 19
12 Fit Table for *Buy Toll Tag (Main Scenario)* . . . 19
13 Fit Table for *Buy Toll Tag (Alternative Scenario)* . . . 19

14 Fit Table for *Check-in with Toll Tag(Main Scenario)* 19
15 Fit Table for *Check-in with Toll Tag(Alternative Scenario)* 20
16 Fit Table for *Check-out with Toll Tag(Main Scenario)* 20
17 Fit Table for *Check-out with Toll Tag(Alternative Scenario)* 20
18 Fit Table for *Pay Overdue Bill (Main Scenario)* 20
19 Fit Table for *Pay Overdue Bill (Alternative Scenario)* 21
20 Fit Table for *Change Rate (Main Scenario)* 21
21 Fit Table for *Change Rate (Alternative Scenario)* 21

1 Introduction

Responsible Student: Johannes, Nikolaos

This report is an examination project for the spring 2022 course 02291 System Integration at DTU, Copenhagen. The report presents a model adaptation of a motorway toll system which handles toll charges for different vehicle types when entering and leaving a motorway. This includes the check-in and check-out of vehicles, the purchase of different tickets, generation of statistics, client interactivity and most importantly the request and response communication between the system components.

This report includes the following sections: Domain Syllabus -Analysis, Detailed Use Cases, Acceptance Tests, Methodology, Different Models used and Conclusion. This report includes the following models: Conceptual Domain Model, Detailed Class Diagram, Use Case Diagram, Component Diagram, Sequence Diagrams, Life Cycle State Machines and OCL Constraints.

1.1 Methodology

Responsible Student: Johannes, Tamas

The methodology used for this project can be separated into the methodology used for modelling and the methodology used for group workflow, whereas we mainly lean against the Agile methods, produce a model, validate and iterate. The main motivation behind the teamwork of the project will be to assign a driver for each work session who controls the endpoint for modelling whereas the remaining group members act as co-drivers, suggesting and steering the modelling process for the driver. Using this methodology, every group member gains experience in both the focus behind modelling and focusing on reflecting on the model. We will also divide the responsibility for some parts of the project, in this way, chunks of the model will be done for each meet up. Specifically, the responsibility of the use cases is divided into subgroups.

The methodology used for modelling is closely connected to the workflow used in the course and leaning toward an exploratory design. Starting by defining the domain, deriving the domain concepts and stating the business logic and the system requirements we can sequentially derive the different components and required interactions between them. Once these have been determined we define the use cases and detailed use cases in order to create the class diagram and iteratively modify the class diagram and sequence diagram to fit the defined use cases. The design and behaviour of the system components relate closely to the use cases. Sequence diagrams are used in the final stage for validating the model. Iterating over this workflow will be the underlying methodology for this project.

As the group consists of seven group members we have modeled the following seven use cases: (1) *Check-in with credit card*, (2) *Check-out with credit card*, (3), *Check-in with toll tag*, (4) *Check-out with toll tag*, (5) *Change rates*, (6) *Buy toll tag* and (7) *Pay overdue bill*.

2 Requirements

2.1 Domain Analysis

Responsible Student: Johannes, Nikolaos
The domain analysis takes part in highlighting the hand-out text. The below glossary is used to define the entities and objects within the scope of the system.

2.1.1 Glossary

Responsible Student: Eleni, Cristina

- **Toll system:** System that is located on motorways and handles toll charges for vehicles.
- **Vehicle:** Motorized means of transportation restricted to cars, trucks and motorbikes.
- **Toll charge:** Amount of money paid by vehicle owners in order to use the motorway. Payment is ensured through the purchase of a single ticket or a toll tag.
- **Single ticket:** One-time-use ticket with a 24-hour validity and fixed price per vehicle type, which gives the holder the right to enter and exit the motorway. It can only be purchased at the normal check-in toll lanes of a toll station upon entering the motorway, by credit card payment.
- **Toll tag:** Multi-time-use tag in the form of an RFID transponder which gives the holder the right to enter and exit the motorway. The tag must be ordered and paid in advance through the system's web application and is bound to one vehicle only.
- **Toll station:** Transit point consisting of toll lanes, found at the entrances and exits of a motorway. In addition, a station server-client and enterprise server-client are present. Only the exits consist of a small cashier desk area destined for support and a credit card reader.
- **Toll lane:** Road lane which can be either normal or express type. It can also be of a check-in or check-out type, depending on how it is situated in relation to the motorway (entry or exit). Each toll lane has a toll lane computer.
- **Normal check-in lane:** Toll lane, part of a toll station situated at the entrance of the motorway, where the customer can check-in by purchasing a single ticket. It is equipped with a touch screen, a credit card reader, a printer and a barrier.
- **Normal check-out lane:** Toll lane, part of a toll station situated at the exit of the motorway, where the customer can check-out by scanning a single ticket. It is equipped with a single ticket reader and a barrier.
- **Express check-in lane:** Toll lane, part of a toll station situated at the entrance of the motorway, where the customer can check-in automatically by toll tag. It is equipped with an antenna and a barrier.

- **Express check-out lane:** Toll lane, part of a toll station situated at the exit of the motorway, where the customer can check-out automatically by toll tag. It is equipped with an antenna and a barrier.
- **Toll lane computer:** This type of computer is located at each toll lane and is used to handle the check-in and check-out process for one vehicle at a time by integrating all devices at the toll lane while contacting the enterprise server.
- **Touch screen:** Interface of the toll lane computer where the customer can select the vehicle type.
- **Credit card reader:** Device which handles the credit card payment.
- **Printer:** Machine that prints single tickets.
- **Barrier:** Mechanical obstacle intended to block passage. After a successful check-in or check-out, the barrier opens. The barrier closes when its sensor detects that the vehicle has passed through.
- **Antenna:** RFID device which authenticates vehicles wirelessly by their toll tags.
- **Single ticket reader:** Device which validates the inserted single ticket.
- **Check-in with credit card:** Process which takes place at a normal check-in lane, where the customer purchases a single ticket and can enter the motorway.
- **Check-out with credit card:** Process which takes place at a normal check-out lane, where the customer inserts their single ticket into the single ticket reader and exits the motorway. Check-out might fail if the ticket is older than 24 hours or was validated in a previous check-out. In case of check-out failure, a cashier charges the owner.
- **Check-in with toll tag:**Process which takes place at an express check-in lane, where the installed antenna detects and reads the radio signals transmitted by the tag. This process enables automatic vehicle identification and authorizes entry to the motorway. Check-in might fail if the toll tag is not readable or is marked as invalid. In case of check-in failure, the customer has to buy a single ticket.
- **Check-out with toll tag:**Process which takes place at an express check-out lane, where the installed antenna detects and reads the radio signals transmitted by the tag. This process enables automatic vehicle identification and authorizes the exit out of the motorway. The price for the trip is computed using the distance travelled and then added to the monthly bill. Check-in might fail if the toll tag is not readable or is marked as invalid. In case of check-out failure due to the toll tag not being recognized by the antenna, a cashier charges the owner. In case of check-out failure due to the toll tag being invalid, the cost of the trip is added to the customer's account.
- **Cashier:** Employee stationed in a toll station at the exit of the motorway responsible for handling check-out failures.

- **Buy toll tag:** Action of purchasing a toll tag via the toll system's web application. This process requires the customer to fill out a form with personal information, such as their name, vehicle type, vehicle license plate number and credit card details. The previous action follows an immediate fixed payment. The toll tag is then mailed to the vehicle owner.
- **Deduct monthly bill:** Monthly process of charging customers who own a toll tag by subtracting money directly from their provided credit cards for all the trips executed since the last deduction. In case of bill deduction failure, the customer is notified and all its toll tags are marked as invalid.
- **Pay overdue bill:** Action of paying a bill after an automatic monthly deduction has failed. The payment is done via the web application, using a credit card.
- **Show reports:** Process performed by the station manager in order to query toll station statistical reports from the station client.
- **Statistical report:** Information regarding how many, what type, and how much money was earned in a given period of time. Additionally, the statistics include how often toll tags and single tickets were used.
- **Station manager:** Employee who uses the station client to query/show reports.
- **Station client:** System which communicates with the station server. It is used by station managers to show reports and modify customer data.
- **Station server:** System that integrates all toll lane computers, a credit card reader and communicates with the station client.
- **Generate reports:** Process of creating statistical reports with the same information for all toll stations from the enterprise client. It is performed by the enterprise manager.
- **Enterprise Manager:** Employee who uses the enterprise client to generate reports, change rates and notify customers.
- **Enterprise client:** System which communicates with the enterprise server. It is used by the enterprise manager to generate reports, change rates and notify customers about planned rate changes.
- **Enterprise server:** System that interacts with the toll stations, the Bank, web server and the enterprise client.
- **Enterprise:** Ensemble of toll stations, an enterprise server and an enterprise client.
- **Bank:** Entity that validates customer payments.
- **Web server:** System that integrates the web application with the enterprise server.
- **Web application:** Application which runs on the web server, used by the customer to buy a toll tag, see open invoices and pay them by credit card.

- **Change rate:** Action performed by the enterprise manager to change the toll rates for single tickets and toll tags. Upon enforcement, the price for all stations updates at once.
- **Notify customers:** Action performed by the enterprise manager to send email alerts to all customers regarding anticipated rate changes.

2.1.2 Domain class diagram

Responsible Student: Johannes, Tamas

The following figure displays the domain class diagram, which defines the underlying domain and interactions and relations between the included entities from a conceptual perspective.

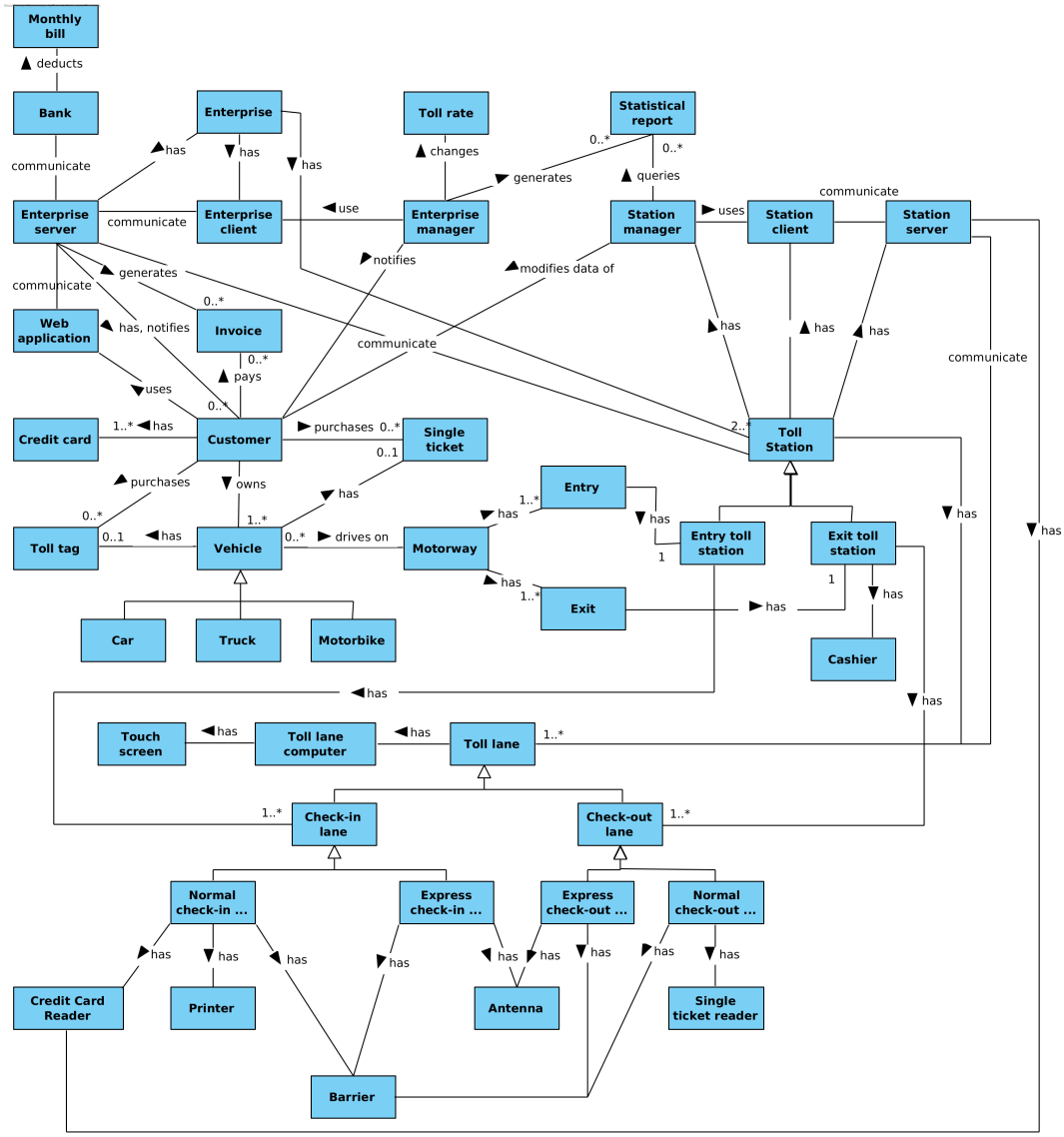

Figure 1: Domain Class Diagram

2.1.3 Activity diagrams

Responsible Student: Tamas, Dinesh

The below figure provides an overview of the business logic of the system in form of an activity diagram. The activity diagram includes the logic of the chosen seven use cases, which act as the base for further modelling.

