Politecnico di Milano
A.A. 2015-2016
Software Engineering 2

Requirements Analysis and Specifications Document

Alessandro Macchi
Caterina Finetti
Simone Manzoli

Index

Purpose

This document represent the Requirement Analysis and Specification Document (RASD). The main goal of this document is to completely describe the system in terms of functional and non-functional requirements, analyse the real need of the customer to modelling the system, show the constraints and the limit of the software and simulate the typical use cases that will occur after the development. This document is intended to all developer and programmer who have to implement the requirements, to system analyst who want to integrate other sy[1]stem with this one, and could be used as a contractual basis between the customer and the developer.

Glossary

User: is a person who is registered in the database of the application. He has acces to all the functions of the program that involves the requiring of a taxi, shared or not. He also has the possibility to save a list of preferred locations, that he can automatically choose when the System require from him an adress as starting position or destination.

Guest: is a person who is using the application but is not registered in the database. He has acces only to the registration functions.

User Informations: all the information that concern a user; most of them have to be inserted during the registration (Name, Surname, tel. Number, e-mail, password), some of them can be inserted at any time after the registration (such as the personals locations) and others are assigned by the system (for example the number of blank-calls).

Basic User Informations:

Blank Call: we define Blank-call a call for a taxi where the client is not at the starting location when the taxi arrives with a maximum late of X minutes, or a call that the user cancel befor X minutes.

Missed Call: we define Missed-call a call for a taxi where the client is not at the starting location when the taxi arrives (X+1) or more minutes late.

Partner: someone who share a run with a user

Pick-up place: the Adress where a user asks a taxi to come

Taxi Driver: a registered one

Goals

A user should be able to:

- Sign up to the system (starting as a guest)
- Log in to the system
- Call for a taxy that should pick him up asap
- Be informed by the system when a taxy driver accepts his call and of the estimated waiting time
- Make a reservation for a taxy on specified date and time
- Give his avilability for sharing a run
- Add, modify or delete a personal location

A taxi driver should be able to:

- accept or decline a call
- visualize the basic informations about the user who is making a call
- visualize the price for a run

The system should provide a fair management of the calls and efficient and essential communication between drivers and users.

Domain properties

- A taxi can always reach a pick-up place and the users can only choose destinations reachable (and in the controlled zones?)
- A zone can't remain without at least one available taxi
- A taxi driver that accepts a call will take it.
- Gps informations and maps are always updated.

Assumptions

- Two or more users can share a taxi only if their pick up place are in the same zone.
- The taxi drivers have a device on-board to communicate with the system, that shows them the incoming calls and permits them to accept or refuse.
- Every taxy has a gps-signal that can always give their position to the system
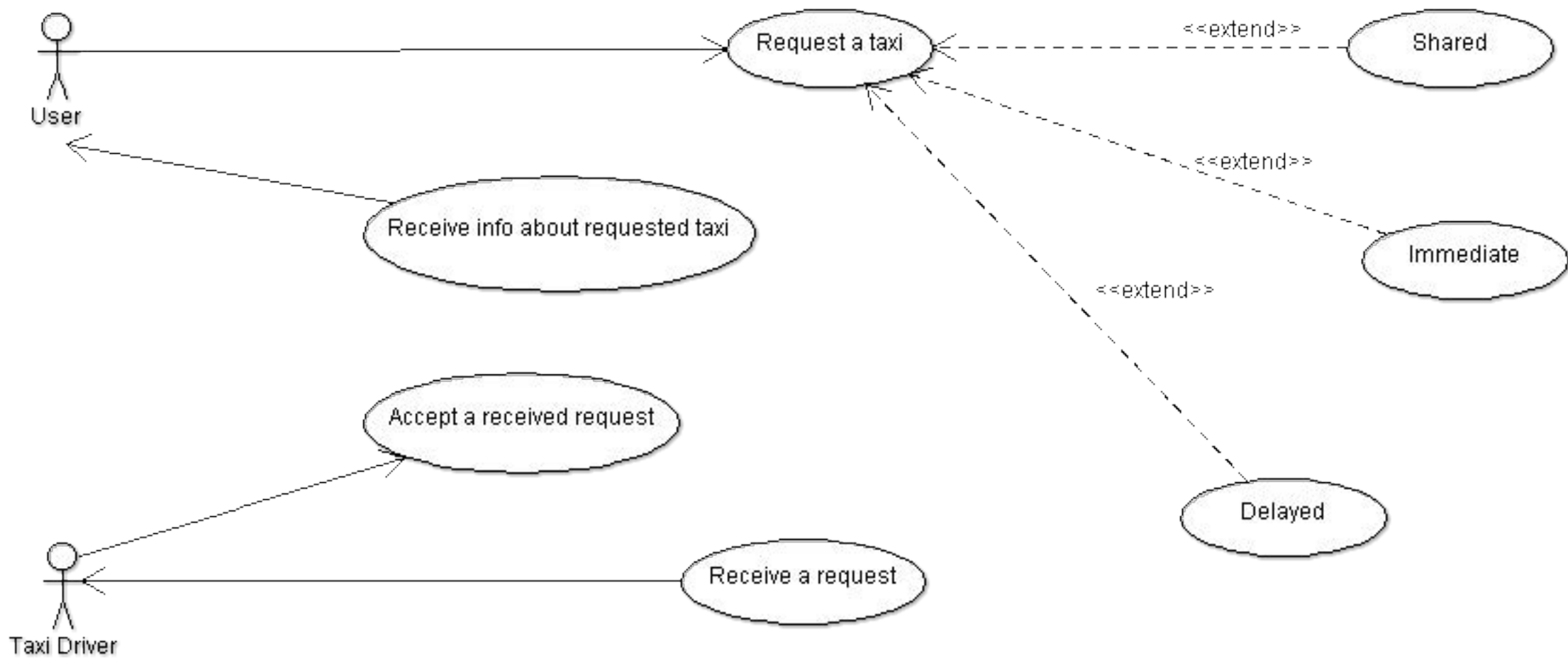
Requirements

