TAXI SERVICE

RASD

Requirements Analysis and Specification Document

version 1.2

Rota Diego, 841344, 21 hours

Montalto Simone, 841359, 21 hours

Politecnico di Milano, A.A. 2015-2016

Software Engineering 2 – Prof.ssa Mirandola Raffaela

Sommario

1. Introduction

1.1 Overview

This document represents the Requirement Analysis and Specification Document (RASD), which goal is to have an high-level description of the system that will be designed. In RASD we will analyze functional and non functional requirements, the goals that must be reached, behavior of the real world (in order to model the constraints) and real use cases that involve the actors with the system. At the end of the document are attached some diagrams that help to understand how all the part interact one with the others. In particular, the diagrams are the UML, Sequence Diagram, State Chart and Use Case. To better model the system and the relative constraints, we have used Alloy. The code and relative output are included in the RASD. This document aims to show a clear description of the system that will be developed to the customer, and it can also help the future developers to understand the actual system in order to improve the functionality. RASD could be seen as a binding contract between the developer and the customer.

1.2 Scope

The scope of the project is to develop a system to manage and optimize the taxi service of a large city. In particular, a passenger can use a mobile or a web application to request a taxi and have information about the availability of a taxi in the request zone of the city and the waiting time. The taxi driver can manage their availability and answer at each request using the mobile application. To use the system, each actor, that could be a taxi driver or a passenger, must be registered and must be logged. The taxies are managed dividing the city in different zones, and each zone include a taxi queue. When a passenger make a request for a certain zone, the first taxi driver in the queue for the specific zone is contacted on the mobile app through a notification and have to answer to the request. If the taxi driver decline the request, the next driver in the queue is notified and so on until one taxi driver accept the request. The taxi driver that decline the request are moved at the end of the queue. The passenger will be notified when a taxi driver accept the request through a notification. When a taxi driver notify with the app that he is available, he is added at the end of the queue of the zone in which he is located through the GPS. Each passenger could

also make a reservation for a certain moment specifying the hour, the origin and the destination address. The passenger must made this reservation at least two hour before the ride, and a notification will be send to the passenger ten minutes before the ride to confirm the reservation. The system also provide programmatic interfaces (API) to give the possibility to an external developer to extend the functionality of the system. The external developers must be registered to use the API.

1.3 Vocabulary

- **Guest**: it is an user that is not yet registered;
- **Customer**: it is an user that is registered and correspond to the passenger;
- **Driver**: it is an user that is registered and correspond to the taxi driver;
- **External Developer**: it is an user that is registered and can only require API of the system;
- **User**: it could be a Guest, Customer, Driver or External Developer.
- **Vehicle**: correspond to the taxi car that is driven by a Driver;
- **Request**: it is a generic reservation made by a Customer;
- **Immediate Request**: it is a specific Request, and it is made by the Customer to require a taxi as soon as possible;
- **Booked Request**: it is a specific Request, and it is made by the Customer to book a taxi for a specific hour, origin and destination address;
- **Zone**: it indicates a specific area of the city that include only one queue.

1.4 Acronyms

- **RASD**: Requirement Analysis and Specification Document;
- **API**: Application Programming Interface;

1.5 Overview

This document is divided into five section in order to improve the readability:

- **Section 1 – Introduction**: it includes an overview of the system, the scope, the vocabulary and the acronyms used in the RASD.

- **Section 2 – Overall Description**: it includes constraints and preliminary assumptions on the concept that are unclear or missing.
- **Section 3 – Requirement Analysis**: it includes functional and non functional requirements and the eliciting of the actors.
- **Section 4 –Scenarios**: it includes some scenarios that includes functionality of the system.
- **Section 5 – Diagrams and Alloy**: it includes UML, Sequence Diagram, State Chart, Use Case and Alloy.

2. Overall Description

2.1 Constraints

1. **Unique username**: Two users cannot have the same username;
2. **License Mandatory**: Taxi Driver License is unique and mandatory for Driver;
3. **Unique License**: Two Drivers cannot have the same license;
4. **Unique Vehicle Driver**: Each Vehicle can have only one Driver;
5. **Registration Mandatory**: The registration is mandatory to use the service;
6. **Parallelism**: The service must manage multiple request from different Customer at the same time;
7. **Limit Queue of a Zone**: For each Zone, the Queue has an upper bound of 30 Vehicle;
8. **Different Address in Booked Request**: When a Customer make a Booked Request, Origin Address and Destination Address cannot be the same but must be different;
9. **Origin Address in the City**: The Origin Address of each Request must belong to the City in which the service is provided;
10. **Subset of addresses**: the set of addresses of each zone is a subset of the addresses of the city.
11. **Request only by Customer**: Only Customer can perform a Request.
12. **Unique queue for a Driver**: a Driver could appear only in one queue at the same time.

2.2 Preliminary Considerations

In the description of the system there are some unclear and missing aspects. During the design phase we have made these assumptions:

- **User that is not registered yet:** if a user is not registered, it can only see the login/registration page. To perform other tasks, it must be registered.
- **Unique username:** it is not possible that two users that are registered have the same username. During the registration phase, if the form seen that a username is already in the database, an error message is showed and the user must choose another username.

- **Notification to the Driver:** in the description is written that a Driver must answer to a Request by the mobile application, but it is not specified how it will be notified when a new Request has been sent. We assume that when a Customer generate a request, a Notification will be send to the assigned Driver, that must confirm or decline the Request.
- **External Developer must be registered:** it is not clear if an External Developer must be registered or not to access to the API of the service. We assume that each External Developer that want to use the API must be registered and a unique Digital Signature must be assigned.
- **GPS information:** It is not specified how GPS works. We assume that the GPS information is sent to the System automatically (once every 1 minute) in every case (Vehicle either available or busy).
- **Taxi fare payment:** It is assumed that a customer must pay only by cash. The system doesn't allow credit card payments.
- **Zone selected by GPS localization:** when a Driver gives its availability, the system automatically check the GPS position and add Driver to the nearest zone.

3. Requirement Analysis

3.1 Actors

There are four main actors that interact with the system:

- **Guest**: it is a not registered or a not logged User and can only see the login/register page. To use the system as Driver, Customer or External Developer, the Guest must complete the registration form or perform the login.
- **Driver**: it must be registered and logged in order to accept or decline new Request and give its availability to be inserted in a queue of a Zone.
- **Customer**: it must be registered and logged in order to make an Immediate Request or a Booked Request.
- **External Developer**: it must be registered and logged in order to access to the API of the system.

3.2 Functional Requirements

3.2.1 Guest

- Allow a Guest to register to the System.
  - Guest mustn't be registered yet to the System.
  - Guest can choose if the account has to be of kind Driver, Customer or External Developer.
  - Guest has to choose an username that is not used by another user (Customer, Driver or External Developer).
  - Guest can only see the Login/Registration page of the system.
- Allow a Guest to login to the System.
  - Guest must be already registered to the System.
  - Guest has to insert its username and password in order to perform the login phase.
  - If username and password aren't correct, Guest can't access to the system.

3.2.2 Driver

- Allow a Driver to accept or decline a Request.

        - User must be registered and logged and its account has to be of kind Driver.
        - When a Driver receives a Request, he can only accept or decline it.
        - A Driver cannot receive Request that are sent to other Drivers.
        - A Driver cannot answer to Request that are sent to other Drivers.
        - If a Driver has received a Request, it is not possible that it receives another Request, because if it decline the Request, it is moved at the end of the queue.
    - Allow a Driver to report its availability.
        - User must be registered and logged and its account has to be of kind Driver.
        - Driver can send its availability only when it is in a Zone of the city.
        - Driver cannot send its availability if it has just accepted a Request.
    - Allow a Driver to see his position in the queue.
        - User must be registered and logged and its account has to be of kind Driver.
        - Driver an see his position only if he has sent the availability.

3.2.3 Customer

    - Allow a Customer to perform an Immediate Request.
        - User must be registered and logged and its account has to be of kind Customer.
        - Customer has to be in the Zone where he wants that the taxi arrives.
        - The Zone in which the Customer is, must belong to the City.
        - Customer can't perform more than once the same Immediate Request.
    - Allow a Customer to perform a Booked Request.
        - User must be registered and logged and its account has to be of kind Customer.
        - Customer has to indicate origin address, destination address and the hour of the Booked Request.
        - The Booked Request must be performed at least two hours before the hour indicated for the meeting.

- Origin address has to be an address of the City.
- Customer can perform the Booked Request everywhere.

3.2.4 External Developer

- Allow an External Developer to request API.
  - User must be registered and logged and its account has to be of kind External Developer.

3.2.5 System

- System send a notification to a Driver when an Immediate Request or a Booked Request is performed.
  - Driver has to be sent its availability.
  - Driver has to be in the queue of the requested Zone.
  - Driver has to be either at the first place in the queue of the requested Zone or the Driver before the specific Driver has to be declined the Request.
- System send a notification to a Customer after that a Driver has accepted an Immediate Request.
- System send a notification to a Customer ten minutes before the meeting, after that a Driver has accepted a Booked Request.