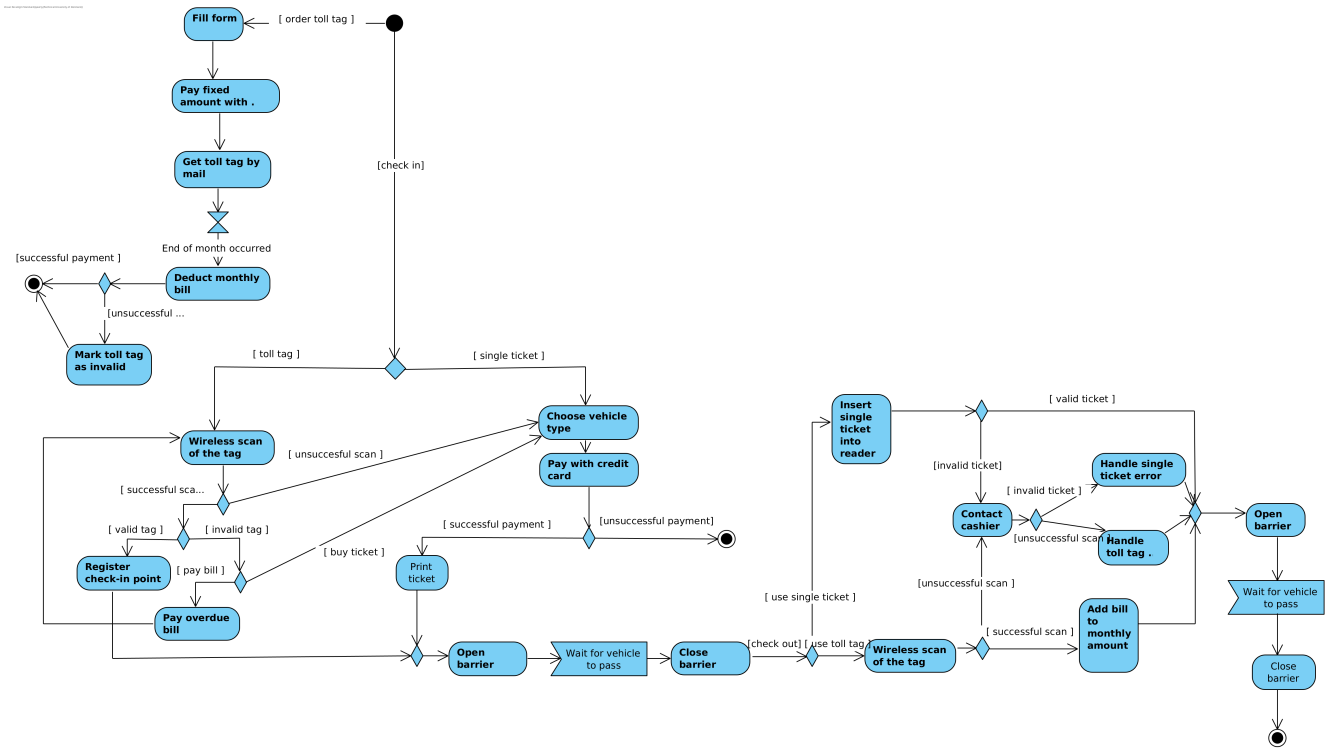

Figure 2: Activity Diagram

2.2 Functional Requirements

Responsible Student: Eleni, Roza

- All toll system clients, servers and endpoints need to be up to date, normally operating and interconnected
- Lane computers need to be functional and connected to toll system servers
- Connection to the web server and bank need to be established and active
- The antenna wirelessly scans toll tags within a certain distance, one vehicle at a time
- Only enterprise managers can access enterprise clients

- Only station managers can access station clients
- Card readers are connected with the bank and should be working accordingly
- The web server is functional and accessible for all customers
- The toll system records statistics on all check-in/check-out, ticket type and vehicle type
- Barriers at an express lane check-in only open when a valid toll tag is registered
- If a toll tag is rendered invalid mid-trip, calculation of price and adding it to the customer bill is sufficient
- Barriers at a normal lane check-in only open when the customer has a single ticket
- Barriers at a normal lane check-out only open when a single ticket is valid
- A station client has access to view the reports
- A station client can modify customer data
- An enterprise client can access the whole system
- Enterprise server has internet connectivity via web server
- A single ticket has a fixed price per vehicle type and is only valid for 24 hours
- The price when using a toll tag is dependent on the distance travelled and the vehicle type
- A toll tag is registered to one specific vehicle
- Payment for use of motorway with toll tag is deducted at the end of each month
- Toll tags can be ordered from the web application
- Toll tags have a fixed price and must be paid when ordered
- A given customer's credit card must be valid

2.2.1 Use Cases

The following diagram represents all the actors of our system who can be a Customer, Station Manager, Cashier, Enterprise Manager or as a secondary actor the bank.

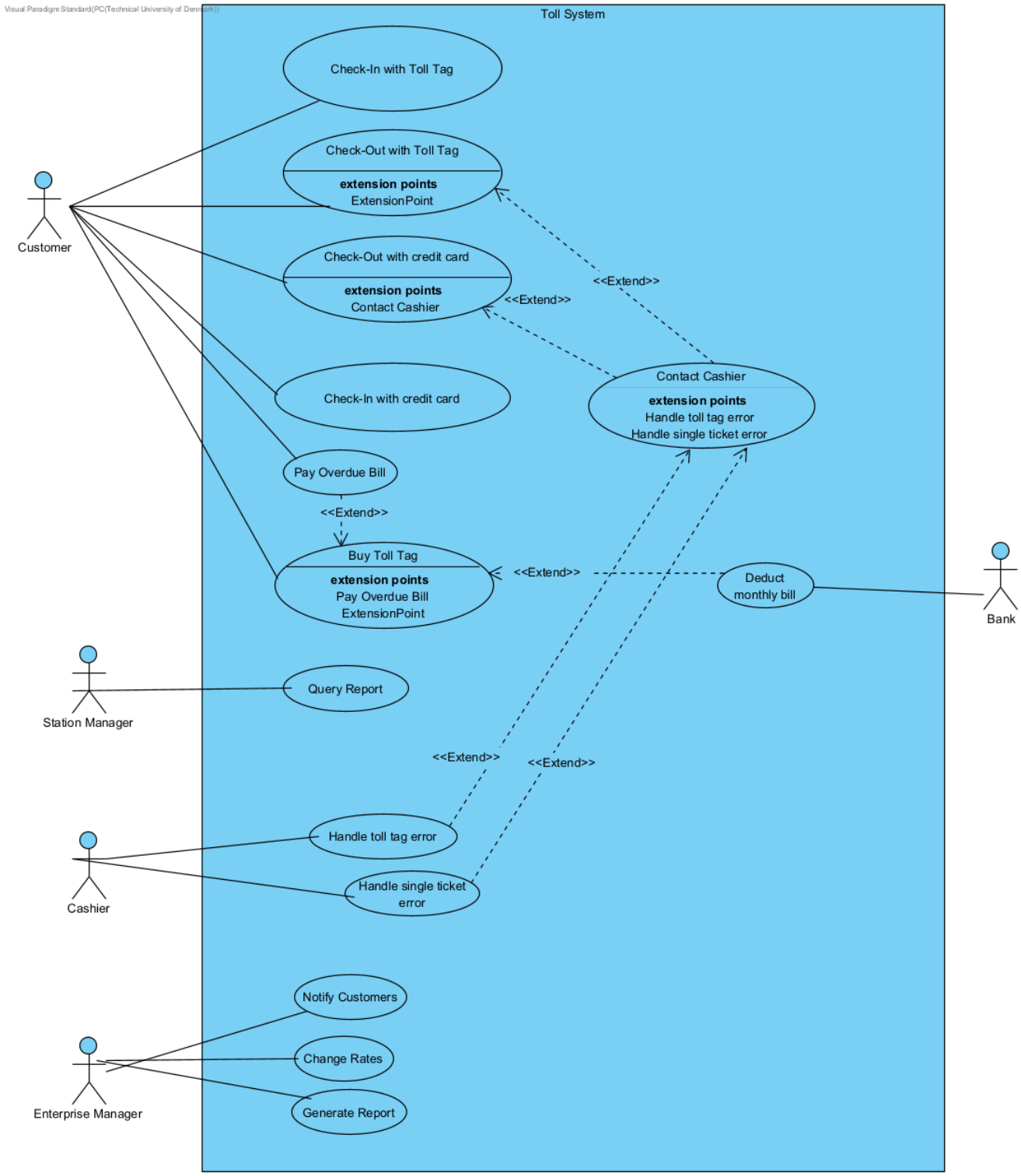

Figure 3: Use Case Diagrams

2.2.2 Detailed Use Cases

Responsible Student: Eleni, Nikolaos

This section provides the above mentioned use cases in a detailed form.

Use case name	Check in with credit card
Description	The scenario is when a customer purchases a single ticket on a normal check in lane at the toll station
Actors	Customer
Preconditions	Customer owns a valid credit card
Main Scenarios	1. The customer chooses the vehicle type via a touch screen 2. The customer pays with credit card using the credit card reader 3. System registers check in 4. The customer gets a printed single ticket from the printer 5. Barrier opens 6. Sensor detects that vehicle has passed 7. Barrier closes
Alternative Paths	2.a The credit card is rejected
Postconditions	The customer owns a printed out single ticket for his vehicle and has entered the motorway
Assumptions	-
Author	Roza Ibrahim Hasso

Table 1: *Check-In with credit card* detailed use case

Use case name	Check out with credit card
Description	The customer checks out his single ticket on a normal check out lane in the toll station
Actors	Customer
Preconditions	1. The customer has a single ticket 2. The customer must have passed through a check-in point
Main Scenarios	1. Customer get the single ticket read using the single ticket reader 2. Toll lane computer checks the validity of the ticket 3. System registers check out 4. Barrier opens 5. Sensor detects that vehicle has passed 6. Barrier closes
Alternative Paths	2.a The ticket is not valid anymore 4. The cashier manually charges the customer for an extra amount
Postconditions	-
Assumptions	The toll system is fully operational
Author	Eleni Gaitaniou

Table 2: *Check-Out with credit card* detailed use case

Use case name	Check in with toll tag
Description	The customer checks in at an express lane using a toll tag at the toll station
Actors	Customer
Preconditions	Customer owns a toll tag
Main Scenarios	1. Antenna scans the toll tag 2. System checks the validity of the toll tag and registers check in 3. Barrier opens 4. Sensor detects that vehicle has passed 5. Barrier closes
Alternative Paths	1.a Toll tag scanning failed 2. The customer checks in with credit card 2.b The toll tag is not valid anymore 4. The customer checks in with credit card
Postconditions	The customer entered the motorway
Assumptions	-
Author	Nikolaos Karavasilis

Table 3: *Check-In with Toll Tag* detailed use case

Use case name	Check out with toll tag
Description	The customer checks out a toll tag on a express check out lane in the toll station
Actors	Customer
Preconditions	The customer entered the motorway
Main Scenarios	1. Antenna scans the toll tag 2. System checks whether toll tag has been checked in 3. System registers check out 4. Barrier opens 5. Sensor detects that vehicle has passed 6. Barrier closes
Alternative Paths	2.a Toll tag has not been checked in
Postconditions	Total price is calculated by multiplying the price per km with the distance between the check-in and check-out and the final price is added to the monthly bill for that toll tag
Assumptions	The toll system is fully operational
Author	Dinesh Kumar Khadka

Table 4: *Check-Out with Toll Tag* detailed use case

Use case name	Change rates
Description	The enterprise manager changes the toll rate for single tickets, toll tags or fixed amount for toll tags
Actors	Enterprise Manager
Preconditions	-
Main Scenarios	1. The enterprise manager chooses between single ticket, toll tag or fixed amount for toll tag 2. The system replaces the old price with the new price
Alternative Paths	-
Postconditions	The price changes for all stations at once and web application
Assumptions	-
Author	Cristina Ailoaei

Table 5: *Change Rates* detailed use case

Use case name	Buy toll tag
Description	The customer buys a toll tag with a credit card via a web application and receives it by mail
Actors	Customer
Preconditions	The customer should be logged in its web application account, registration form is opened
Main Scenarios	1. The customer fills the form with their name, vehicle type, license plate number and credit card number 2. The system initiates the payment 3. The system registers a new toll tag
Alternative Paths	2.a The initiated payment fails
Postconditions	The customer receives the ordered toll tag and places it on the specific vehicle
Assumptions	-
Author	Tamas Janos Paulik

Table 6: *Buy Toll Tag* detailed use case

Use case name	Pay overdue bill
Description	The customer pays its toll tag overdue bill via its web application account
Actors	Customer
Preconditions	The customer should be logged in its web application account and have a valid credit card
Main Scenarios	1. Customer views its overdue bills 2. Customer enters credit card information 3. System initiates payment and mark all toll tags for the customer as valid 4. Customer receives an indication that the payment is successful
Alternative Paths	3.a Payment is unsuccessful
Postconditions	All the toll tags are valid again and bill is marked as paid
Assumptions	-
Author	Johannes Vaht Steinitz

Table 7: *Pay Overdue Bill* detailed use case

2.3 Non-Functional Requirements

Responsible Student: Eleni, Roza

These non-functional requirements are requirements that are out of scope for the models but exist as condition on some of the systems functionality.

- Motorway should only be accessible by vehicles
- Accessing the motorway is only possible through a normal lane or express lane check-in

- Exiting the motorway is only possible through a normal lane or express lane check-out
- Normal lanes only allow the purchase of one ticket at a time
- Normal lanes are only accessible with single-ticket
- Normal lanes have a touch screen
- Normal lanes only accept credit card
- In normal lanes, a single ticket is printed after purchase
- If a single ticket is invalid at check-out, manual handling is required
- Express lane antennas should only scan toll tags
- Express lanes are only accessible with toll tags
- Generating report is only authorized by enterprise manager
- The cashier handles invalid check-outs manually with a device connected to both a credit card reader and the toll station

3 Acceptance Tests

Responsible Student: Dinesh, Nikolaos

In this section, we present the acceptance tests for our use cases. These tests show the compliance of the use cases and their intended steps detailed in section 2.2.