- Registration of a person to the system
  - the system has to provide a sign up functionality

- Modification of personal informations
  - The system has to provide a function that allows a user to modify his personal information, except for the number of calls and blank calls

- Call for a taxi
  - the system has to provide a function to call for a taxi
  - the user should be able to visualize the estimated arrival time and confirm or not his call
  - the system has to manage the number of calls and blank calls of every user

- Make a reservation for a taxi to a specified date and time
  - the system has to provide a function to allow a user to call for a taxi in a specified date and time
  - the system has to call for the taxi 10 minutes before the specified time

- Give is avilability for sharing a run
  - the system has to provide a function to allow a user to declare his avilability for a taxi sharing
  - the system has to search for other users in the same zone that wanna share a ride, and automatically inform the interested users and the taxi driver; then, if they accepts, has to calculate the cost of the ride.

- Create, modify delete personal location
  - The system must provide a function to allow a user to create a new personal location
  - The system has to provide a function to allow a user to modify a personal location
  - The system has to provide a function to allow a user to delete a personal location.

- Visualization of ride information
  - the system has to provide a function that allows a user to visualize the informations about his actual ride (basicaly the cost calculated by the taximeter)

- Accept or decline a call
  - The system has to forward the calls to the first taxi of the queue
  - The system has to provide a function to allow a taxi driver to accept or decline a call when he receives it
  - If a call is declined, the system has to move the taxi driver that