3.3 Non Functional Requirements

3.3.1 Performance

The system must answer at each request in less than 1 second (a good response time is between 0.3-0.6 seconds). Loading time of the info on the mobile app and web app depends on the user internet connection (GPRS, 3G, LTE, Wi-Fi). In the worst case, response time + loading time mustn't exceed 40 seconds, instead in the better case users mustn't wait more than 5 seconds when they are using the system and a new page must be loaded and information must be retrieved.

3.3.2 Design Constraints

The system will be developed in different languages. For the web application will be used Java EE 7, Javascript, HTML5 and CSS3, in order to be compatible with all modern browser that are up-to-date (if a browser is not supported, a list of compatible browser will be shown). For the iOS application, Swift language will be used, instead for the Android application Java language will be used. In this way will be possible use all the functionalities of the mobile devices.

3.3.3 Software Availability

A Customer can Request a taxi at each hour of the day. For this reason the system must works 24 hours a day. To guarantee this availability, will be used dedicated server, that will be maintained and controlled continuously.

3.3.4 Error Handling

External catastrophic events or lost connection may cause the lost of data. For this reason must be ensured consistency and persistency of the data.

3.3.5 Security

The system will manage personal information of the users. For this reason will be used secure connection (https) during all the sessions. Moreover, also the passwords must be preserved. When an user compile the registration form, password send to the server will be inserted in the database after the use of an hashing mechanism. To improve the robustness of the password, it must be at least of 6 characters, with at least one capital letter and one number. Besides the password field, an indicator will show if the password is secure or not.

4. Scenarios

4.1 Bob need a taxi immediately

Scenario 1	Bob need a taxi immediately
Description	During a shopping day in Milan, Bob need a taxi in the centre of the city. Unfortunately no one is available on the street. He decide to search on the web a different way to get it. He find the TaxiService website and, after a quickly registration, he require an immediate taxi. In a few minutes a vehicle arrives and Bob can go to the next shopping centre.
Actors who participate	Guest (Bob when find TaxiService website) Customer (Bob after the registration)
Entry conditions	Bob is a guest user of the service Bob need taxi immediately
Flow of events	1) Bob searching TaxiService on the web 2) Bob fill the registration form and submit it 3) Bob make a request of immediate taxi (communicating his position) 4) Bob receive a confirmation from the system 5) Bob join the taxi and start for the final destination
Exit conditions	Bob arrives to the final destination and pay for the trip
Exception	No taxies are available in the queue's zone of Bob Session is expired

4.2 Steve develop an external trips application

Scenario 2	Steve develop an external trips application
Description	Steve is a developer and he need to get real time data from the TaxiService system to implement additional services in his own application. After a registration (as External Developer) the system provide it a digital signature. Steve use his signature to get real time data using the system API.
Actors who participate	External Developer

Entry conditions	Steve require an account as External Developer
Flow of events	1) Steve create an account 2) Steve require API using his digital signature
Exit conditions	Steve obtain all the needed data
Exception	Digital signature is invalid Session is expired

4.3 John is driving around San Babila square

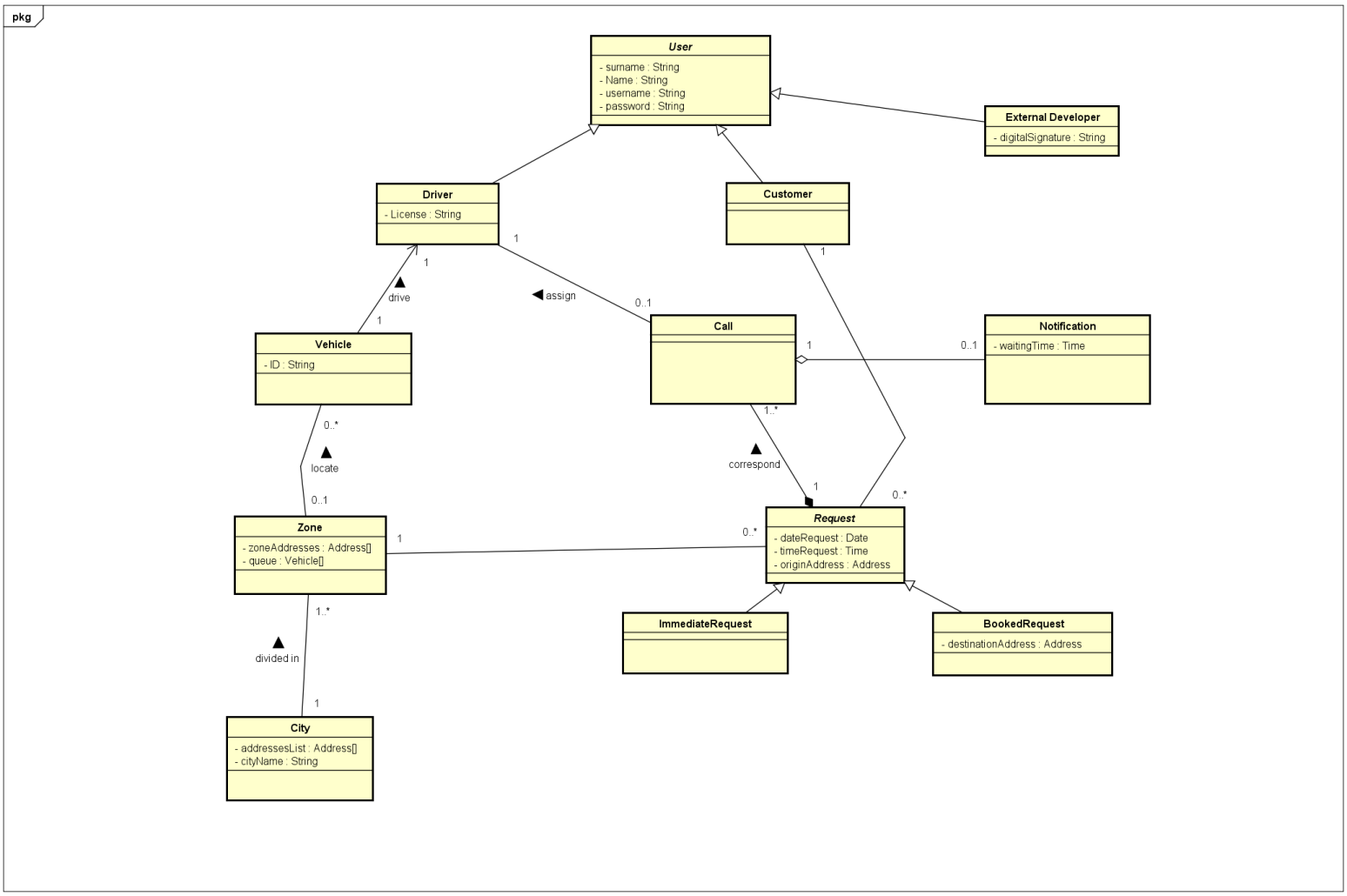
Scenario 3	John (a Driver) is driving around San Babila square
Description	John is a Driver registered on TaxiService. He terminate the last trip on San Babila square, so he have communicate, through the mobile application, his availability. After few minutes he receives from the system a request for a possible customer (Marie) in Duomo square. He decide to accept and send a positive answer to the system. After that, he moving to Duomo square to meet the customer. When they arrive to the final destination (Expo), John communicate to the system his availability.
Actors who participate	Driver (John) Customer (Marie)
Entry conditions	John communicate his availability There's at least one customer requiring taxi in the same zone of John
Flow of events	1) John communicate his availability 2) Marie requests a taxi in Duomo square 3) John accepts the call 4) Marie and John meets in Duomo square 5) Marie and John arrives at Expo 6) John communicate to the system his availability
Exit conditions	John and Marie reach the final destination
Exception	John cannot find Marie, so he mark his taxi as available

4.4 Alice need to book a taxi

Scenario 4	Alice need to book a taxi
Description	Alice is a frequent user of the TaxiService app. On Monday evening she have to go to the airport. She open the application and she select from the menu the “Book a trip” option. After have fill the form with date, time, original and final destination, she confirm the request. On Monday, ten minutes before the appointment, she receive a reminder notification and so she go out of her home for waiting the taxi. The Driver (Simon) arrives in 5 minutes and they can go to the Linate airport.
Actors who participate	Alice (Customer) Driver (Simon)
Entry conditions	Alice is logged into the application Alice wants to create a Booked Request
Flow of events	1) Alice do the login 2) Alice make a booking 3) Ten minutes before the trip, Alice receives the notification 4) Alice meets Simon and they start the trip 5) Arrived at the airport, Alice pay the taxi fare
Exit conditions	Alice pay the taxi fare
Exception	Alice change her planning and delete the Booked Request