3.1 Check-in with Credit Card

ActionFixture		
start	CheckInWithCreditCardFixture	
enter	Vehicle Type	[car]
enter	Credit Card	[1234]
check	Payment Succeeded	True
check	Ticket Printed	True
check	Barrier Opened	True
enter	Vehicle Passed	True
check	Barrier Closed	True

Table 8: Fit Table for *Check-in with Credit Card(Main Scenario)*

ActionFixture		
start	CheckInWithCreditCardFixture	
enter	Vehicle Type	[car]
enter	Credit Card	[1234]
check	Payment Succeeded	False
check	Barrier Closed	True

Table 9: Fit Table for *Check-in with Credit Card(Alternative Scenario)*

3.2 Check-out with Credit Card

ActionFixture		
start	CheckOutWithCreditCardFixture	
enter	Single Ticket ID	[id = 1]
check	Validity of the Ticket	True
check	Barrier Opened	True
enter	Vehicle Passed	True
check	Barrier Closed	True

Table 10: Fit Table for *Check-out with Credit Card(Main Scenario)*

ActionFixture		
start	CheckOutWithCreditCardFixture	
enter	Single Ticket ID	[id = 1]
check	Validity of the Ticket	False
check	Barrier Closed	True

Table 11: Fit Table for *Check-out with Credit Card(Main Scenario)*

3.3 Buy Toll Tag

ActionFixture		
start	BuyTollTagFixture	
enter	Name	"Mock Name"
enter	Vehicle Type	[Car]
enter	License Plate Number	[KJK-420]
enter	Credit Card Number	[1234-5678-1234-5678]
check	Successful Payment	True

Table 12: Fit Table for *Buy Toll Tag (Main Scenario)*

ActionFixture		
start	BuyTollTagFixture	
enter	Name	"Mock Name"
enter	Vehicle Type	[Car]
enter	License Plate Number	[KJK-420]
enter	Credit Card Number	[1234-5678-1234-5678]
check	Successful Payment	False

Table 13: Fit Table for *Buy Toll Tag (Alternative Scenario)*

3.4 Check-in with Toll Tag

ActionFixture		
start	CheckInWithTollTagFixture	
enter	Antenna Scanned Toll Tag	[id = 1]
check	Validity of the Toll Tag	True
check	Barrier Opened	True
enter	Vehicle Passed	True
check	Barrier Closed	True

Table 14: Fit Table for *Check-in with Toll Tag(Main Scenario)*

ActionFixture		
start	CheckInWithTollTagFixture	
enter	Antenna Scanned Toll Tag	[id = 1]
check	Validity of the Toll Tag	False
check	Barrier Closed	True

Table 15: Fit Table for *Check-in with Toll Tag(Alternative Scenario)*

3.5 Check-out with Toll Tag

ActionFixture		
start	CheckOutWithTollTagFixture	
enter	Antenna Scanned Toll Tag	[id = 1]
check	Toll Tag Checked-In	True
check	Barrier Opened	True
enter	Vehicle Passed	True
check	Barrier Closed	True

Table 16: Fit Table for *Check-out with Toll Tag(Main Scenario)*

ActionFixture		
start	CheckOutWithTollTagFixture	
enter	Antenna Scanned Toll Tag	[id = 1]
check	Toll Tag Checked-In	False
check	Barrier Closed	True

Table 17: Fit Table for *Check-out with Toll Tag(Alternative Scenario)*

3.6 Pay Overdue Bill

ActionFixture		
start	PayOverdueBillFixture	
enter	Bill to Paid	[id = 3]
enter	Request Payment	Credit Card Number, Name, Date, CVC
check	Successful Payment	True

Table 18: Fit Table for *Pay Overdue Bill (Main Scenario)*

ActionFixture		
start	PayOverdueBillFixture	
enter	Bill to Paid	[id = 3]
enter	Request Payment	Credit Card Number, Name, Date, CVC
check	Successful Payment	False

Table 19: Fit Table for *Pay Overdue Bill (Alternative Scenario)*

3.7 Change Rate

ActionFixture		
start	ChangeRateFixture	
enter	Ticket Type	[Single Ticket]
enter	Vehicle Type	[Car]
enter	New Price	[45]
check	Single Ticket Price == 45	True

Table 20: Fit Table for *Change Rate (Main Scenario)*

ActionFixture		
start	ChangeRateFixture	
enter	Ticket Type	[Single Ticket]
enter	Vehicle Type	[Car]
enter	New Price	[45]
check	Single Ticket Price == 45	False

Table 21: Fit Table for *Change Rate (Alternative Scenario)*

4 Design

Responsible Student: Tamas, Nikolaos

The modelling of the toll system continues with its design phase, which is regarded from three different perspectives. Section 4.1 Component Design, which focuses on identifying the components and their ports of communication, section 4.2 Class Design, where we define the classes needed to implement the before mentioned components, and finally section 4.4 Behaviour Design. This last part describes the behaviour of the found classes using sequence diagrams. It is important to note that the following design choices only concern the seven chosen use cases.

4.1 Component Design

Responsible Student: Cristina, Dinesh

The selected use cases led to the identification of several components and sub-components, as can be observed in figure 4. The main component is the *Toll System*, which consists of a *Toll Lane*, a *Toll Station* and an *Enterprise*. Furthermore, each of these components contains a few others sub-components, depending on the needed functionality.

The communication between these reusable software units is ensured through the use of ports accompanied by connectors. The naming of the ports has been made in a suggestive way of the relationship between the two components. For instance, it is easily observable that the *Toll Lane Computer* component is connected to few other components (5 at least at first sight: *Antenna*, *Single Ticket Reader*, *Credit Card Reader*, *Barrier* and *Printer*). For clarity, we define the name of the ports as a combination of two things: the initials of the component to which it belongs and the initials of the components to which it connects. Thus, for illustrating a relationship between the *Toll Lane Computer* and *Single Ticket Reader*, we will use the ports/interfaces TlcStr and StrTlc.

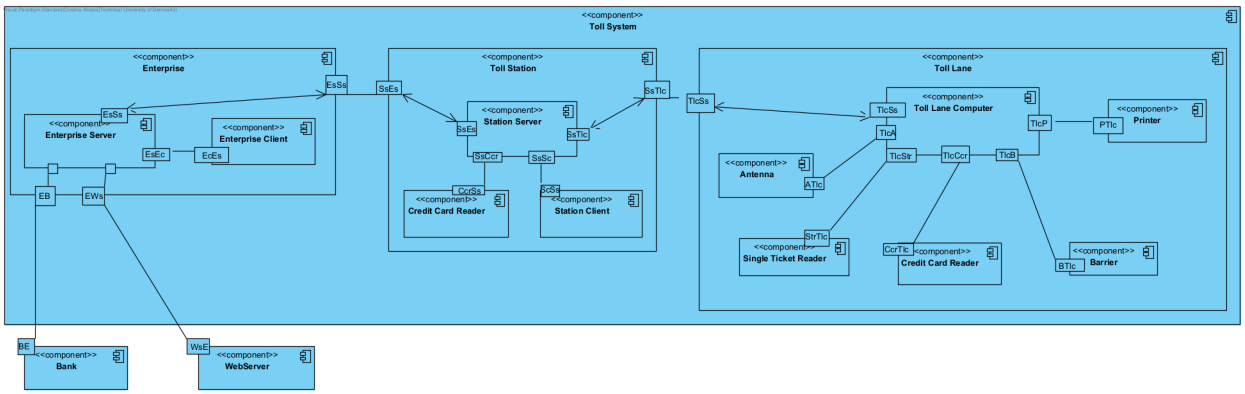

Figure 4: Component Diagram

Also, is observed that despite the fact that the 3 main components *Enterprise*, *Toll Station* and *Toll Lane* have ports for communication, these ports are just ports of their sub-components. Con-

ceptually, the ports EeSs on the *Enterprise Server* and EeSs on the *Enterprise* are the same. If we wanted to hide information about how the component *Enterprise* is implemented, we could simply name the port on the *Enterprise* as ESs. But that would still just be the second name of EeSs. For simplicity, we don't give second names to the ports.

Finally, it is worth noting that the *Enterprise Server* inside the *Enterprise* component is the only component having a connection outside the system boundaries. For that, we mention that the interfaces of the *Bank* and *WebServer* will be defined, but no further modelling will concern them.

As ports are merely interfaces for communication, we add additional diagrams in order to show the provided and required methods for each component. Figure 5 shows the the provided and required interfaces for the components residing inside *Toll Lane*. Not all components have both provided and required interfaces. For instance, the *Toll Lane Computer* component provides the interface *TollLaneComputerToAntenna*, which the *Antenna* component can use. However, the *Antenna* component does not provide any interface, and therefore the *Toll Lane Computer* component cannot call methods on the *Antenna*. This is a one-way communication. The idea behind modelling it this way is that *Antenna* will use the method *tollTagDetected(tollTagID : int)* whenever a car reaches the top of the queue of an express lane. As we don't need to signal back anything to the *Antenna* component, no interface is present.

An example where both interfaces are needed is between the communication of a *Toll Lane Computer* and a *Barrier*. The *Toll Lane Computer* component uses methods of its required interface *BarrierToTollLaneComputer*. For instance, when the *Toll Lane Computer* decides to open the barrier for a car, it will use the method *open(secondsToWait : int)* inside *BarrierToTollLaneComputer*. This method is implemented by the *Barrier* component. The implementation of the *Barrier* is out of scope. However, according to the contract, we expect that after the sensors of the barrier detected that the first car passed through, the *Barrier* will use the callback method *vehiclePassed()* on the *TollLaneComputerToBarrier* in order to signal that the vehicle has passed. If no callback is received from the *Barrier* within *secondsToWait*, the *TollLaneComputer* can just call *close()* on the *Antenna*.

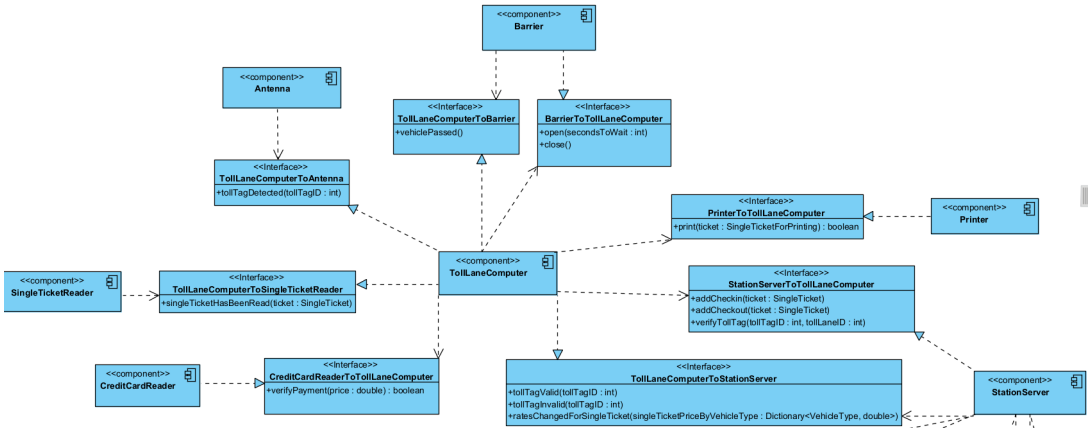

Figure 5: Component Diagram with interfaces - Toll Lane

Similarly, figure 6 shows the provided and required interfaces for the components residing inside the *Toll Station* component. Interesting to see here is the relationship between *Station Server* and *Enterprise Server*. The *Station Server* can use the method *verifyTollTagEntry(tollTagID : int, stationID : int, tollLaneID : int)* on the required interface *EnterpriseServerToStationServer* to ask the *Enterprise Server* to check the validity of a specific toll tag. This is an async call, to which the *Enterprise Server* can reply with one the callback methods *tollTagValid(tollTagID : int, stationID : int, tollLaneID : int)* or *tollTagInvalid(tollTagID : int, stationID : int, tollLaneID : int)*.

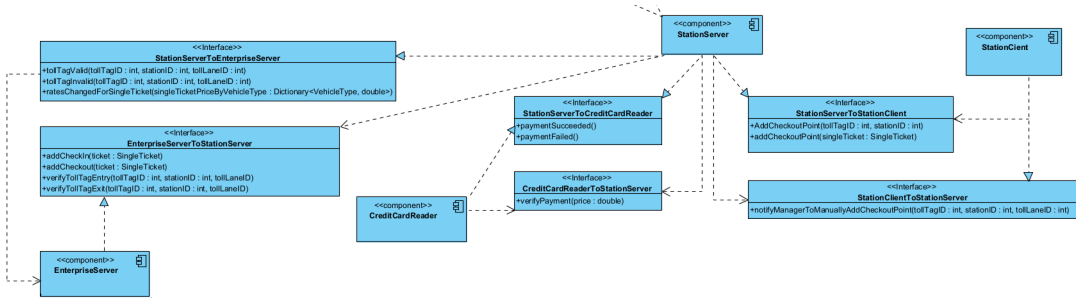

Figure 6: Component Diagram with interfaces - Toll Station

Lastly, figure 7 shows the provided and required interfaces for the components residing inside the *Enterprise* component. While on the web application, the customer can choose to pay overdue bills, if any are present. The *Web Application* component will use the method *payBill(customerID : int, billID : int)* provided by *EnterpriseServerToWebApplication*. The *Enterprise Server* can then send one of the two callback methods provided by the *WebApplication*: *payOverdueBillFailed(customerID : int)* or *payOverdueBillSucceeded(customerID : int)*.

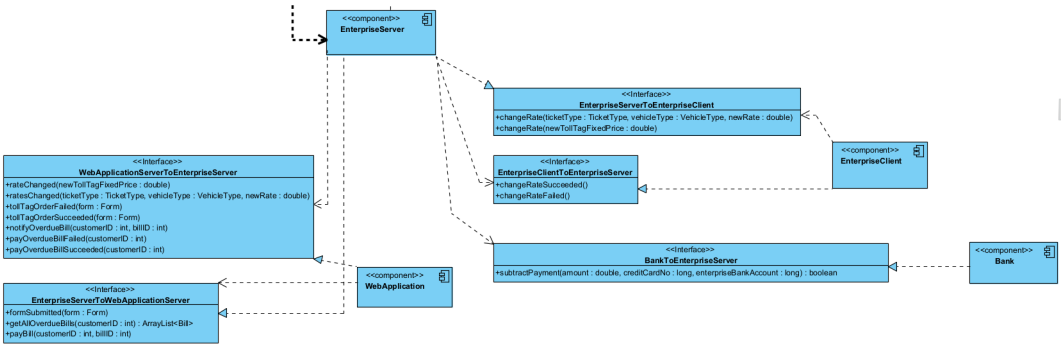

Figure 7: Component Diagram with interfaces - Enterprise

4.2 Class Design

Responsible Student: Roza, Cristina

A holistic view of the detailed class design is presented in Figure 8, it consists of three main packages: *Toll Lane* to the right, *Toll Station* in the middle, *Enterprise* to the left and finally a *Shared Library* in the bottom.