declined it at the end of the queue and send the call to the new firts in the queue
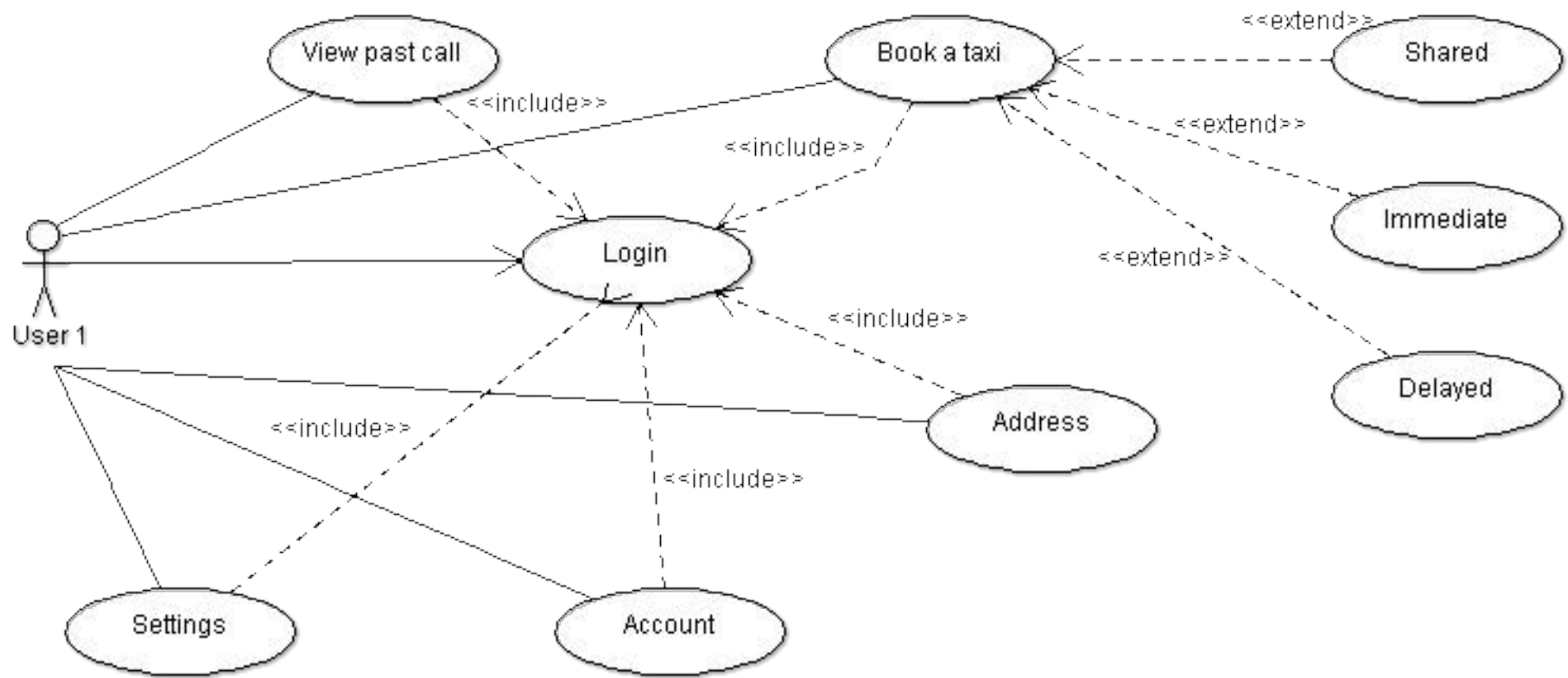
- Visualize the basic informations about a user who is making a call
  ◦ the system has to provide a function to allow a taxi driver to visualize the basic informations of a user that is making a call

- Signal a blank call
  ◦ the system has to provide a function to allow a taxi driver to signal a blank call
  ◦ the system has to put the taxi back in te queue and update the blank_calls attribute of the user

Functional requirements

- Guest can
  ◦ sign up

- User can
  ◦ log in
  ◦ log out
  ◦ modify his personal information except for the number of calls and blank calls
  ◦ create a new personal location
  ◦ modify an existing personal location
  ◦ delete an existing personal location
  ◦ do an immediate call for a taxi (shared or not)
  ◦ do a delayed call for a taxy
  ◦ visualize the ride informations

- Taxi driver can
  ◦ receive notifications for the calls
  ◦ visualize the basic information of a user that is making a call
  ◦ accept a call
  ◦ decline a call
  ◦ signal a blank call

First Use Case

Name	Taxi request
Actors	User, Taxi Driver
Entry conditions	User have to be logged in and must be in the reservation page
Flow of events	•The user request a taxi in his actual position •The system send the request and the user's basic information to the first available taxi driver in the zone •The taxi driver accepts the request. •The system send to the user the expected waiting time and the code of the incoming taxi, and place the taxi driver in "waiting" state •the user confirms the request within 1 minute • the system removes the taxi driver form the queue and give to him/her confirmation •The taxi driver goes to pick up the user and brings him/her to the destination. •The taxi driver informs the system about his/her availability.
Exit conditions	•The System put the taxi driver in the last position of his/her actual zone's queue
Exceptions	•The taxi driver doesn't accept the call. In this case the system forwards the request to the second in the queue and move the first taxi in the last position in the queue. •The user does not confirm his request after receiving notification of the expected arrival time. In this case the system put the taxi driver in "available" state. •The user is not in the pick up place when the taxi arrives. See the dedicated use case.