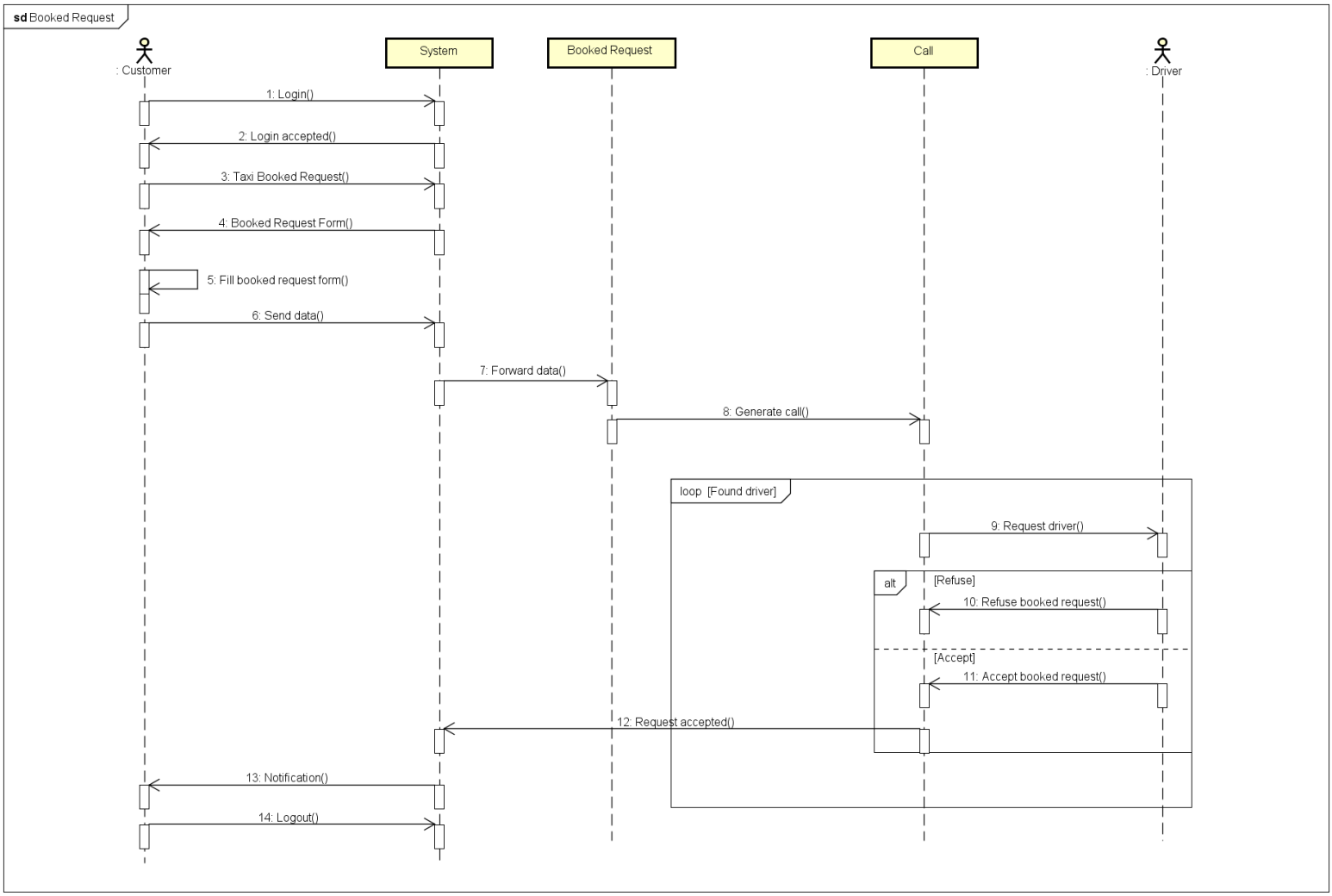
5. Diagrams and Alloy

5.1 UML

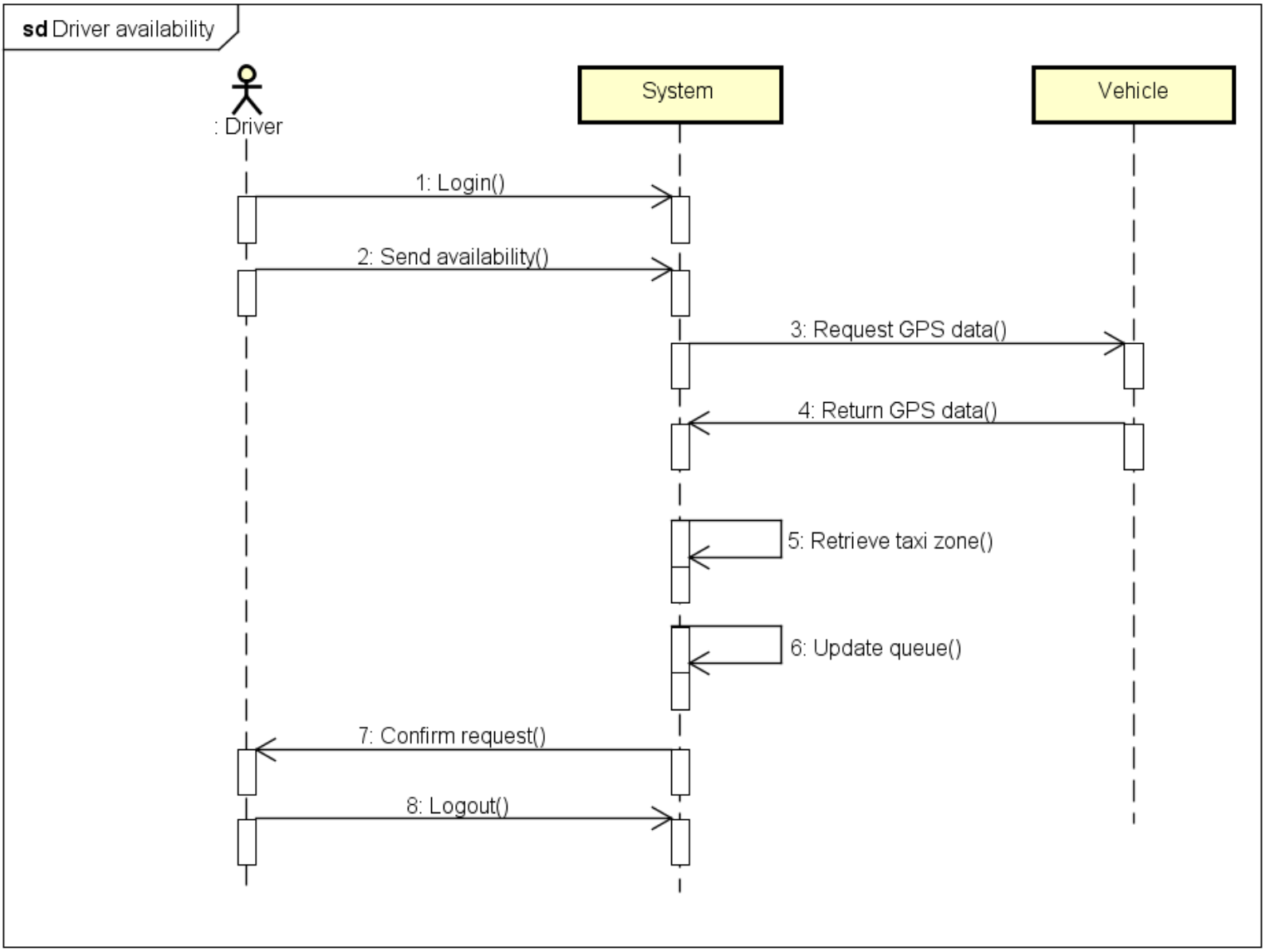


5.2 Sequence Diagrams

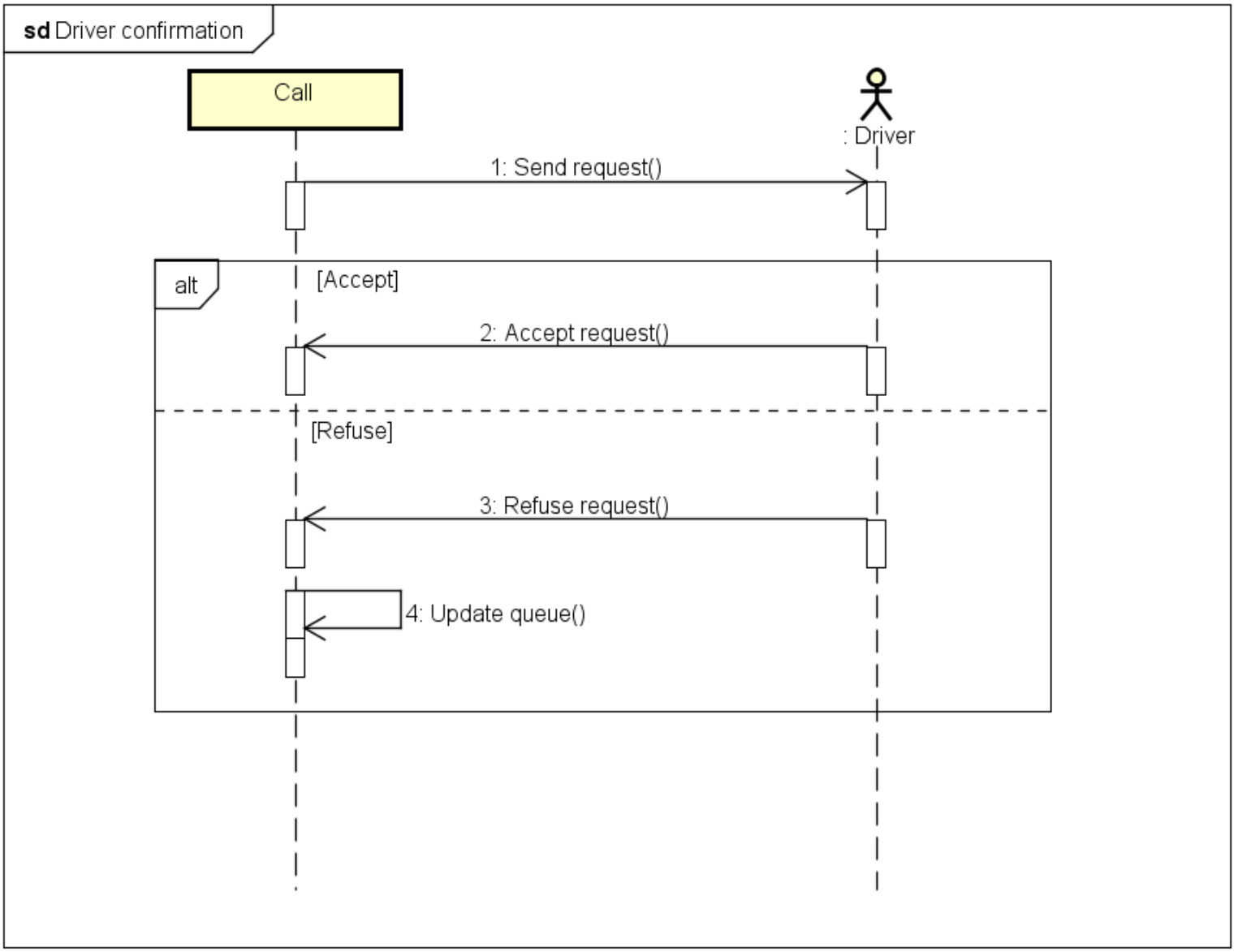
5.2.1 Booked Request



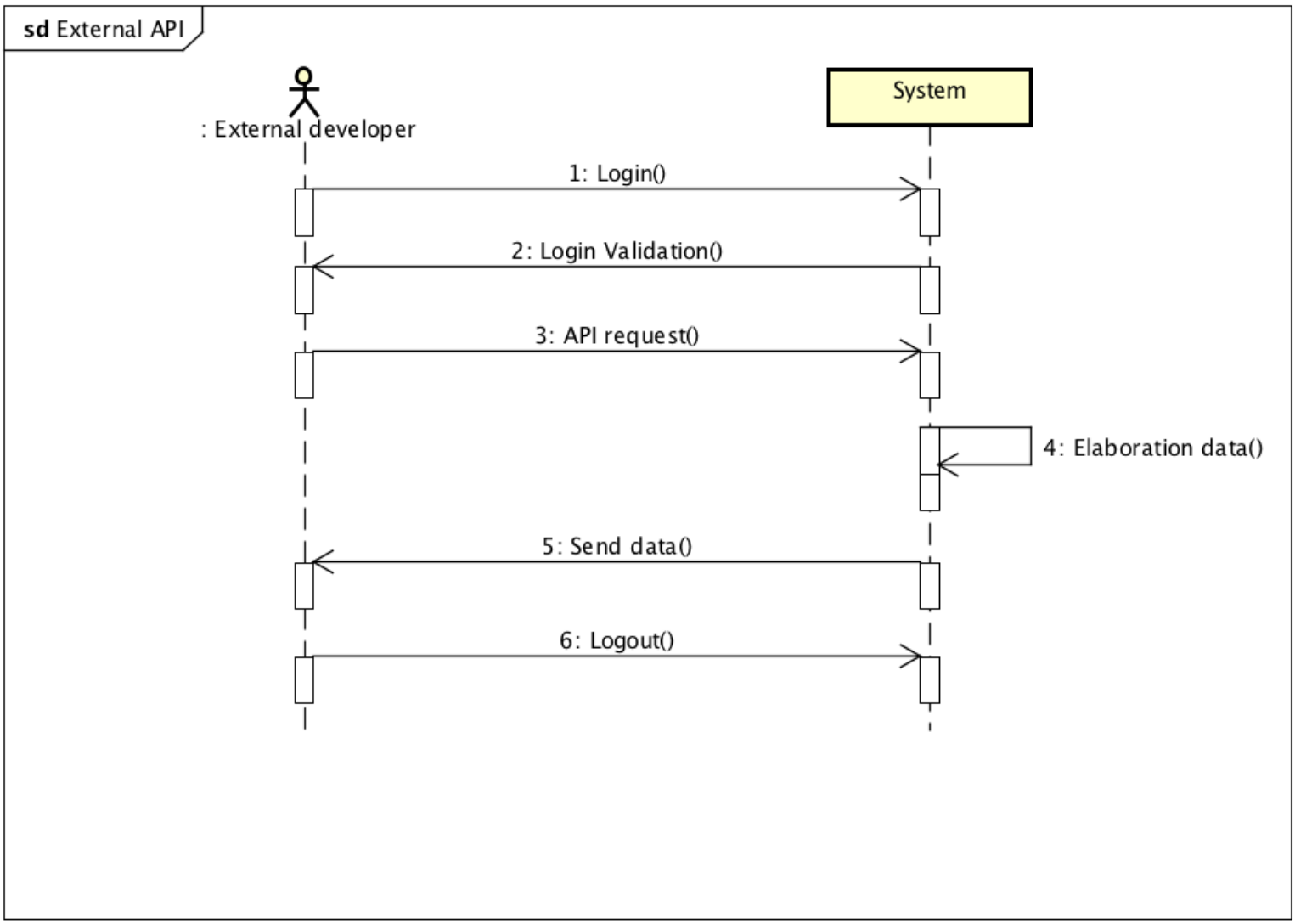