This shared library and all the other three packages are going to be described in detail in the following sections.

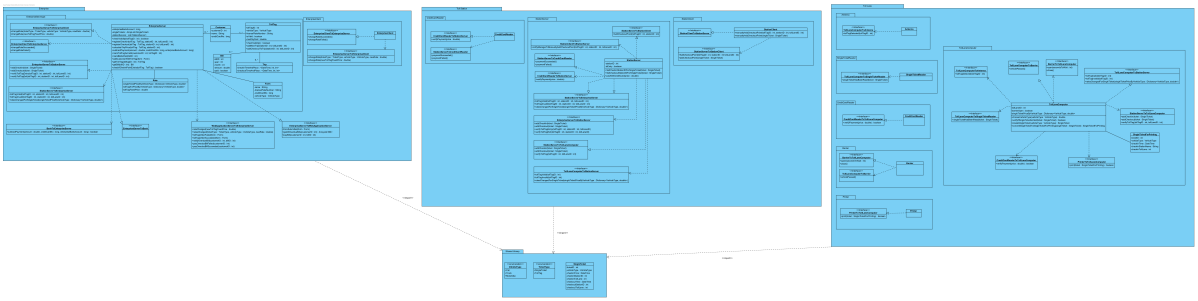

Figure 8: Detailed class diagram - holistic view

Starting with the *Shared Library* which consists of some classes that are shared between all the three packages in the detailed class design, therefore the relationship between those packages and the *Shared Library* is *import*, there are three different shared classes in the *Shared Library*, two of them are related to single tickets, one of them is a class called *SingleTicket* which contains all the attributes that are saved in the single ticket database on the *Enterprise* side, those attributes are: *ticketID*, three attributes for the check-in point called: *checkinTime* of type DateTime, *checkinStationID* and *checkinTollLaneID*, in addition to that, three other similar attributes are created but this time for the check-out point. Finally, this class consists of one more attribute called *vehicleType*

of type *VehicleType* which is another *enumeration* class in the shared library, that indicated that a vehicle can be one of three types: car, truck or motorbike.

The third final class in the shared library is again an enumeration class called *TicketType*, which indicated that a ticket can be one of two types: single ticket or toll tag.

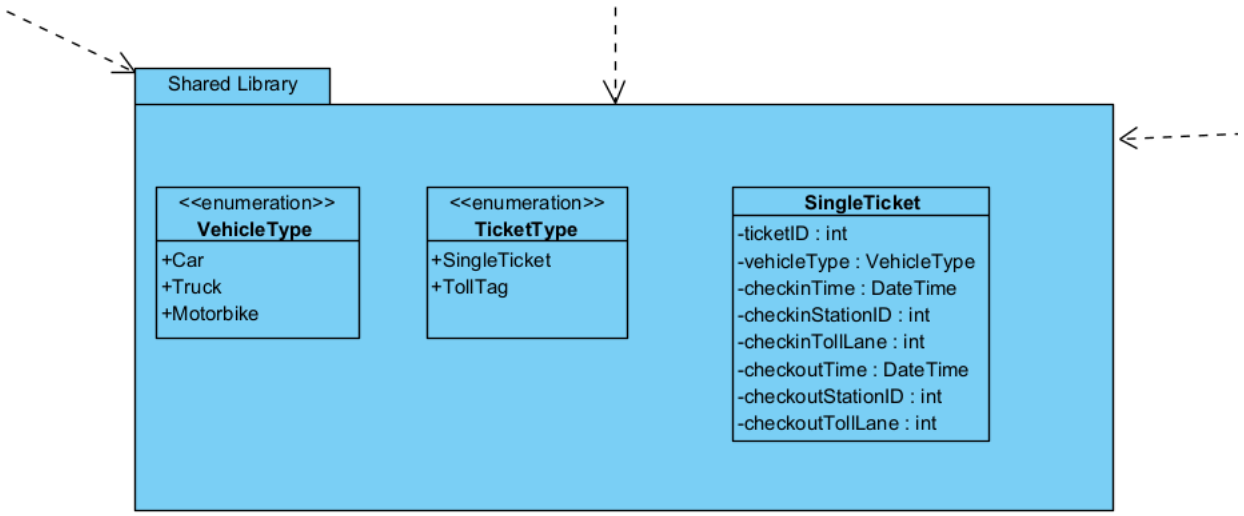

Figure 9: Detailed Class Diagram: Shared Library

4.2.1 Toll Lane

Responsible Student: Roza, Cristina

The *Toll Lane* package comprises six different packages that are needed for the check-in and check-out processes at a specific toll lane.

Each toll lane has a toll lane computer that has a touch screen connected to it, however, all the actions and methods that can be executed through the touch screen are shown in the *TollLaneComputer* class since the touch screen can be considered as a component of the toll lane computer.

This toll lane computer is interacting with all the other interfaces inside the *TollLane* package at some point depending on the use case. The toll lane computer holds information about the *tollLaneID*, the state of the barrier whether it is open or closed and finally the price of a single ticket for each vehicle type.

In addition to that, the toll lane computer package has an instantiated class called *SingleTicketForPrinting* that indicates the data that needs to be printed on a single ticket.

Finally, The toll lane computer is communicating to the *TollStation* package through two interfaces: *TollLaneComputerToStationServer* and *StationServerToTollLaneComputer*, implemented and used by the toll lane computer respectively.

All of these packages are interacting with the *TollLaneComputer* The first package called *Antenna*, that contains one interface implemented by the *Toll Lane Computer*.

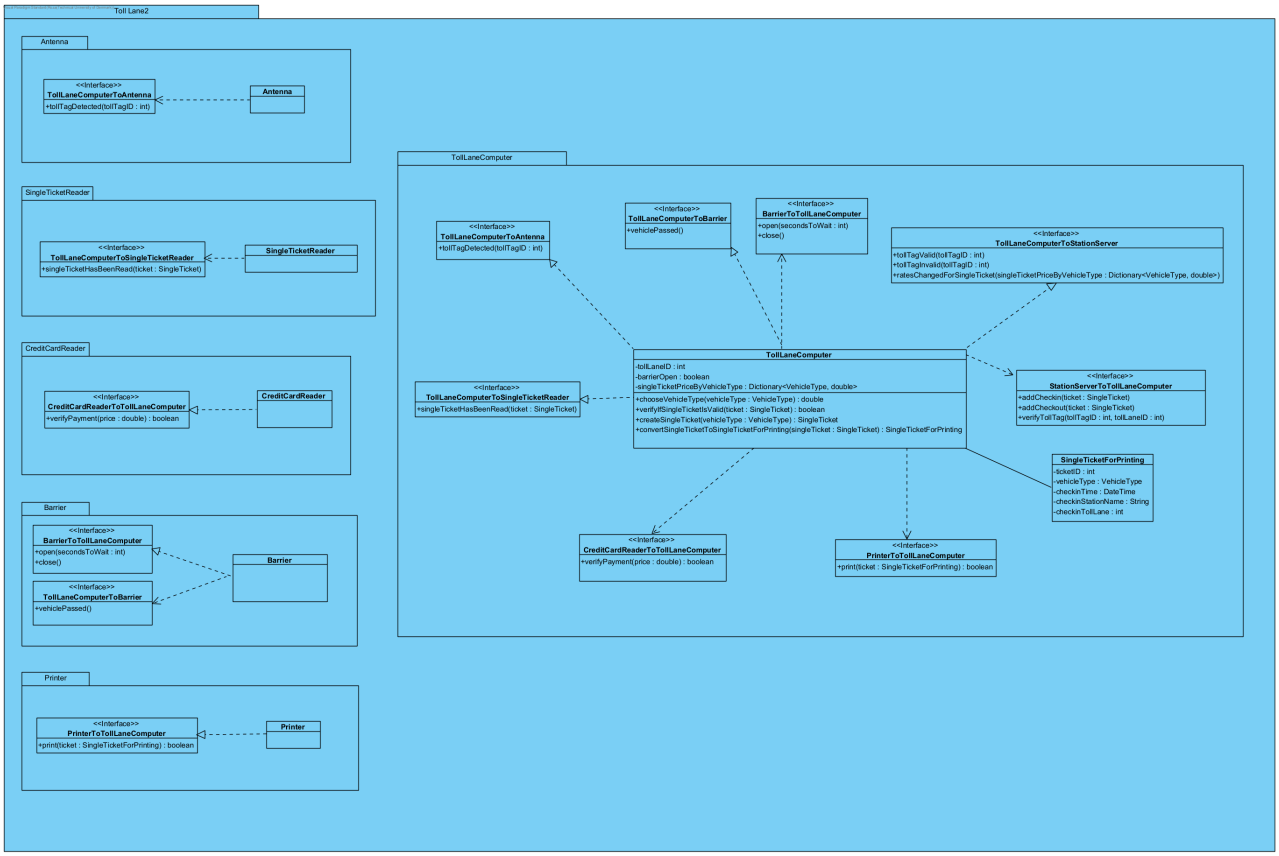

Figure 10: Detailed Class Diagram: Toll Lane

4.2.2 Toll Station

The *TollStation* package comprises 3 other packages, namely *CreditCardReader*, *StationClient* and *StationServer*. The latter contains the *StationServer* class, which has a *stationID* and *isEntry* attribute. The *StationServer* serves as a node between the TollLane Computers and Enterprise Server.

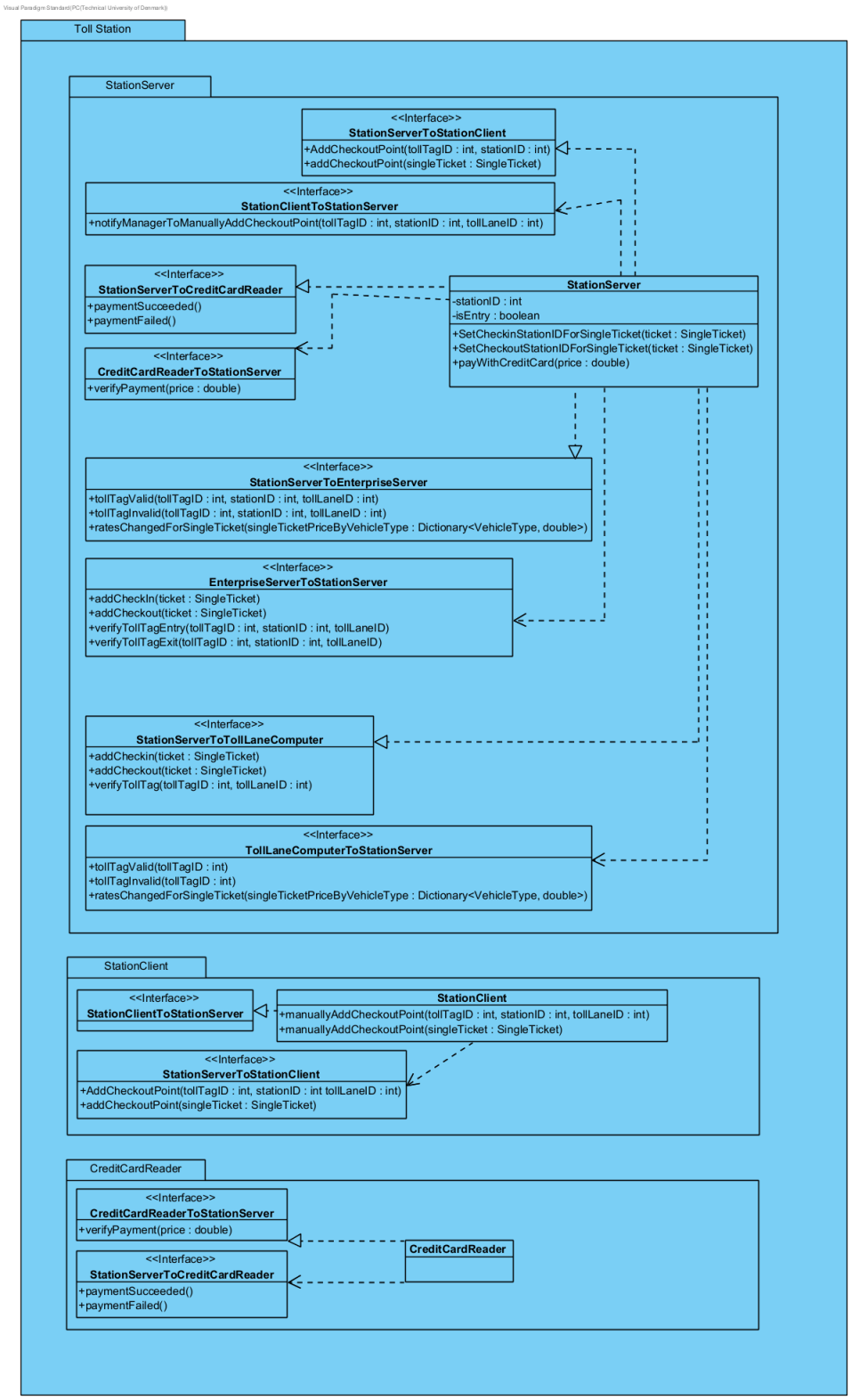

Figure 11: Detailed Class Diagram: Toll Lane

4.2.3 Enterprise

Responsible Student: Tamas, Dinesh

The *Enterprise* package comprises two other packages, namely *EnterpriseClien* and *StationServer*. The latter contains the *EnterpriseServer* class, which has the *enterpriseBankAccount*, *stationServers* List and the *singleTickets* List. Moreover, the Enterprise Server has a list of all the customers and is associated with the Rate Class. Every customer can have zero to many *TollTag*s, which has *Bill*s and *Trip*s. The *Customer*s can register by the submission of the *Form* object. The *EnterpriseServer* is the main class of our system, it keeps track of all the customers, trips, rates, handles the monthly deduction and communicates with the *WebApplication*. Only this class has a communication line with the *EnterpriseClient*.