Second Use Case

Name	Delayed Taxi request
Actors	User, Taxi Driver
Entry conditions	User have to be logged in and must be in the reservation page
Flow of events	The user requests the taxi specifying the time and the position
	The system register the request and 10 minutes before the scheduled deparure send the request's and the user's basic information to the first taxi driver in the zone
	The taxi driver accepts the request
	The system removes the taxi driver from the queue and send to the user the reminder of the incoming taxi and the expected waiting time
	The taxi driver goes to pic up the user and brings him/her to the destination
Exit conditions	The taxi driver informs the system about his/her availability
	The System put the taxi driver in the last position of his/her actual zone's queue
Exceptions	The taxi driver doesn't accept the call. In this case the system forwards the request to the second in the queue and move the first taxi in the last position in the queue
	The user is not in the pick up place when the taxi arrives. See the dedicated use case

Third Use Case

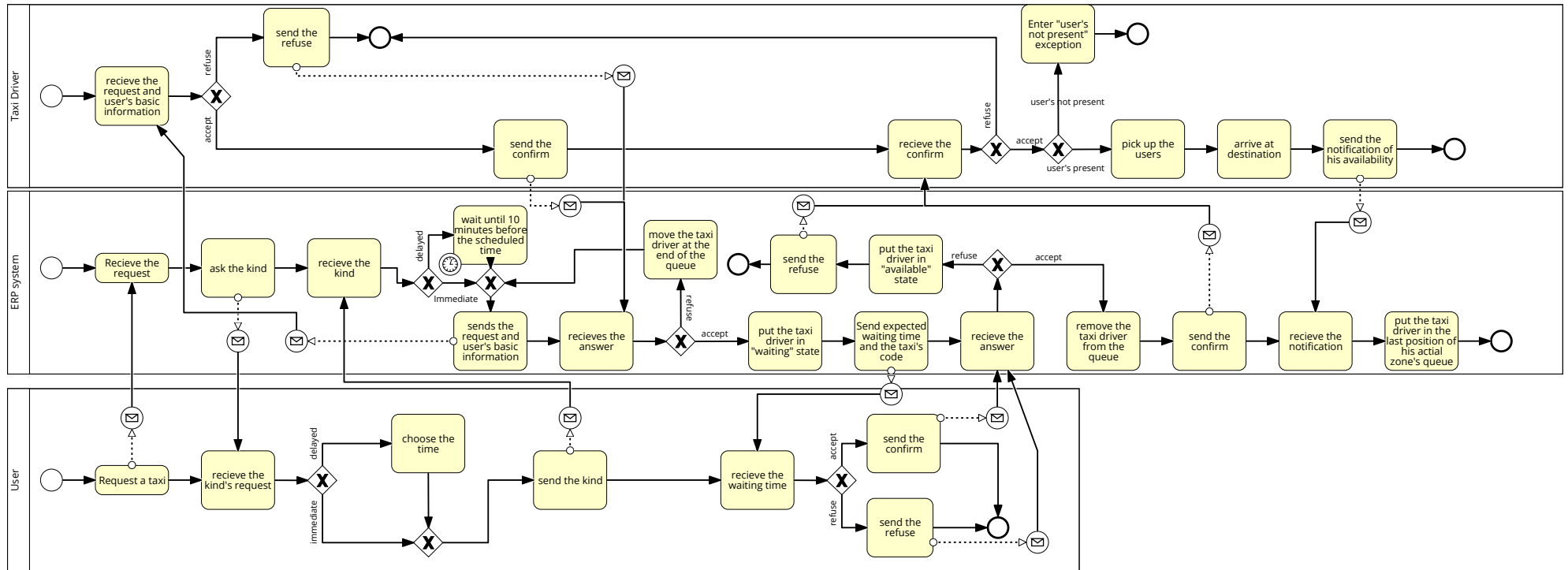|	
Name	Failed pick up
Actors	Taxi Driver, user
Entry conditions	User have requests a taxi but he/she isn't in the pick up place when the taxi arrives
Flow of events	The taxi driver sends the "failed pick up" signal The system sends a notification to the user that informs him/her of his reputation will be decreased The system decreases the user's reputation
Exit conditions	The system places the taxi driver at the top of the queue andhis/her state to "available" and sends him/her a notification