5.2.2 Driver Availability



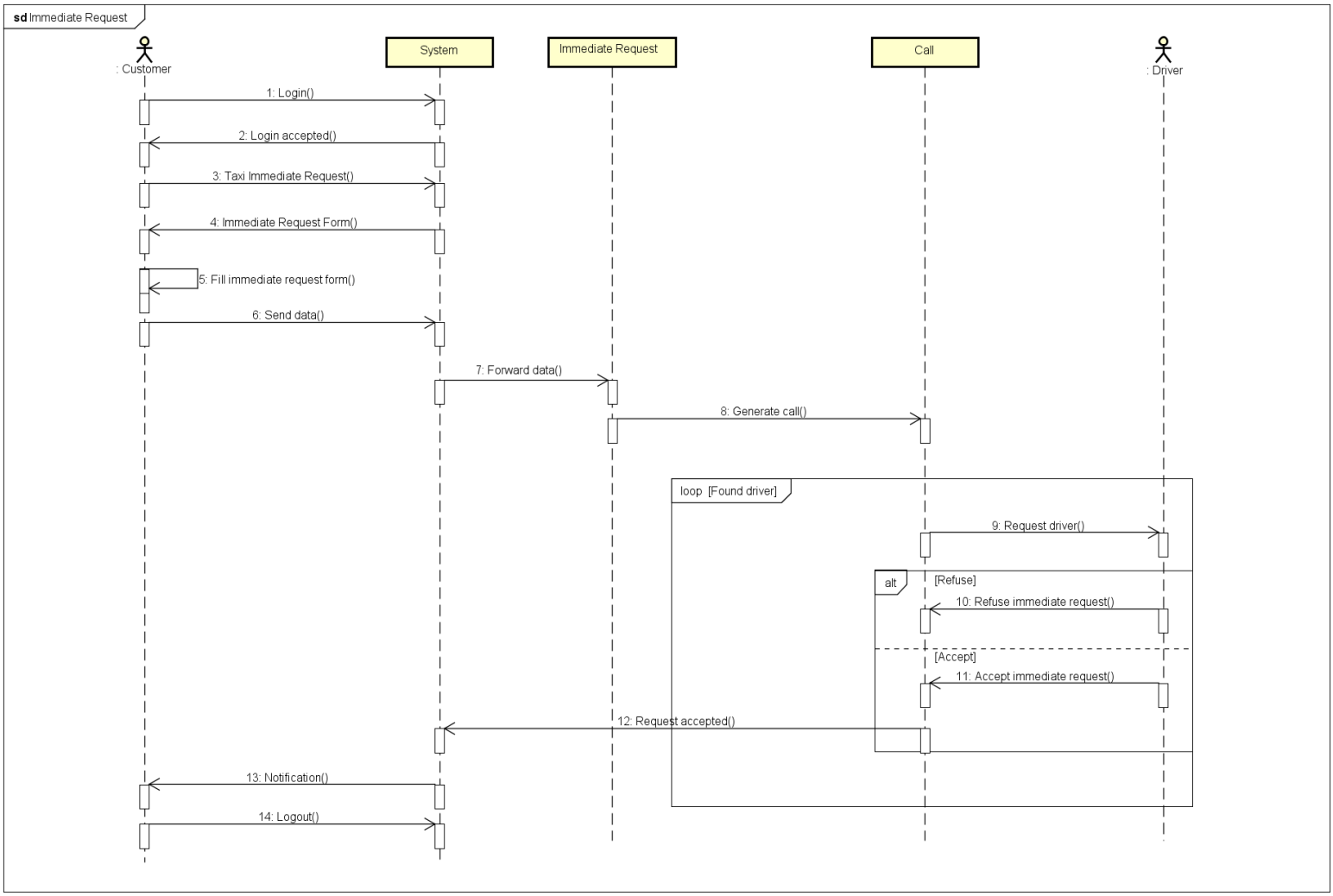
5.2.3 Driver Confirmation



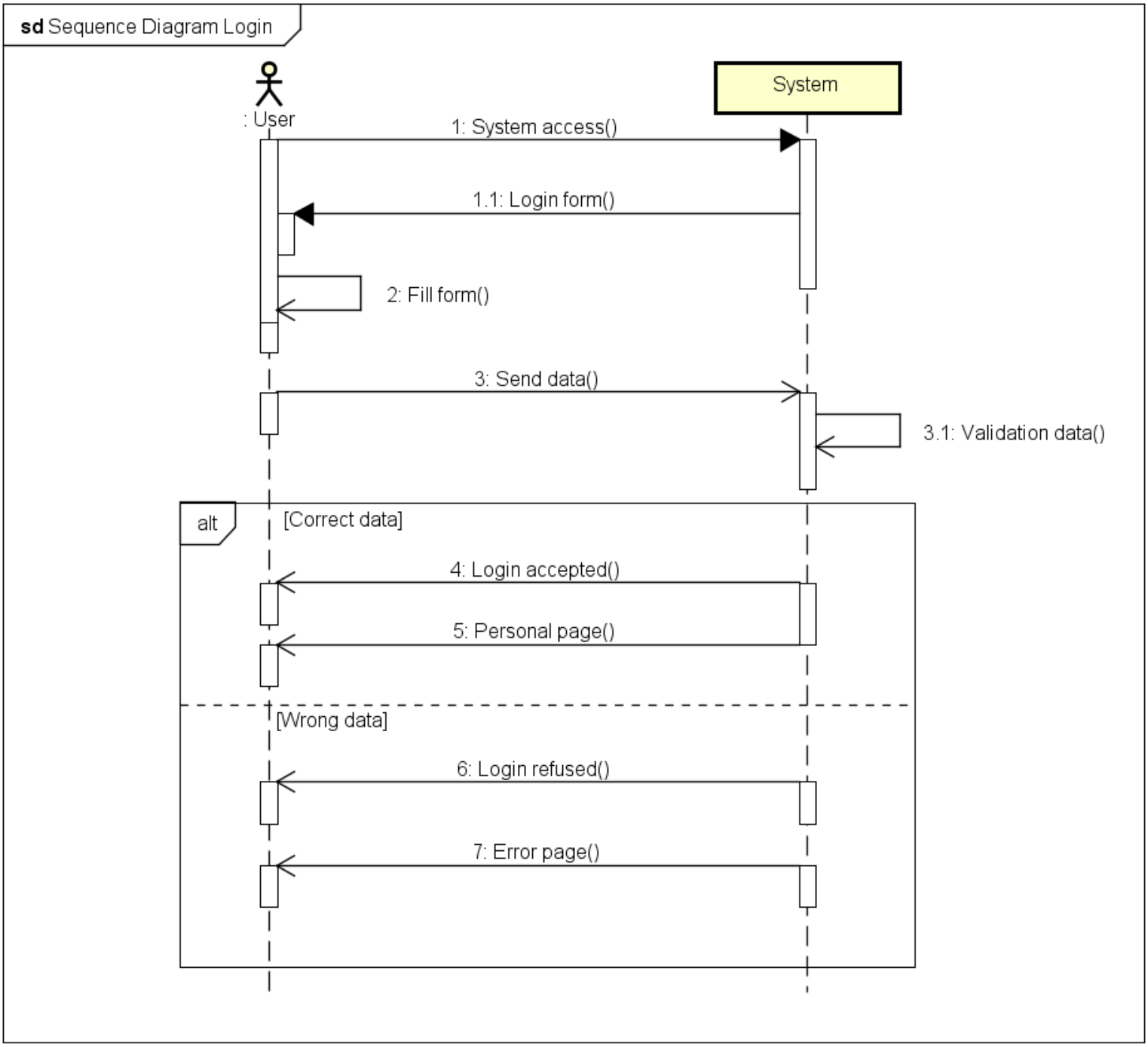
5.2.4 External API



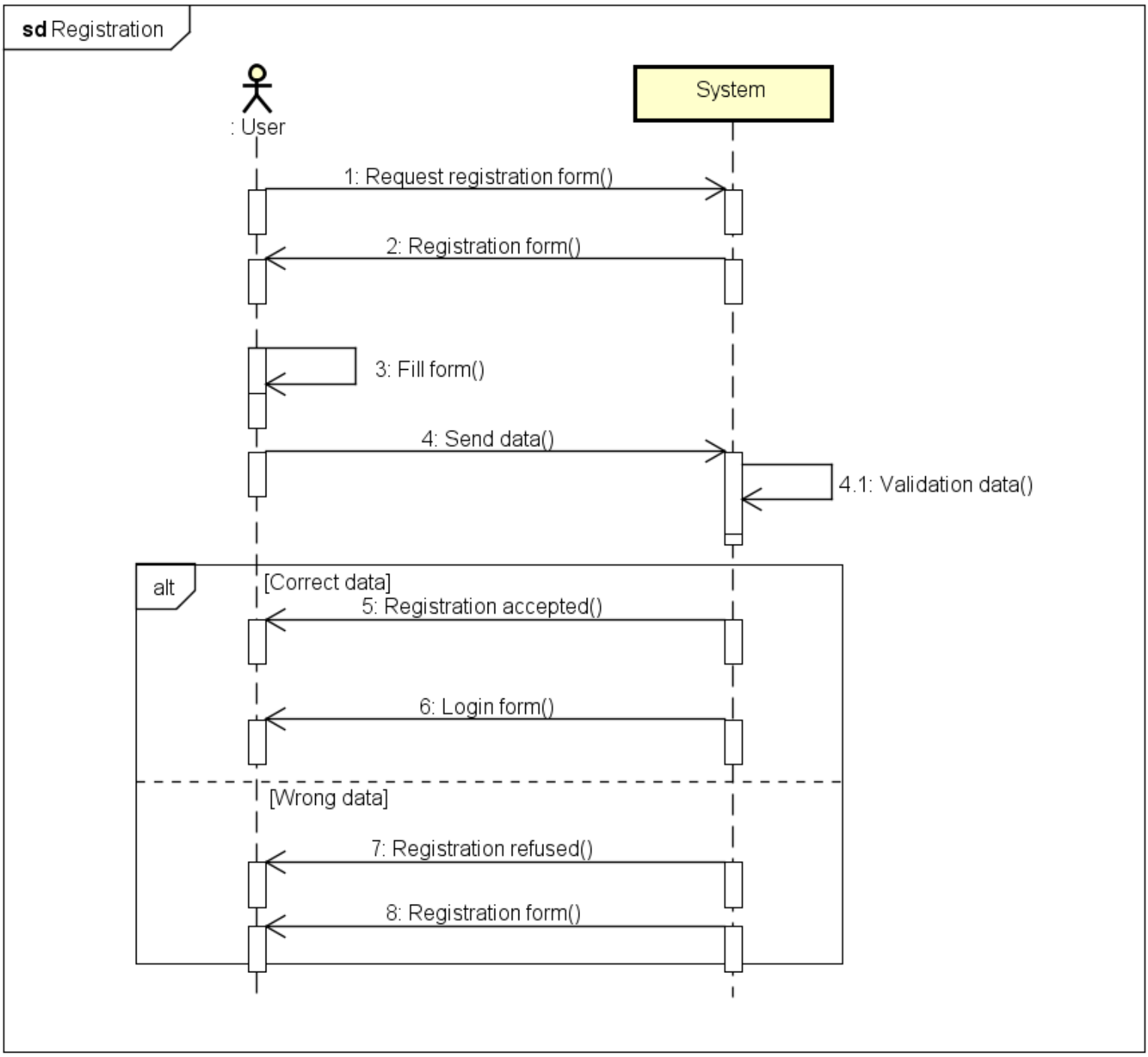
5.2.5 Immediate Request



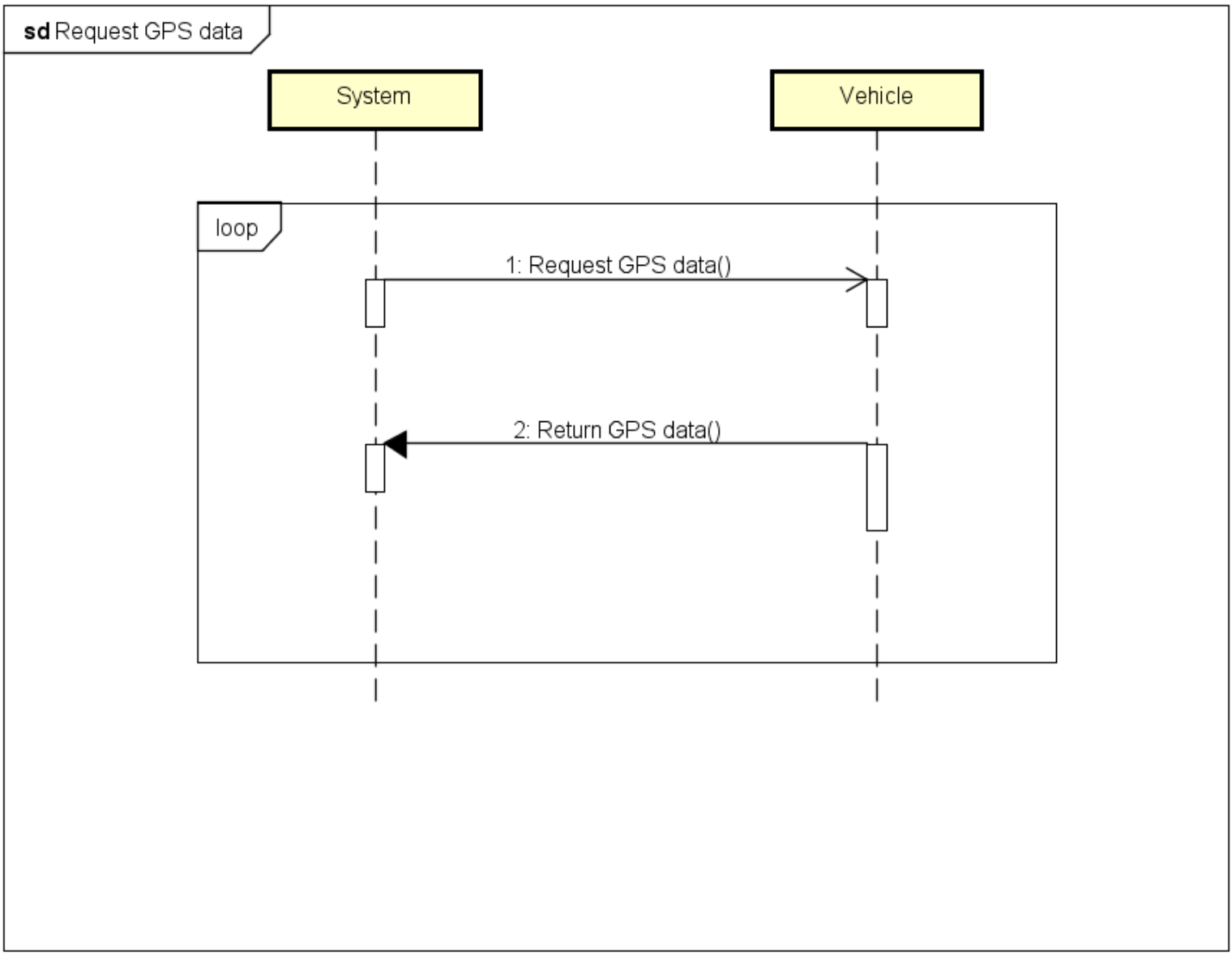
5.2.6 Login


sd Sequence Diagram Login
: User
System
1: System access()
1.1: Login form()
2: Fill form()
3: Send data()
3.1: Validation data()
alt
[Correct data]
4: Login accepted()
5: Personal page()
[Wrong data]
6: Login refused()
7: Error page()