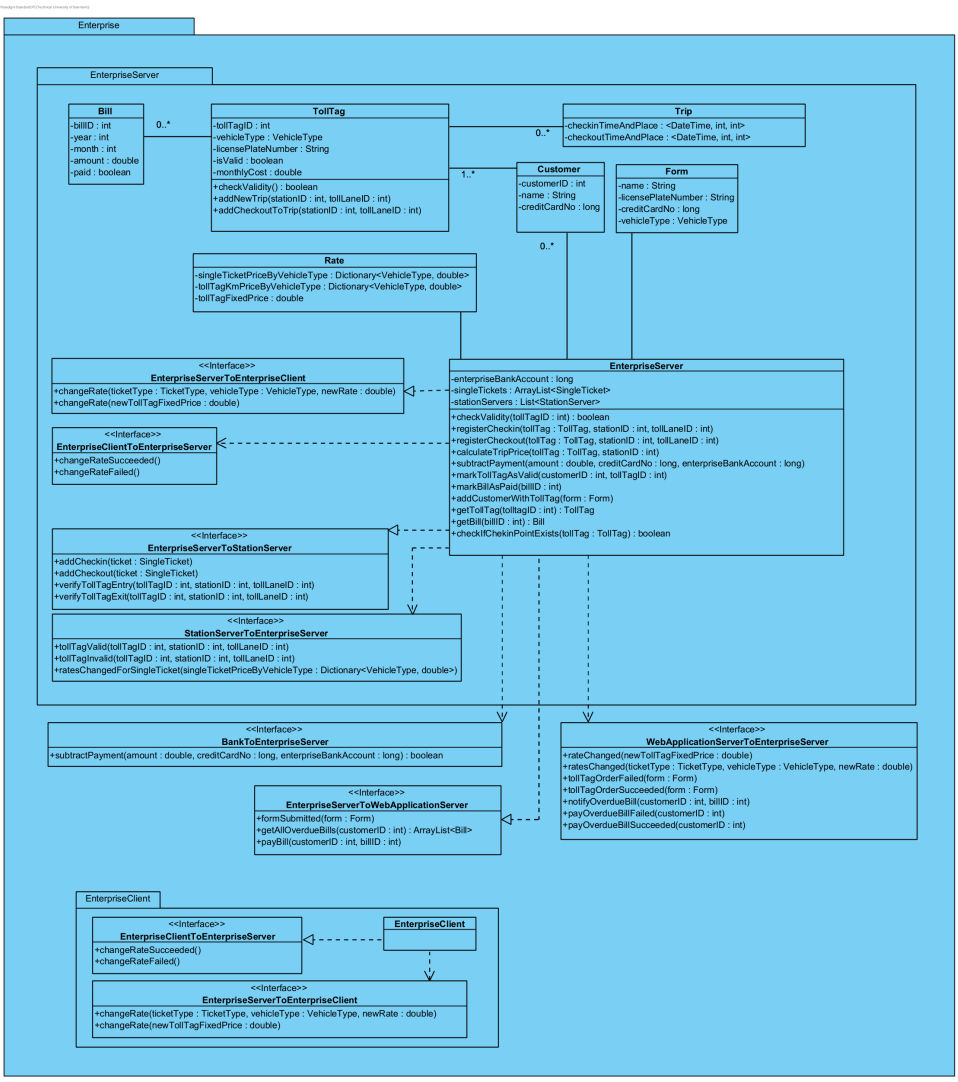

Figure 12: Detailed Class Diagram: Enterprise

4.3 OCL Constraints

Responsible Students: Dinesh, Eleni

From the methods outlined throughout the report, fourteen were randomly selected to be written as OCL constraints for this section.

context **EnterpriseServer :: registerCheckout**(tollTag : TollTag, stationID : int, tollLaneID : int)
pre: tollTag<> null and stationID<> null and tollLaneID<> null
post:tollTag.addCheckoutToTrip(stationID, tollLaneID) and
ss.tollTagValid(tollTagID, stationID, tollLaneID)

context **EnterpriseServer :: registerCheckin**(tollTag: TollTag, stationID: int, tollLaneID: int)
pre: tollTag<> null and stationID<> null and tollLaneID<> null
post:tollTag.addNewTrip(stationID, tollLaneID) and ss.tollTagValid(tollTagID, stationID, tollLaneID)

context **EnterpriseServer: getTollTag**(tollTagID : int) : TollTag
body: tollTag -> select(id=tollTagID)

context **EnterpriseServer: verifyTollTagExit**(tollTagID: int, stationID: int, tollLaneID: int)
pre: tollTagID<>null and stationID<>null and tollLaneID<>null
post: tollTag = getTollTag(tolltagID) and
chekinExist = checkIfCheckinPointExist(tollTag)

context **EnterpriseServer: verifyTollTagEntry**(tollTagID: int , stationID: int, tollLaneID: int)
pre: tollTagID<>null and stationID<>null and tollLaneID<>null
post: tollTag = getTollTag(tolltagID) and
tollValid = tollTag.checkValidity(tollTagID)

context **EnterpriseServer: addCheckout**(ticket : Ticket)
pre: ticket<> null
post: oldTicket = singleTickets.Where(t => t.ticketID == ticket.ticketID)

context **EnterpriseServer: formSubmitted**(form: Form)
pre: form <> null
post: paymentOK = bank.subtractPayment(amount, creditCardNo, enterpriseBankAccount)

context **TollLaneComputer :: tollTagDetected**(tollTagID)
pre: tollTagID <> null and tollLaneID <> null
post: ss.verifyTollTag(tollTagID,tollLaneID)

context **TollLaneComputer :: convertSingleTicketToSingleTicketForPrinting**(singleTicket)
pre: singleTicket <> null
post: ticketForPrinting:=convertSingleTicketToSingleTicketForPrinting(singleTicket) and p.print(ticketForPrinting)

context **TollLaneComputer :: verifyIfSingleTicketIsValid**(ticket)
pre: ticket <> null and
singleTicketValid := (time.now() - ticket.checkinTime).hours < 24
post: ss.addCheckoutvalid(ticket) and br.open(20)

Context **TollTag :: addCheckoutToTrip**(stationID, tollLaneID)
pre : stationID <> null and tollLaneID <> null
Post: trips.last.chekoutTimeAndPlace = (time.now(), stationID, tollLaneID)

Context **TollTag :: addNewTrip**(stationID, tollLaneID)
Pre: stationID <> null and tollLaneID <> null
Post: trips.add(new Trip(time.now(), stationID, tollLaneID, 0, 0, 0)

Context **StationServer :: addCheckout**(ticket)
Pre: ticket <> null
Post: setCheckoutStationIDForSingleTicket(ticket) and es.addCheckout(ticket)

Context **StationServer :: tollTagInvalid**(tollTagID, stationID)
Pre: tollTagID <> null and stationID <> null
Post: tlc.tollTagInvalid(tollTagID)

4.4 Behaviour Design

Responsible Students: Eleni, Cristina
In the following subsections are modelled the behaviour diagrams classes of the toll system.

4.4.1 Toll Lane Computer Component

The diagram below shows all the actions associated with single tickets, toll tags and vehicle type information. First a toll tag is detected by the antenna or a single ticket by the single ticket reader, then all the information provided should be checked and the ticket or toll tag should be valid in order for the barrier to open and the vehicle to pass. Additionally, the case for change rate is included.

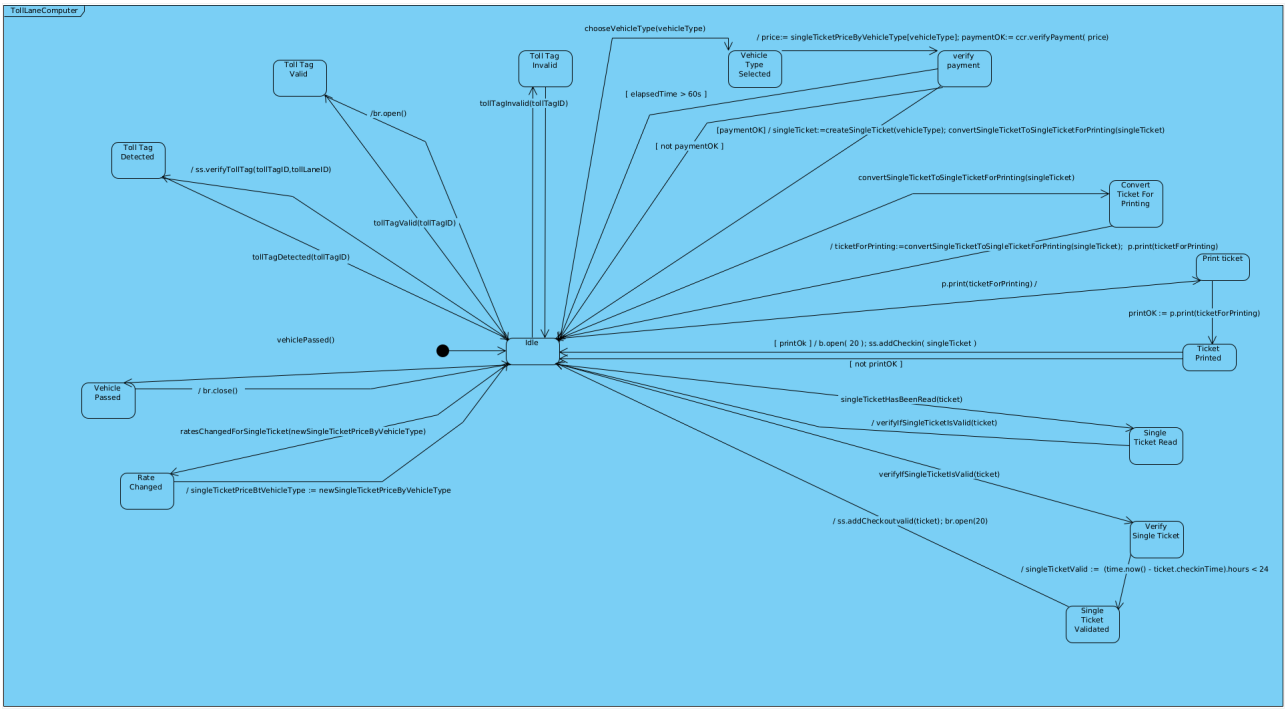

Figure 13: Toll Lane Computer Component (LSM)

4.4.2 Toll Tag Component

To illustrate how a new trip is added to a toll tag and how its validity is checked, the diagram below was created.

In order to check the validity of the toll tag, its attribute *isValid* is simply returned. When a customer checks in with a toll tag, the method *addNewTrip(stationID, tollLaneID)* is called on the toll tag. This method simply creates a new Trip object and adds it to the list of existing trips. It is important to note that when we create the new Trip object, we only set the attributes *checkinTime*, *checkinStationID* and *checkinTollLaneID*, but when the method *addCheckoutToTrip(stationID, tollLaneID)* is called, we simply set the remaining attributes *checkoutTime*, *checkoutStationID* and *checkoutTollLaneID*.

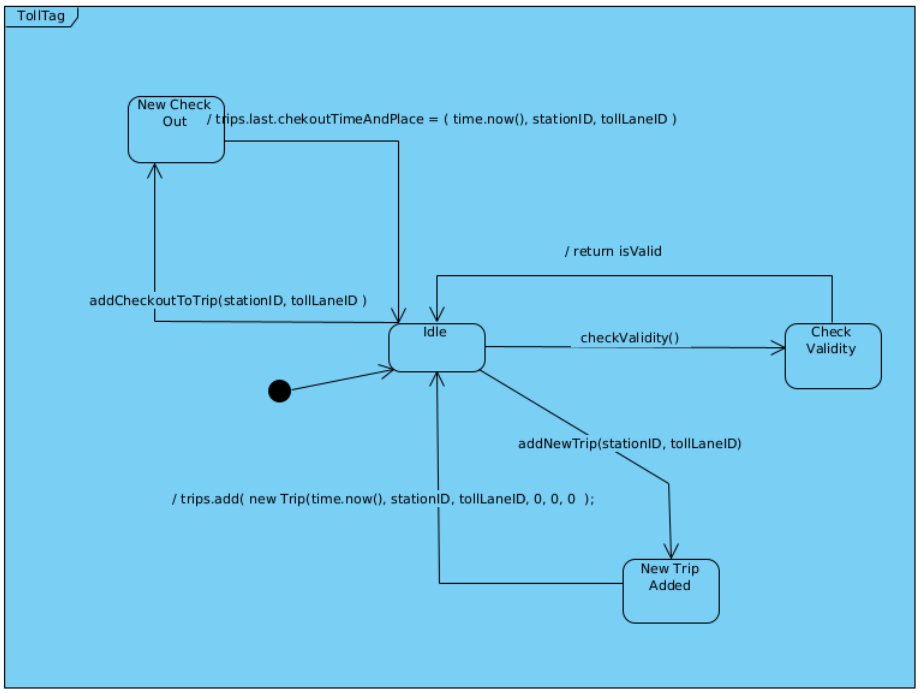

Figure 14: Toll Tag Component (LSM)

4.4.3 Station Server Component

The station server verifies each action performed in the toll lanes and checks the validity of each toll tag or single ticket through the check-in and check-out processes. These actions are formulated in this diagram.