Fourth Use Case

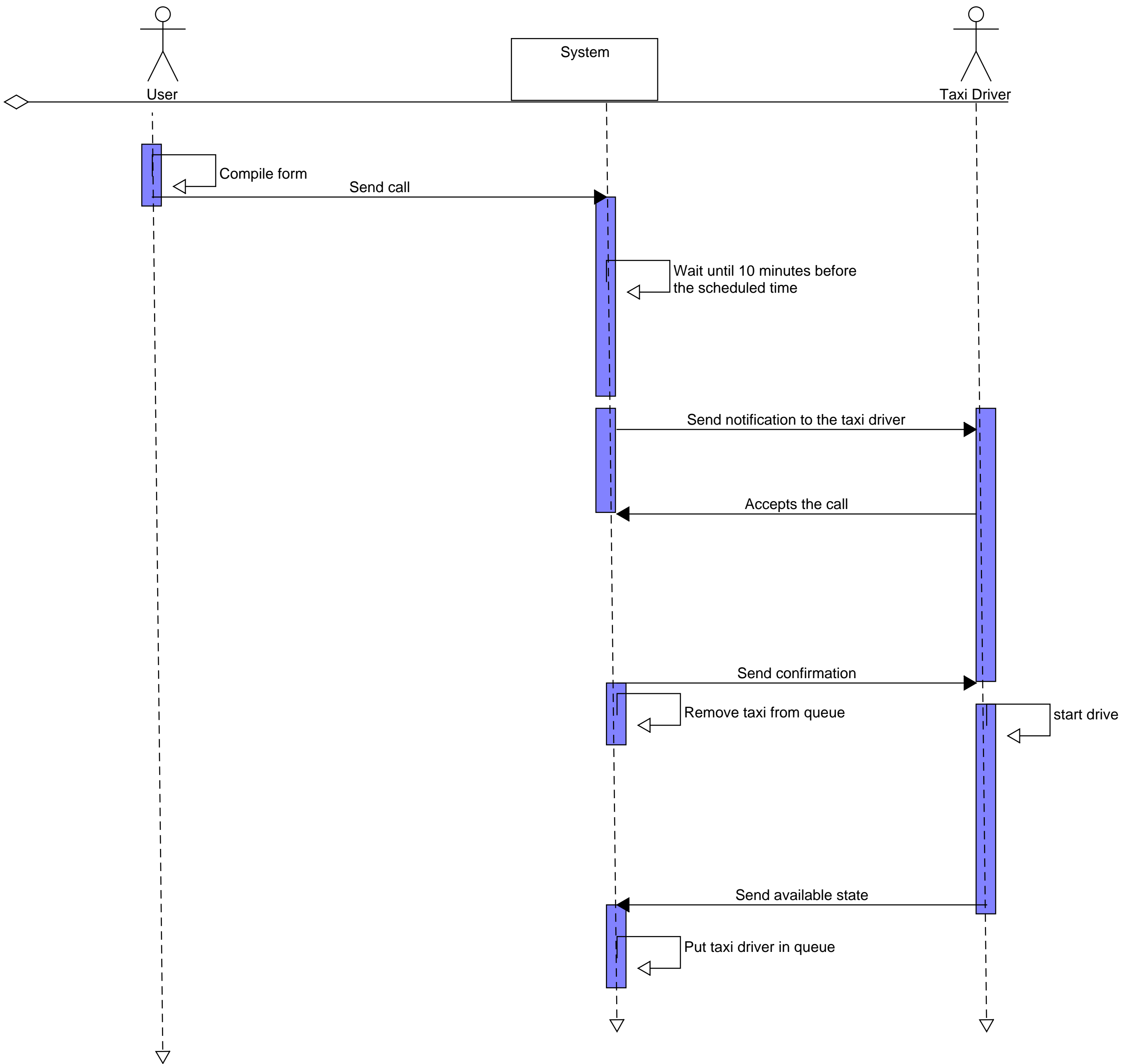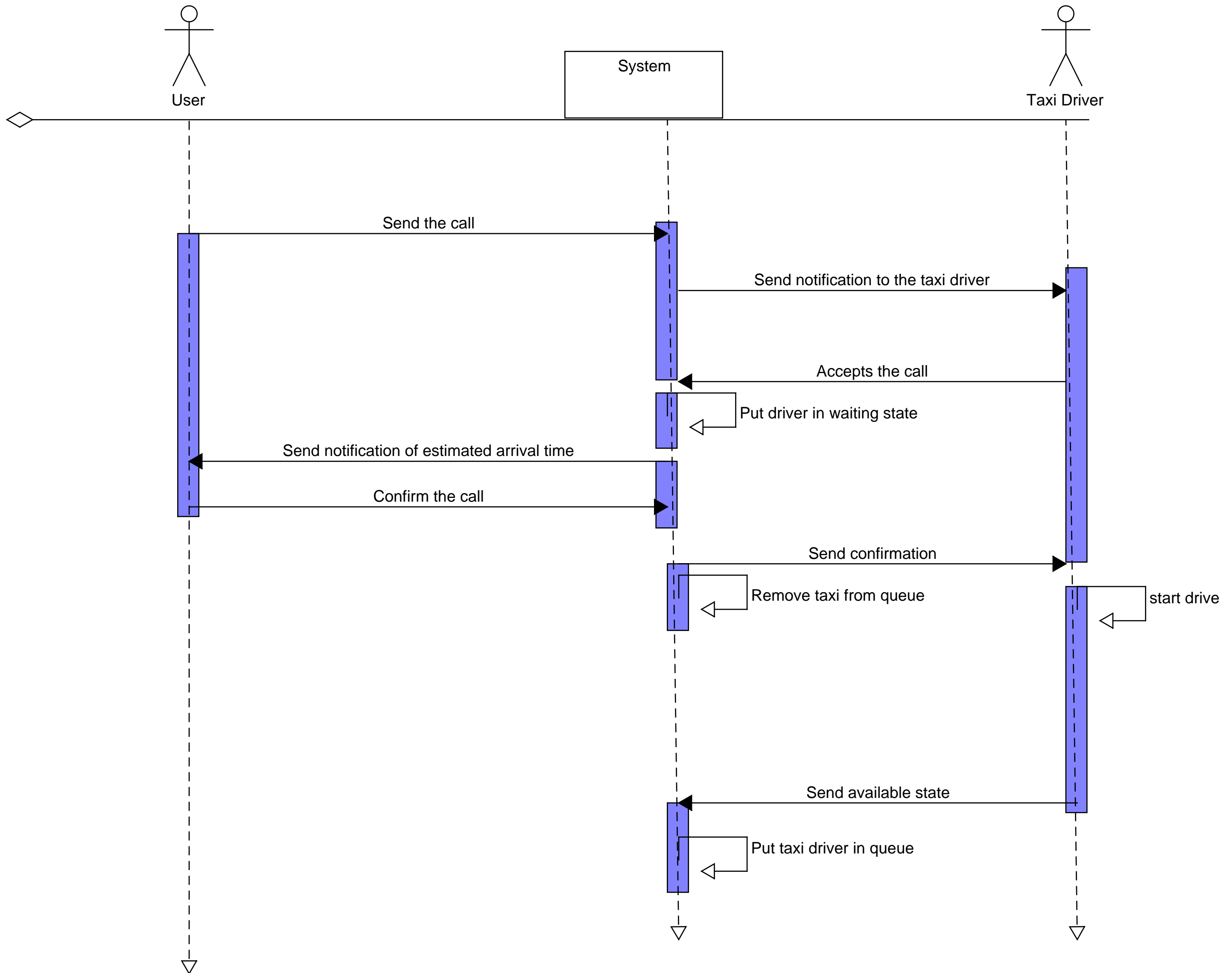|	
Name	Shared taxi request
Actors	Taxi Driver, user1, user2, user 3, taximeter
Entry conditions	User1, user 2 and user 3 have requests a taxi in the same zone, have the sharing option active and must to go in the same direction
Flow of events	the system sends a request with the users' basic information to the first available taxi driver in the zone The taxi driver accepts the request The system sends to the users the expected waiting time and the code of the incoming taxi, and places the taxi driver in "waiting" state The users confirm the request within 1 minute The system removes the taxi driver from the queue and gives him/her the confirmation The taxi driver goes to the pick up the users and bring the user 1 to his/her destination The taxi driver sends to the system the signal that taxi is arrived at the first destination The system reads the fee from the taximeter and divides it by the number of users in the taxi and saves the payed ammount The taxi arrives at the next destinations and the taxi driver sends to the system the signals that taxi is arrived The system reads the fee from the taximeter, subtracts the already payed ammount and divides the result by the number of users in the taxi The taxi driver informs the system about his/her availability.
Exit conditions	The System put the taxi driver in the last position of his/her actual zone's queue
Exceptions	The taxi driver doesn't accept the call. In this case the system forwards the request to the second in the queue and move the first taxi in the last position in the queue. One or more of the users but not all does not confirm his request after receiving notification of the expected arrival time. In this case the system notifies the taxi driver that that user refused the call All the users does not confirm the request after receiving notification of the expected arrival time. In this case the system put the taxi driver in "available" state. One or more users ar not at the pick up place when the taxi arrives. See the dedicated use case.