5.2.7 Registration

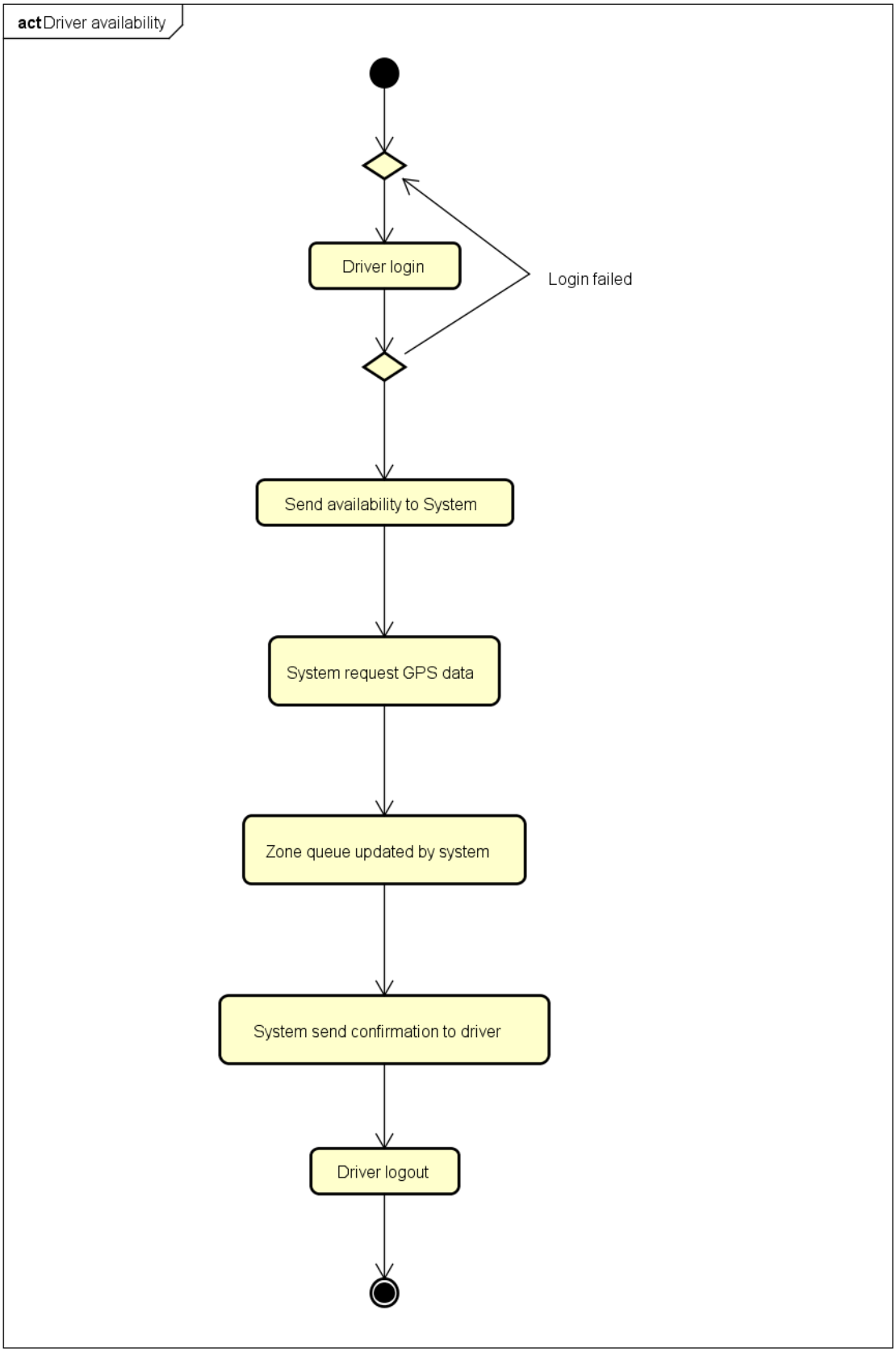


5.2.8 Request GPS Data

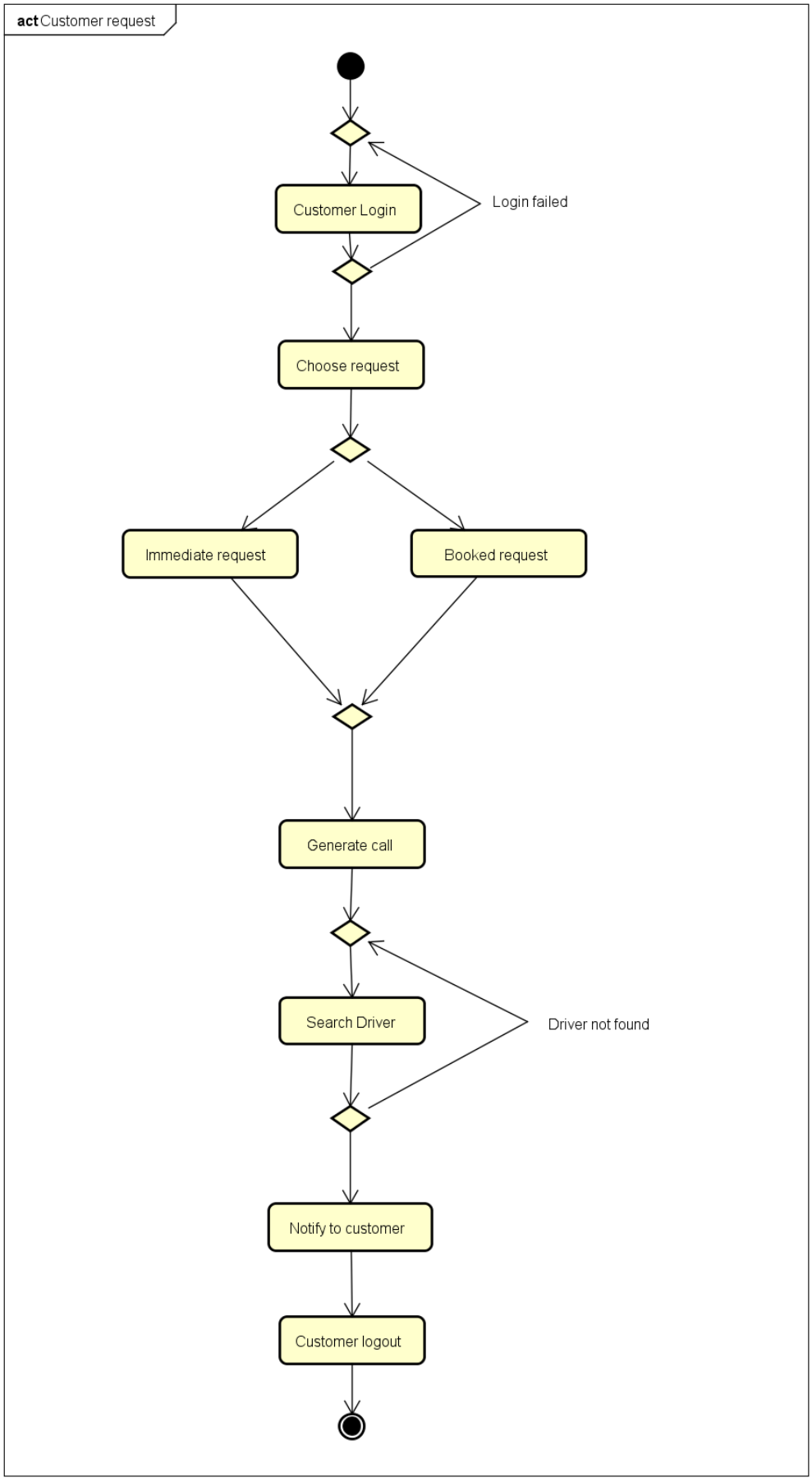


5.3 State Chart

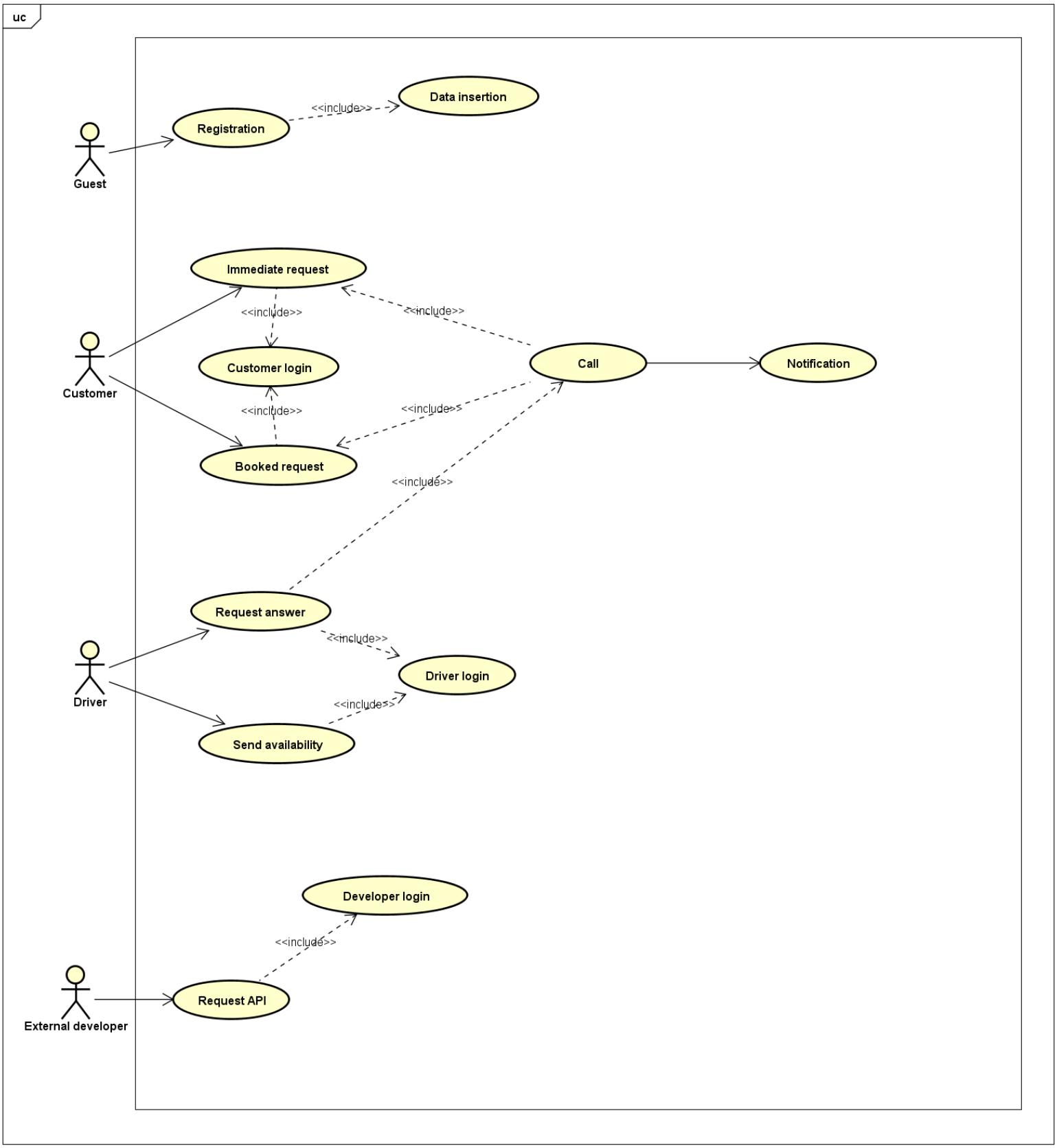
5.3.1 Driver Availability



5.3.2 Customer Request



5.4 Use Case


uc
Guest
Registration
<<include>>
Data insertion
Immediate request
<<include>>
<<include>>
Customer
Customer login
Call
Notification
<<include>>
<<include>>
Booked request
<<include>>
Request answer
<<include>>
Driver
Driver login
<<include>>
Send availability
Developer login
<<include>>
External developer
Request API

5.5 Alloy

5.5.1 Signature

In this section are represented the signature utilized in Alloy to construct our model, identified with the relatives attributes.

```
/*****************************************************/
/*********************sign****************************/
/*****************************************************/

sig Str{}

sig Date{}

sig Time {}

sig Address {}

one sig City {
    cityZones: some Zone,
    cityAddresses: some Address
}

abstract sig User {
    name: one Str,
    surname: one Str,
    username: one Str,
    password: one Str
}

sig Driver extends User {
    license: one Str,
    vehicleDriver: one Vehicle,
    requestAccepted: lone Request,
    requestCall: lone Call
}

sig Customer extends User {
    openRequest: lone Request
}

sig ExternalDeveloper extends User {
    digitalSignature: one Str
}
```

```
sig Vehicle {
    vehicleID: one Str,
    driverVehicle: one Driver
}

sig Zone {
    zoneAddresses: some Address,
    queue: set Vehicle,
    belongCity: one City,
    activeRequest: set Request
}

abstract sig Request {
    dateRequest: one Date,
    timeRequest: one Time,
    customerRequest: one Customer,
    driverRequest: one Driver,
    originAddress: one Address,
    requestZone: one Zone,
    requestCall: some Call
}

sig ImmediateRequest extends Request {}

sig BookedRequest extends Request {
    destinationAddress: one Address
}

sig Call {
    callDriver: one Driver,
    callRequest: one Request,
    callNotification: lone Notification
}

sig Notification {
    waitingTime: one Time,
    notificationCall: one Call,
    notificationCustomer: one Customer

}
```