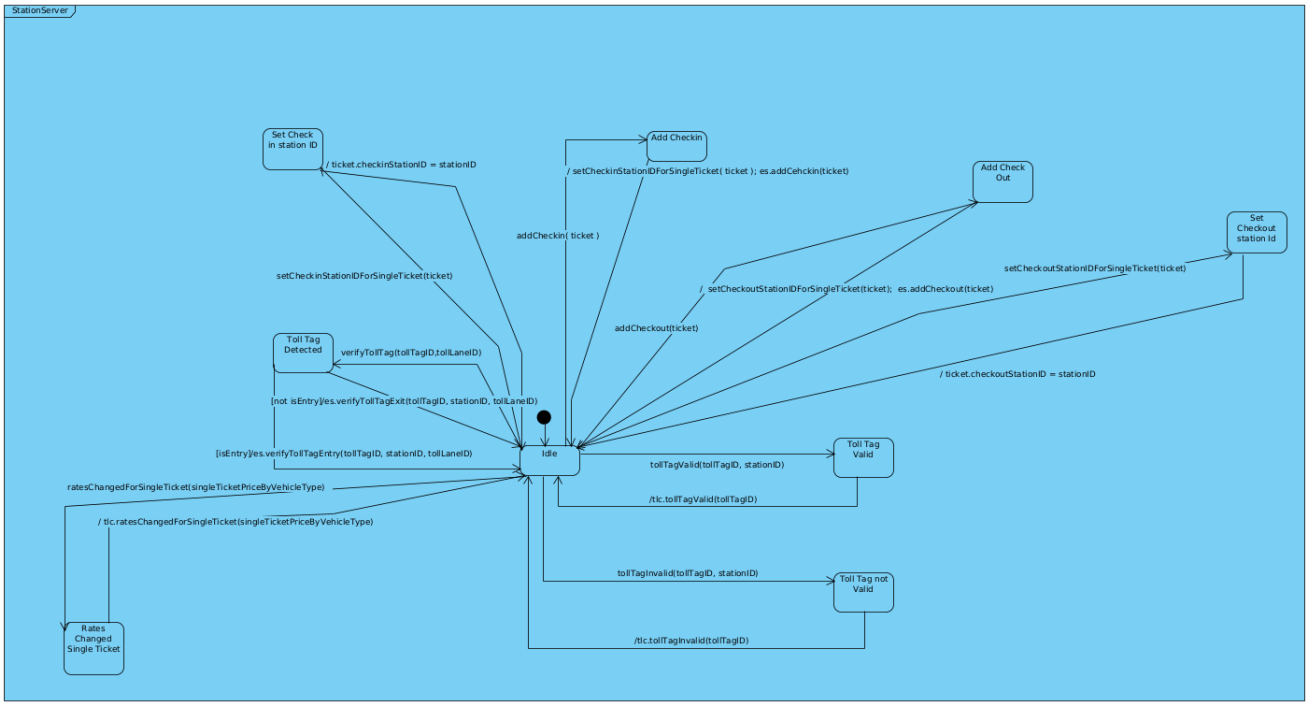

Figure 15: Station Server Component (LSM)

4.4.4 Enterprise Server Component

In order for the station server to correctly verify and transfer the answer to the toll lanes, needs first to receive the validation from the enterprise server. The enterprise server's actions are mainly to check the validity of toll tags or to add the check-in's or check-out's station ID to the toll tag's system.

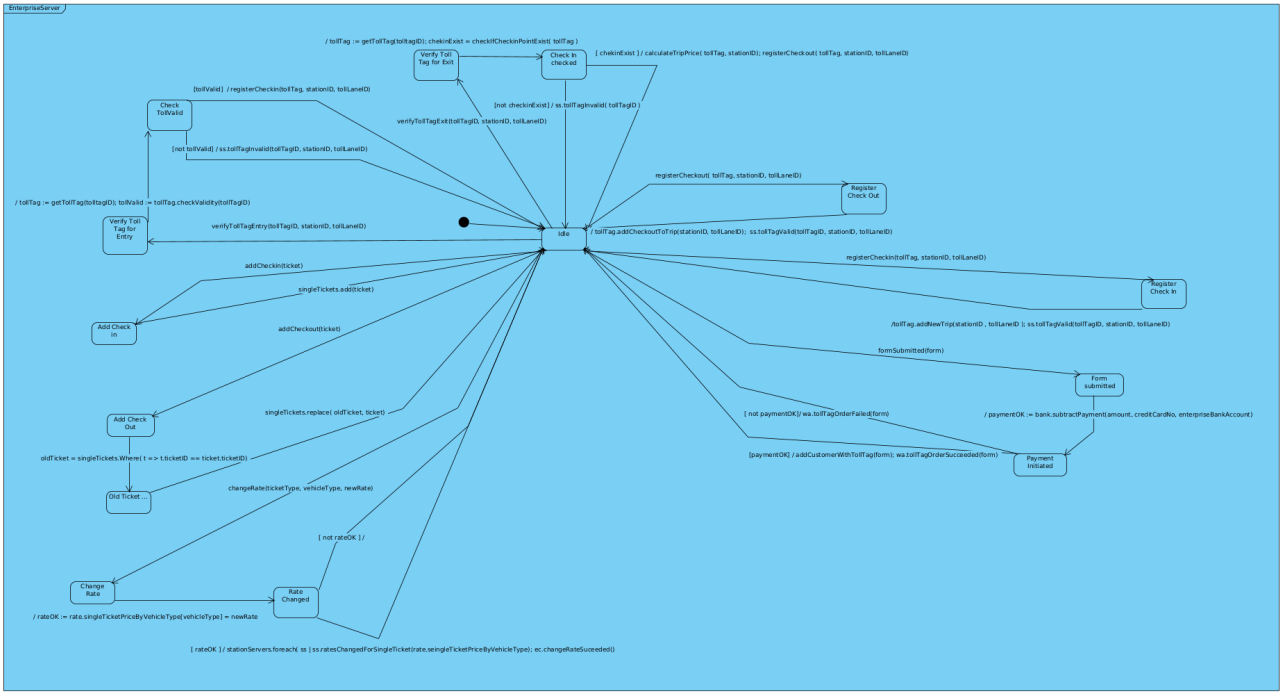

Figure 16: Enterprise Server Component (LSM)

4.4.5 Enterprise Client Component

The enterprise client is the computer that the enterprise manager is using for generating reports and changing rates. In the next diagram, the action of changing the rates is described. The enterprise client will delegate the work to the enterprise server es, from which might receive one of the callbacks *changeRateSucceeded()* or *changeRateFailed()*.

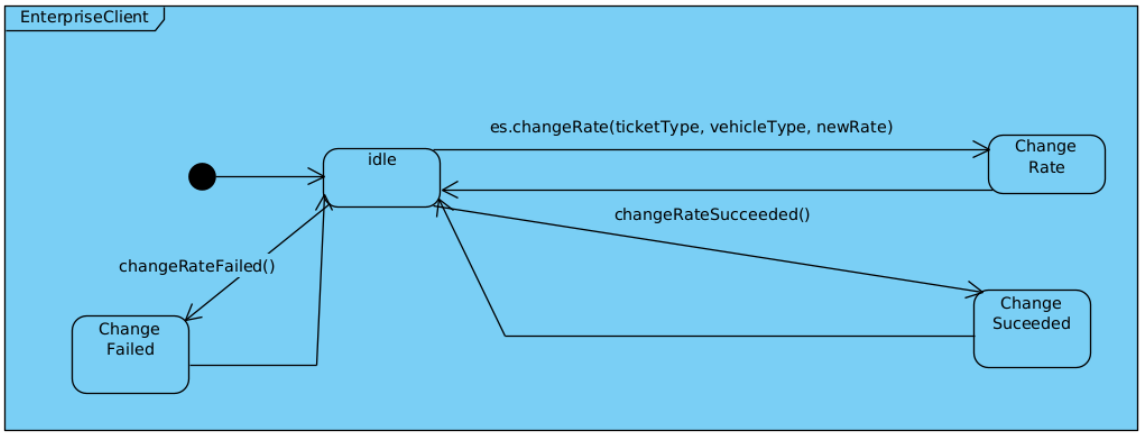

Figure 17: Enterprise Client Component (LSM)

5 Validation

5.1 Use Case Realisation

5.1.1 Check in with Credit Card

Responsible Student: Roza
Figure 18 below is a representation of the sequence diagram of check-in with credit card use case. The sequence starts with a customer advancing to a toll lane and using the touch screen that is connected to the toll lane computer for that toll lane for choosing the vehicle type e.g. "car", represented in the method from the *Customer* life line to the *TollLaneComputer* life line which calls this method on itself and returns the price of that ticket. Now the customer needs to pay using the *Credit Card Reader* that is connected to that toll lane computer. The *Toll Lane Computer* calls the method *verifyPayment(price: double): boolean* on the credit card reader asking for checking the validity of the credit card by returning a Boolean variable. In the case of a successful payment, the toll lane computer creates an object of that single ticket, but since the object of the single ticket that is represented in the shared library includes some information related to the check-out time, station and toll lane, another method *convertSingleTicketToSingleTicketForPrinting(singleTicket)* is called in order to make a new single ticket object which is similar to the above mentioned single ticket but with no information regarding the check-out point.

Afterwards, the toll lane computer communicates with the printer in order to print a single ticket with the information in the single ticket for printing. The print method returns a Boolean type again, in the case of a successful printing, two methods are executed in parallel, one of them is directed from the toll lane computer to the station server in order to add that new single ticket to the list of single tickets. Now the station server calls a method on itself *setCheckinStationIDForSingleTicket(ticket)* to add the stationID to the single ticket object. Afterwards, another call is directed from the station server to the enterprise server for the same purpose of adding the new created single ticket to the list of single tickets in the database on the enterprise side. The second method that was executed in parallel is a call from the toll lane computer to the barrier to open. This method here is synchronous, that is waiting for a response from the Barrier when a vehicle has passed. This action can happen since the barrier has a sensor for that purpose, once that method is sent, the toll lane computer is asking the barrier to close. In the case that no method is returned back from the barrier, the barrier is lowered automatically after some time out.
In the alternative scenario when the payment is failed, this sequence diagram is not going further and the customer is not allowed to enter the motorway.

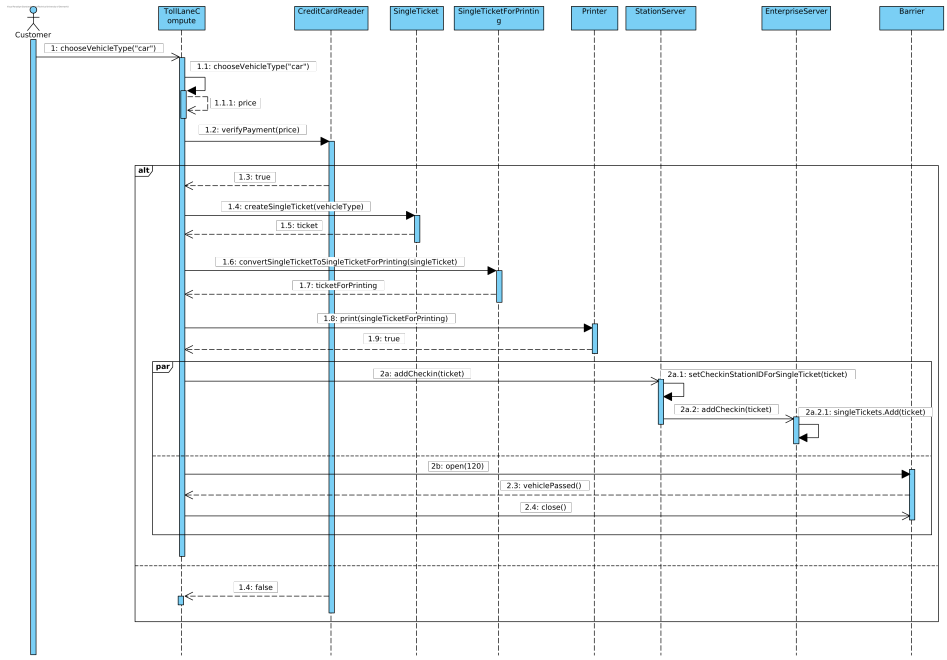

Figure 18: Sequence diagram: Check-in with Credit Card

5.1.2 Check out with Credit Card

Responsible Student: Eleni

Check out with Credit Card sequence diagram describes the process of checking out with a single ticket from a normal toll lane. First, the customer enters a normal lane and inserts the single ticket *insertTicket()* into the single ticket reader. After the single ticket is detected and read by the single ticket reader *singleTicketHasBeenRead(ticket)* a message is sent to the toll lane computer in order to check the validity of the ticket *verifyIfSingleTicketIsValid(ticket)*. If the single ticket is valid a message is forwarded from the toll lane computer to the station server requesting for the check-out time and place to be added to the ticket *addCheckout(ticket)*. Thus the station server adds the requested data to the ticket *setCheckoutStationIDForSingleTicket(ticket)* and sends a finalizing message to the enterprise server for the check out to be admitted into the system *addCheckout(ticket)*, and be classified as an old ticket *singleTickets.Where(t => t.ticketID == ticket.ticketID)*, *oldTicket*, *singleTickets.replace(oldTicket, ticket)*. The last step is for the toll lane computer to give the order to the barrier to open *open(120)* and stay in the open state for a maximum of 120 seconds while waiting for the vehicle to pass, then the barrier sends a confirmation to the toll lane computer *vehiclePassed()* and receives feedback for it to close again *close()*.

In case of an invalid single ticket, the customer has to buy a new single ticket. For this action, the station server communicates with the credit card reader asking for the condition of the payment *verifyPayment(price)* after it receives a successful reply *paymentSucceeded()* the rest of the process goes as in the previous paragraph.