Fifth Use Case

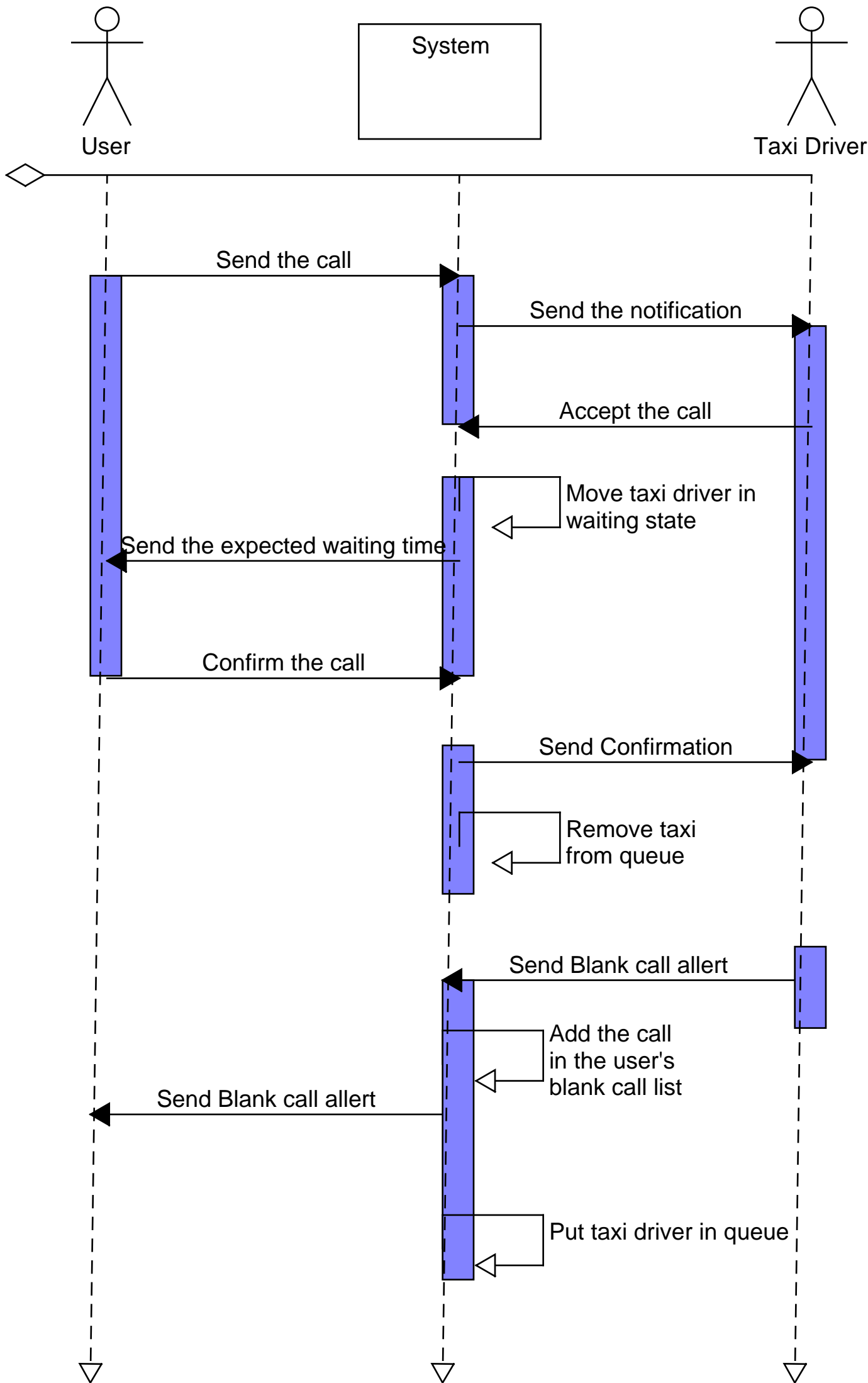|	
Name	User registration
Actors	User
Entry conditions	User is not registered yet
Flow of events	The user open the app
	The user click on "sing up" button
	The user fill the form where has to specify:
	with the "user information"
	The user click on "register" button
	The system show the "book a taxi" page
Exit conditions	Registration succesfully done
	Username already taken: when the user click on register
	the page will resfresh with a message that says "username
Exceptions	already taken"
	Not all field are filled: when the user click on register the
	page will refresh and the empty fields will be outlined in red
	The user click "back" after "sing up" the system will show
	the homepage and the registration will be discharged

Taxi request

Taxi Driver lane:
- recieve the request and user's basic information
- refuse → send the refuse
- accept → send the confirm
- recieve the confirm
- refuse
- accept → user's present / user's not present → Enter "user's not present" exception
- pick up the users
- arrive at destination
- send the notification of his availability

ERP system lane:
- Recieve the request
- ask the kind
- recieve the kind
- delayed → wait until 10 minutes before the scheduled time
- Immediate
- sends the request and user's basic information
- recieves the answer
- refuse
- accept → put the taxi driver in "waiting" state
- move the taxi driver at the end of the queue
- Send expected waiting time and the taxi's code
- recieve the answer
- refuse → send the refuse
- accept → put the taxi driver in "available" state
- remove the taxi driver from the queue
- send the confirm
- recieve the notification
- put the taxi driver in the last position of his actial zone's queue