5.5.2 Fact

The Fact represents all the limitations of our System.

```
/*************************************************/
/*******************fact*************************/
/*************************************************/

// There is the same number of Vehicle and Driver
fact sameDriverVehicleNumber { #Vehicle = #Driver }

// The queue of each zone can contain at most 30 Vehicle
fact queueBound {all z: Zone | #z.queue <= 30 }

//The same Vehicle cannot be in two different queue
fact vehicleUniqueQueue {no z,z': Zone, v: Vehicle | z != z' && v in z.queue && v in z'.queue }

// A Driver User is the Driver of its Vehicle
fact uniqueDriverVehicle { all v: Vehicle, d: v.driverVehicle | d.vehicleDriver = v }

// Each Zone must belong to one City
fact uniqueZoneCity { all z: Zone | one c: City | z.belongCity = c }

// Two Users cannot have the same username
fact uniqueUsername { no u,u': User | u.username=u'.username && u!=u' }

// Two Drivers cannot have the same license
fact uniqueLicense { no d,d': Driver | d.license=d'.license && d!=d' }

// Two Vehicle cannot have the same vehicleID
fact uniqueVehicleID { no v,v': Vehicle |v.vehicleID=v'.vehicleID && v!=v' }

// Driver license and ExternalDeveloper digitalSignature cannot be the same
fact uniqueLicenseDigitalSignature { no d: Driver, e: ExternalDeveloper |d.license=e.digitalSignature }

// The set of addresses of all zones is a subset of the set of addresses of the city
fact addressesZoneCity { all z: Zone | one c: City | z.zoneAddresses in c.cityAddresses }

// If an address is in a zone, it could not be in another zone
fact uniqueAddressZone { no z,z': Zone | (z.zoneAddresses & z'.zoneAddresses) = none && z!= z' }
```

```
// Origin Address of a Request is in one Zone
fact origAddressInZone { all r: Request | one c: City | r.originAddress in c.cityAddresses }

// Destination Address of a Request is in one Zone
fact destAddressInZone { all r: BookedRequest | one c: City | r.destinationAddress in c.cityAddresses }

// Each Customer can have onlyone Request
fact oneCustomerRequest { no r,r': Request | r.customerRequest = r'.customerRequest && r != r' }

// Each Driver can have onlyone Request accepted
fact oneDriverRequest { no r,r': Request | r.driverRequest = r'.driverRequest && r!=r' }

// A Request has a Driver and that Driver has that Request
fact uniqueDriverRequest { all r: Request, d: r.driverRequest | d.requestAccepted = r }

// A Request has a Customer and that Customer has that Request
fact uniqueCustomerRequest { all r: Request, c: r.customerRequest | c.openRequest = r }

// A Request must be in the set of requestZone of the Zone
fact requestInSetZone { all r: Request, z: r.requestZone | r in z.activeRequest }

// A Request has only one Zone
fact requestOneZone { no z,z': Zone, r: Request | r in z.activeRequest && r in z'.activeRequest && z!= z'  }

// A Call has a Driver and the Driver has that Call
fact uniqueDriverCall { all c: Call, d: c.callDriver | d.requestCall = c }

// A Call has a Request and the Request has that Call
fact uniqueRequestCall { all c: Call, r: c.callRequest | r.requestCall = c }

// A driver that receives a call must be in the queue of the requested zone
fact callDriverInTheZone { all c: Call, z: c.callRequest.requestZone | c.callDriver.vehicleDriver in z.queue  }

// A Notification has a Call and the Call has that Notification
fact uniqueNotificationCall {  all n: Notification, c: n.notificationCall | c.callNotification = n }

// Notification is sent to the Customer that has made the Request
fact notificationToRightCustomer { all n: Notification, c: n.notificationCall | n.notificationCustomer = c.callRequest.customerRequest }
```

5.5.3 Assertions and Predicate

The assertions evaluate the consistency of our model. The predicate instead show a simulation of the entire world.

```
/*****************************************************/
/*********************assert**************************/
/*****************************************************/

// It is impossible that two different Vehicle are driven by the same Driver
assert uniqueDriverVehicle { no v,v': Vehicle | v.driverVehicle=v'.driverVehicle && v!=v' }
check uniqueDriverVehicle for 5

// It is impossible that the same Driver has to Call at the same time
assert uniqueDriverCall { no c,c': Call | one d: Driver | c.callDriver = d && c'.callDriver = d && c != c' }
check uniqueDriverCall for 5

//Origin and Destination Address are two address that belong to the City
assert origAndDestBelongCity { all r: BookedRequest | one c: City | r.destinationAddress in c.cityAddresses && r.originAddress in c.cityAddresses }
check origAndDestBelongCity for 5


/*****************************************************/
/*********************pred**************************/
/*****************************************************/

pred show {
   #City = 1
   #Driver = 3
   #Vehicle = 3
   #Customer = 3
   #ExternalDeveloper = 1
   #Zone = 1
   #ImmediateRequest > 0
   #BookedRequest > 0
   #Call > 0
   #Notification > 0
}

run show for 10
```

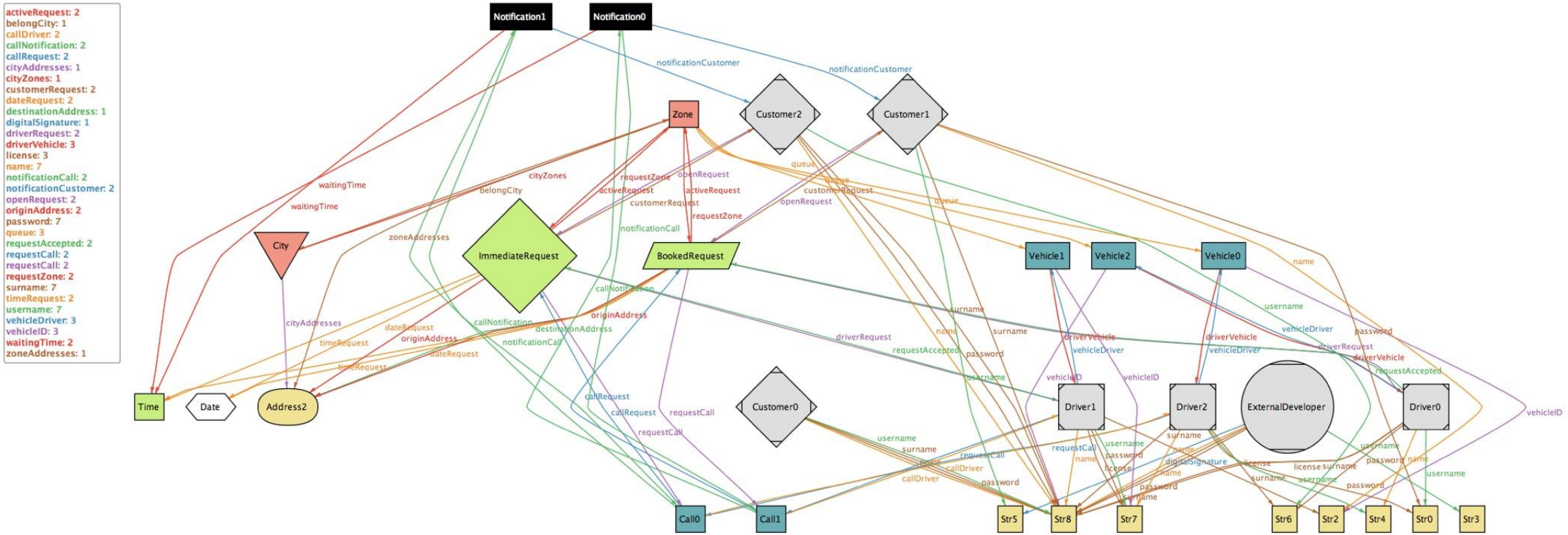
5.5.4 Execution

```
Executing "Run show for 10"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  42219 vars. 3060 primary vars. 81607 clauses. 361ms.
  Instance found. Predicate is consistent. 196ms.

Executing "Check uniqueDriverVehicle for 5"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  9875 vars. 815 primary vars. 16738 clauses. 46ms.
  No counterexample found. Assertion may be valid. 2ms.

Executing "Check uniqueDriverCall for 5"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  9905 vars. 815 primary vars. 16880 clauses. 45ms.
  No counterexample found. Assertion may be valid. 1ms.

Executing "Check origAndDestBelongCity for 5"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  9825 vars. 810 primary vars. 16555 clauses. 40ms.
  No counterexample found. Assertion may be valid. 8ms.
```


activeRequest: 2
belongCity: 1
callDriver: 2
callNotification: 2
callRequest: 2
cityAddresses: 1
cityZones: 1
customerRequest: 2
dateRequest: 2
destinationAddress: 1
digitalSignature: 1
driverRequest: 2
driverVehicle: 3
license: 3
name: 7
notificationCall: 2
notificationCustomer: 2
openRequest: 2
originAddress: 2
password: 7
queue: 3
requestAccepted: 2
requestCall: 2
requestCall: 2
requestZone: 2
surname: 7
timeRequest: 2
username: 7
vehicleDriver: 3
vehicleID: 3
waitingTime: 2
zoneAddresses: 1
Notification1
Notification0
Zone
Customer2
Customer1
City
ImmediateRequest
BookedRequest
Vehicle1
Vehicle2
Vehicle0
Time
Date
Address2
Customer0
Driver1
Driver2
ExternalDeveloper
Driver0
Call0
Call1
Str5
Str8
Str7
Str6
Str2
Str4
Str0
Str3