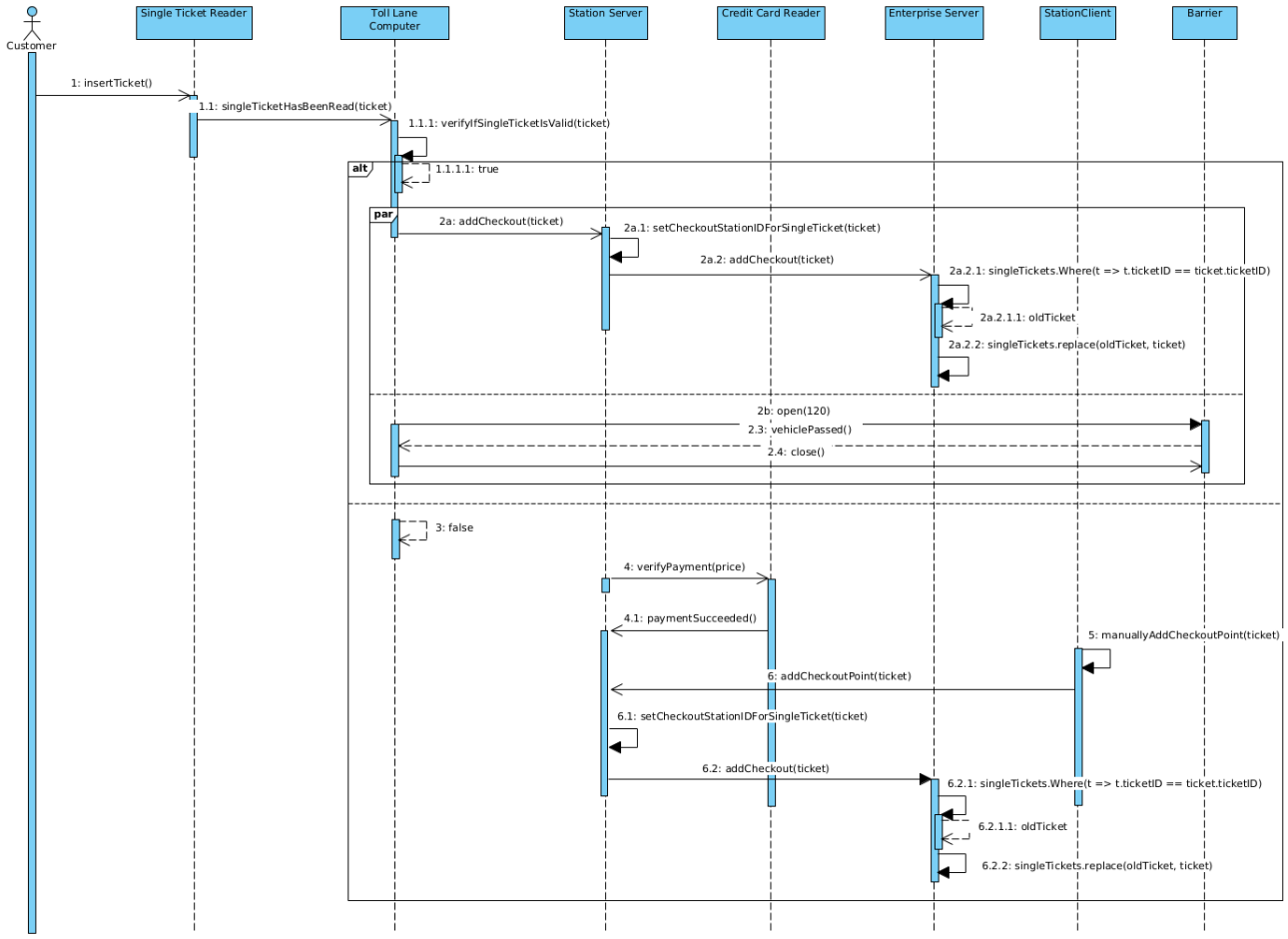

Figure 19: Sequence diagram: Check-out with Credit Card

5.1.3 Buy Toll Tag

Responsible Student: Tamas

The Buy Toll Tag sequence diagram describes the process, when the customer had logged in and loaded the Toll Tag buying page. Here the customer fills in the registration form and with the provided details, the *formSubmitted(form)* method is called on the Enterprise Server. After this the Enterprise Server calls the *subtractPayment(amount, creditCardNo, enterpiseBankAccount.* In case of successful payment the True returned value results the call of the *addCustomerWithTollTag(form)* and the process finishes with the *tollTagOrderSuceeded(form)*, called on the Web Application.

In case of unsuccessful payment the False returned value results the call of the *tollTagOrdeFailed(form)* on the Web Application. The form: Form object that with the following methods are called contains a correlation ID too, which makes the asynchronous response call possible.

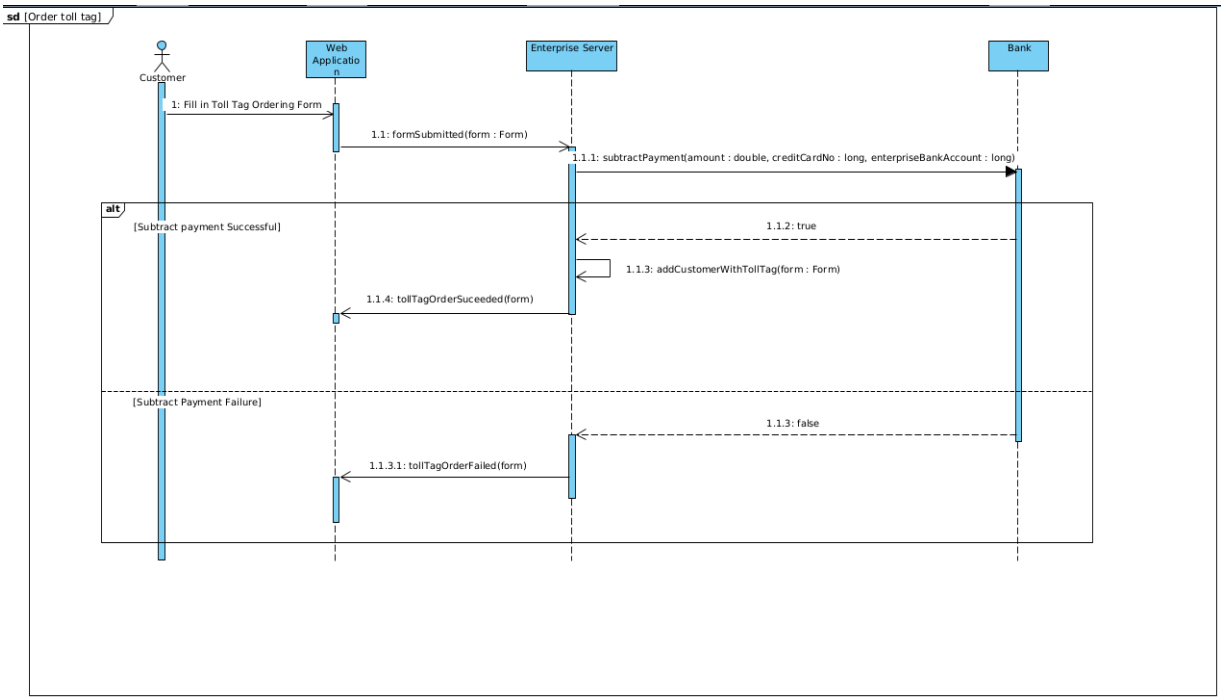

Figure 20: Sequence diagram: Buy Toll Tag

5.1.4 Check in with Toll Tag

Responsible Student: Nikolaos

Check in with Toll Tag sequence diagram demonstrate how the system operates in case of a check-in with a toll tag through an express toll lane. First, the vehicle moves forward to the express lane and waits for the antenna to scan the toll tag *tollTagDetected(tollTagID)* and forwards the information to the toll lane computer awaiting for a response. The toll lane computer resends the message to the station server *verifyTollTag(TollTagID,tollLaneID)* which redirects it to the enterprise server *verifyTollTagEntry(tollTagID , stationID, tollLaneID)*. Now the enterprise server has all the information needed *getTollTag(TollTagID)* to check the validity of the toll tag *checkvalidity(tollTagID)*. After the toll tag confirms its information, a positive response is sent back to the enterprise server which registers the check in *registerCheckin(tollTag, stationID, tollLaneID)* and authorizes a new trip to be added by the toll tag *addNewTrip(stationID, tollLaneID)*. At that point the enterprise server answers to the station server that the toll tag is valid *tollTagValid(tollTagID, stationID, tollLaneID)* which forwards the answer to the toll lane computer *tollTagValid(tollTagID)*. At last, the toll lane computer permits the barrier to open *open(120)* for a maximum of 120 seconds, while waiting for the vehicle to pass. Lastly, the barrier confirms to the toll lane computer that the vehicle has passed *vehiclePassed()* and then it is ordered to close *close()*. Oppositely, when a toll tag is found invalid, an invalid message is sent to both the station server *tollTagInvalid(tollTagID, stationID, tollLaneID))* and the toll lane computer *tollTagInvalid(tollTagID)* and the customer is can instead buy a single ticket in order to enters the motorway.

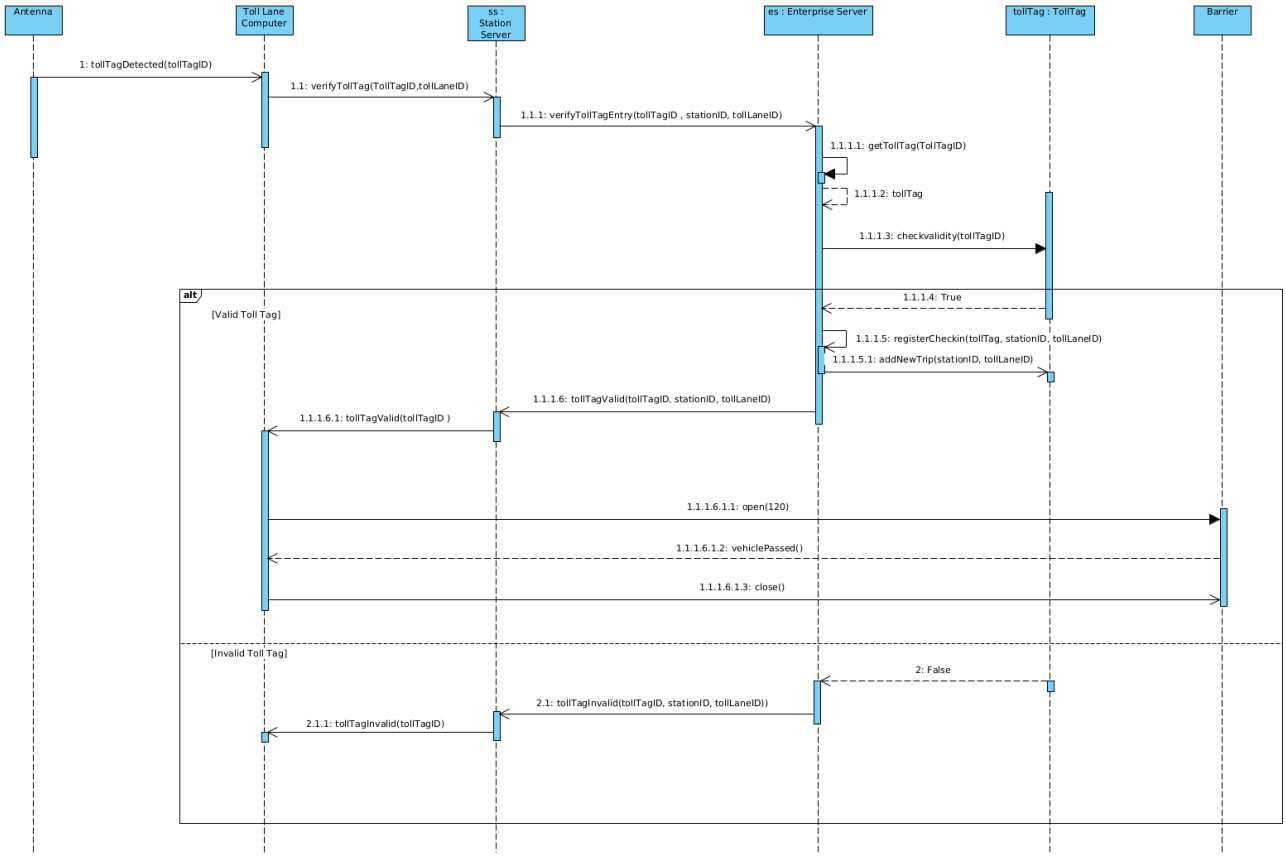

Figure 21: Sequence diagram: Check-in with Toll Tag

5.1.5 Check out with Toll Tag

Responsible Student: Dinesh

Check out with Toll Tag sequence diagram 22 illustrates how the system flows during checkout of the vehicle when it approaches to the checkout lane.

When a vehicle approaches the Toll Station, the Toll tag which is associated (registered) with the vehicle is scanned and detected *tollTagDetected(tollTagID)* by Antenna. When Toll lane computer receives the Toll Tag information from Antenna, it calls Station Server *verifyTollTag(tollTagID, tollLaneID)* to verify the detected Toll Tag. Station Server redirects *verifyTollTagExit(tollTagID, stationID, tollLaneID)*to Enterprise Server for further verification. Enterprise Server collects all the information about Toll Tag with *getTollTag(tollTagID)* and checks if there is any check in registered with current Toll Tag with *checkIfChekinPointExists(tollTag)*. If there is a registered checkin point then the cost for the travel is calculated using *calculateTripPrice(tollTag, stationID)*. After that checkout is officially registered as *registerCheckout(tollTag, stationID, tollLaneID)* and Toll Tag is updated by adding the checkout trip with *addCheckoutToTrip(stationID, tollLaneID)*. Finally, En-

terprise Server returns valid response with *tollTagValid(tollTagID)* to Toll Lane Computer through Station Server. Then, Toll Lane Computer will open the Barrier with *open(120)*. Where *120* is seconds to wait to close barrier. When the sensor attached to the Barrier detect that the vehicle has passed, it sends response to Toll Lane Computer with *vehiclePassed()* that vehicle is moved out, then finally Toll Lane computer close the Barrier with *close()* . Toll Lane Computer Waits 120s to get response from Barrier, after that it will close barrier, assuming that vehicle is moved out.

In the case where Enterprise Server can not find the registered check in point associated with the Toll Tag, it will return a invalid Toll Tag response to the Toll Lane Computer without calculating the trip price and without registering the checkout. And Cashier in the Toll Station will handle the situation. Finally, vehicle moves out.

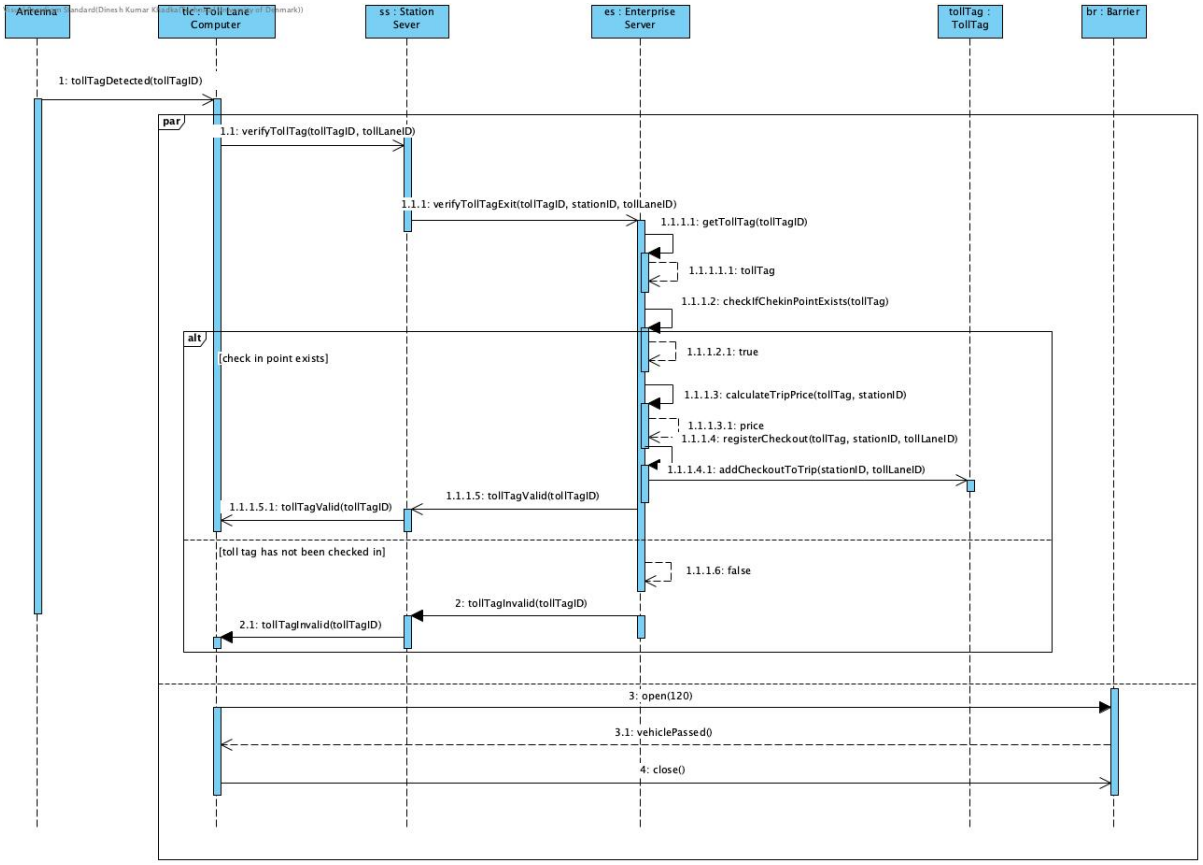

Figure 22: Sequence diagram: Check-out with Toll Tag

5.1.6 Pay Overdue Bill

Responsible Student: Johannes

Pay overdue bill sequence diagram illustrates the system flow for the pay overdue bill use case.

Whenever a monthly bill hasn't been paid by the customer, the enterprise server initiates a *notifyCustomerOverduBill(customerID, billID)* notification to the customers account at the web application. Whenever a customer logs into the system he/she can see the notification and send a *get all overdue bills* request to the web application whereas the web application in turn requests the enterprise server with *getAllOverdueBills(customerID)* for the customer in question. All the customers existing overdue *bills* objects will be sent to the web application UI whereas the customer can prompt to *pay bill*. This initiates a *payBill(customerID, billID)* request from the web application to the enterprise server which in turn sends a *subtractPayment(amount, creditCardNo, enterpriseBanAccount)* message to the bank which returns a boolean. Whenever this message returns *true*, the system recursively invokes the methods *markTollTagAsValid(tollTagID), getCustomer(customerID)* and *getTollTags()* where the system needs to fetch information regarding the customer with the Customer class in order to further fetch information on the toll tags. When all the *tollTags* are fetched and marked as valid the Enterprise server invokes *payOverdueBillSucceeded(customerID)* in order to provide a successful feedback to the web application.

In the alternative case where *(amount, creditCardNo, enterpriseBanAccount)* fails, the bank returns a *false* where the Enterprise Server enters the alternative scenario where *payOverdueBillFailed(customerID)* is invoked, responding to the web application that the payment was erroneous.

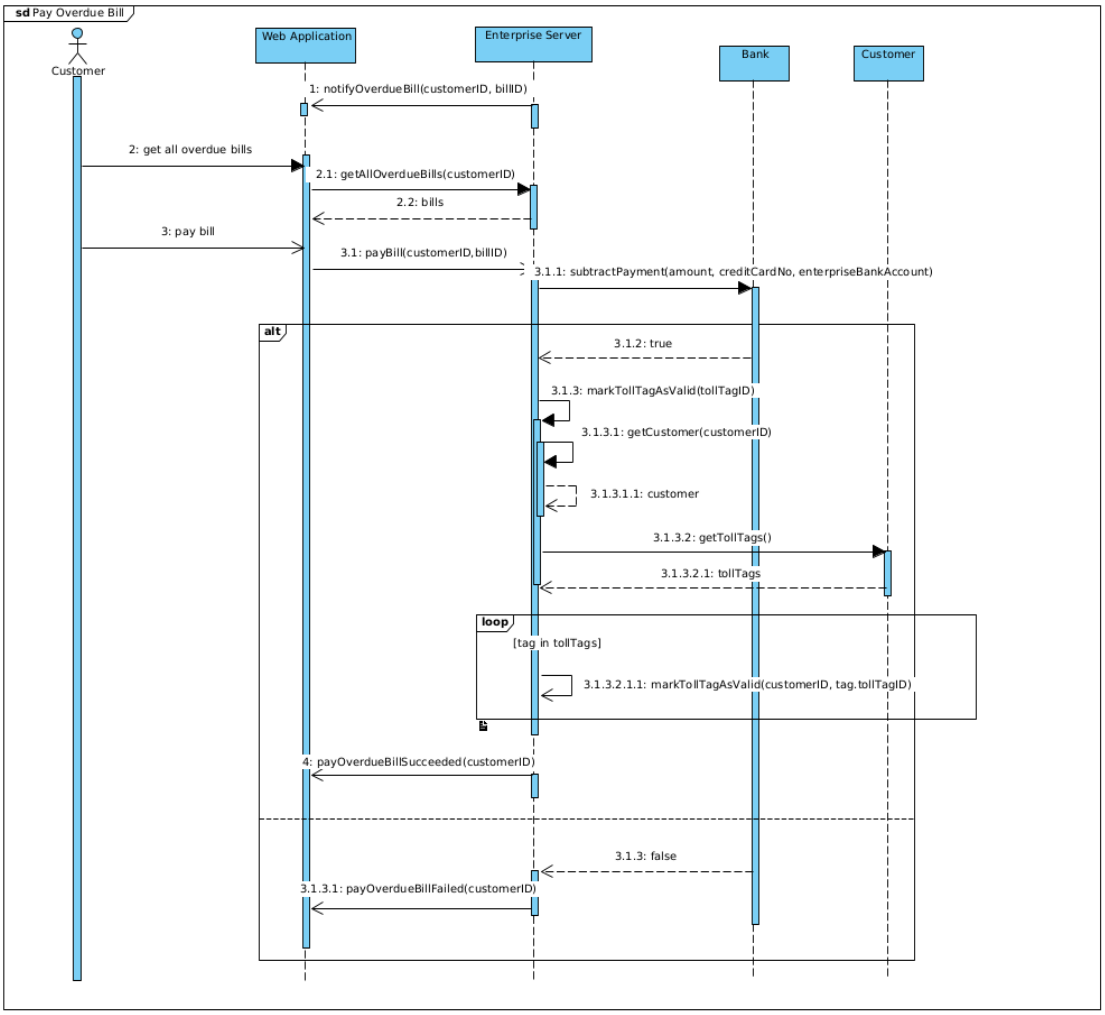

Figure 23: Sequence diagram: Pay Overdue Bill

5.1.7 Change Rate

Responsible Student: Cristina

Figure 24 presents the interactions of the objects included in the *Change rate* use case. Particularly, this sequence diagram captions the process of changing the rate for a single ticket.

To begin with, the *Enterprise Manager* uses the *Enterprise Client* to modify the rate for a single ticket corresponding to a specific vehicle type, let it be 'car', 'truck' or 'motorbike'. Then, the *Enterprise Client* calls *changeRate('SingleTicket', vehicleType, newRate)* on the *Enterprise Server*. The *Enterprise Server* implements this method simply by setting the attribute *singleTicketPrice-ByVehicleType* of its *rate* attribute to the new rate. Then, depending on the change raising any errors, we either call on the *Enterprise Server* the method *changeRateFailed*, in which case we stop, or *changeRateSucceeded*. If the rate has been changed successfully on the *Enterprise Server*,

this will notify all *Station Servers* at once, along with the *Web Application*. We decided to also notify the *Web Application*, since it might be nice to show the customers on the web application the price of a single ticket, initial price of toll tag, as well as price per km for toll tag. The notification process is done concurrently, as it can be seen in the diagram. For the sake of example, we assume that we only have 2 toll stations (st1 and st2) , each of which has a toll lane (tlc1 and tlc2). Now, the *Enterprise Server* calls in parallel the method *ratesChangedForSingleTicket(rate.singleTicketPriceByVehicleType)* on the *Station Servers*, which will also call in parallel *ratesChangedForSingleTicket(rate.singleTicketPriceByVehicleType)* on the *Toll Lane Computers*. As for the *Web application*, the *Enterprise Server* calls *ratesChanged(rate.singleTicketPriceByVehicleType)*.

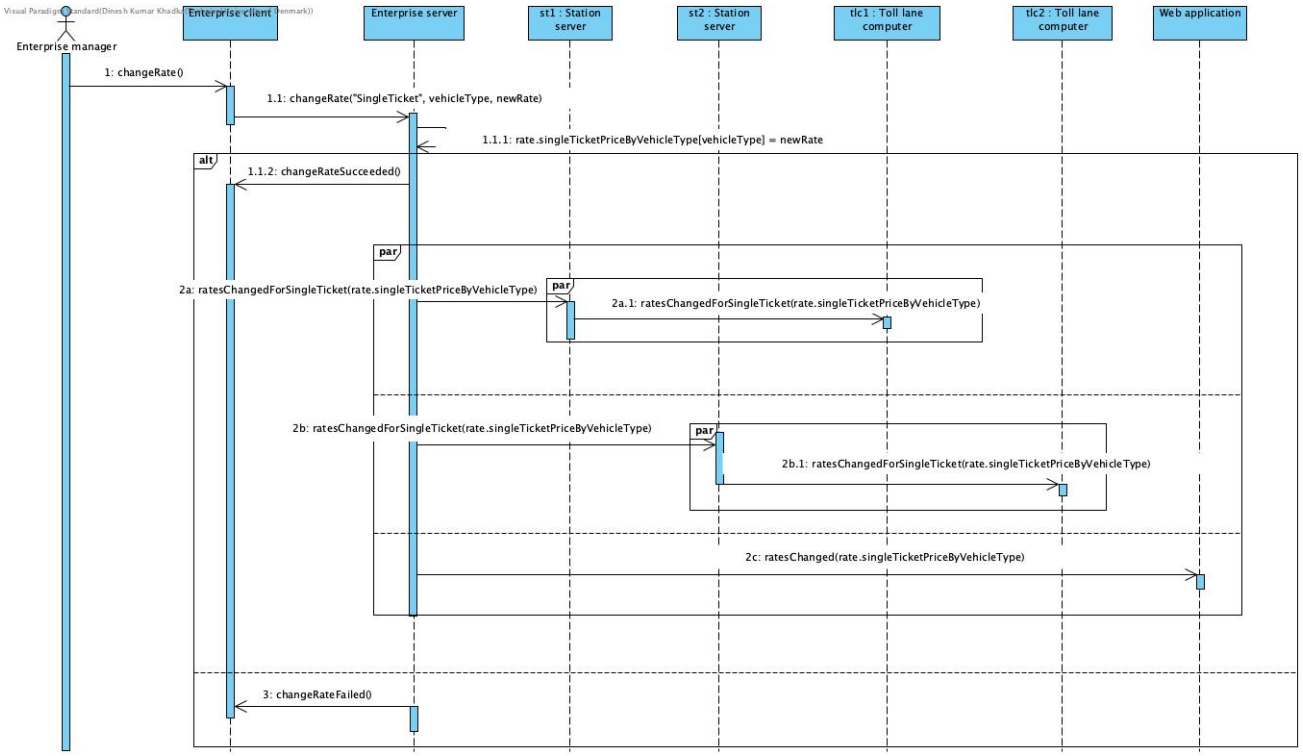

Figure 24: Sequence diagram: Change Rate

6 Conclusion

Responsible Student: Johannes, Roza

In the presented project we have created a confined model of the given toll system only considering the mentioned use cases. This model can act as basis for a software development scenario, as the required components and diagrams are defined, verified and validated in cohesion. Extending the model, thus creating a complete system model for a toll station would require adding multiple use cases, but this as-is model is sufficient to the chosen use cases.

6.1 Experience with the project

Working on the project in regards to the mentioned methodology has proven quite fruitful, both in terms of learning and efficient production. Iterative reassessing our work with agile, made us reach our goal in producing a decent model that fits the use cases.

7 Work Division

All the group members contributed equally and collaborated together in creating and implementing the diagrams and the methods for this project. The report sections are divided in the way based on who wrote the specific report section. However, it is worth mentioning that the whole report was reviewed by all members and the delivery happened after agreeing on the content.