User lane:
- Request a taxi
- recieve the kind's request
- delayed → choose the time
- immediate
- send the kind
- recieve the waiting time
- accept → send the confirm
- refuse → send the refuse

UML

User | System | Taxi Driver

Send the call

Send notification to the taxi driver

Accepts the call

Put driver in waiting state

Send notification of estimated arrival time

Confirm the call

Send confirmation

Remove taxi from queue

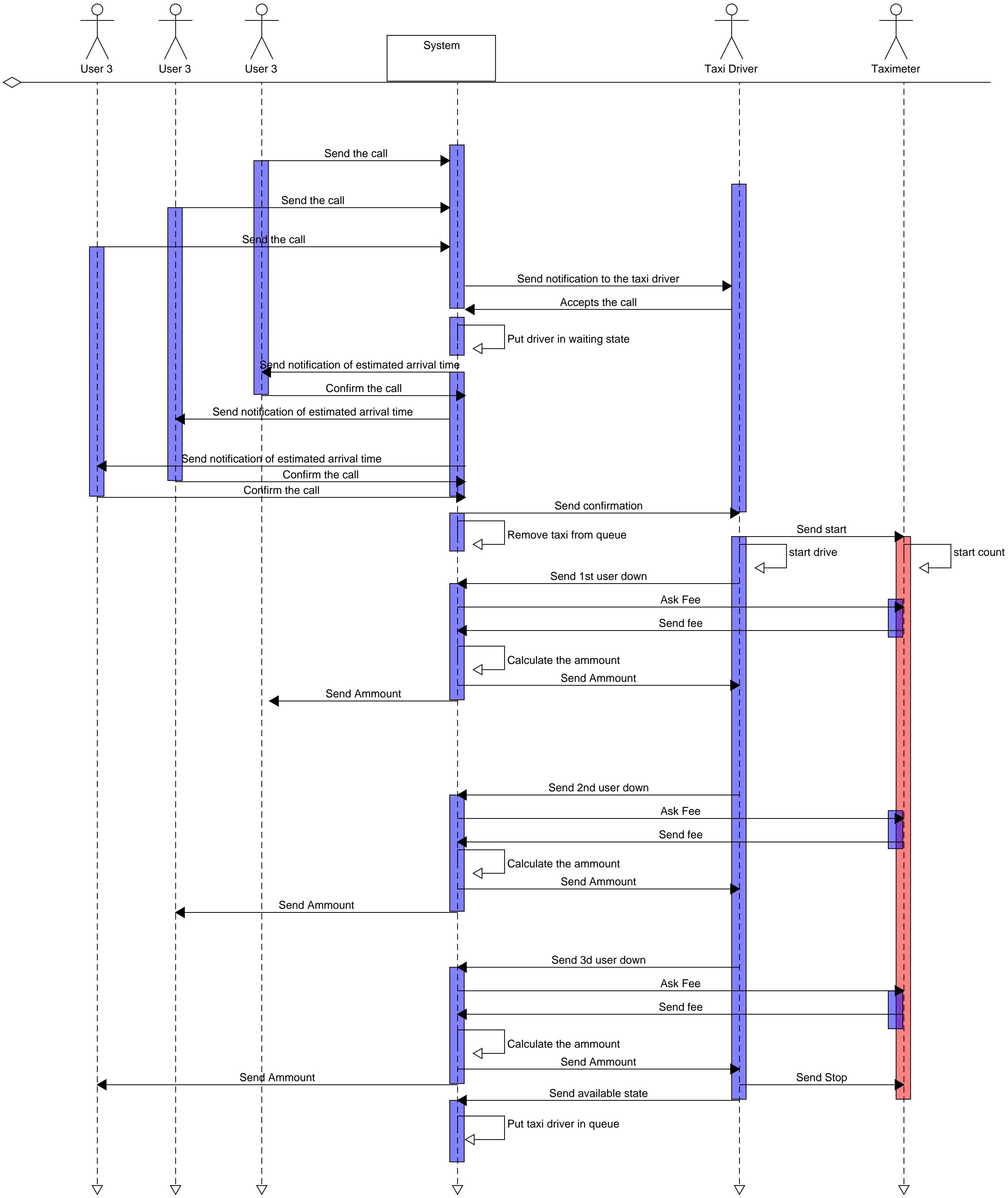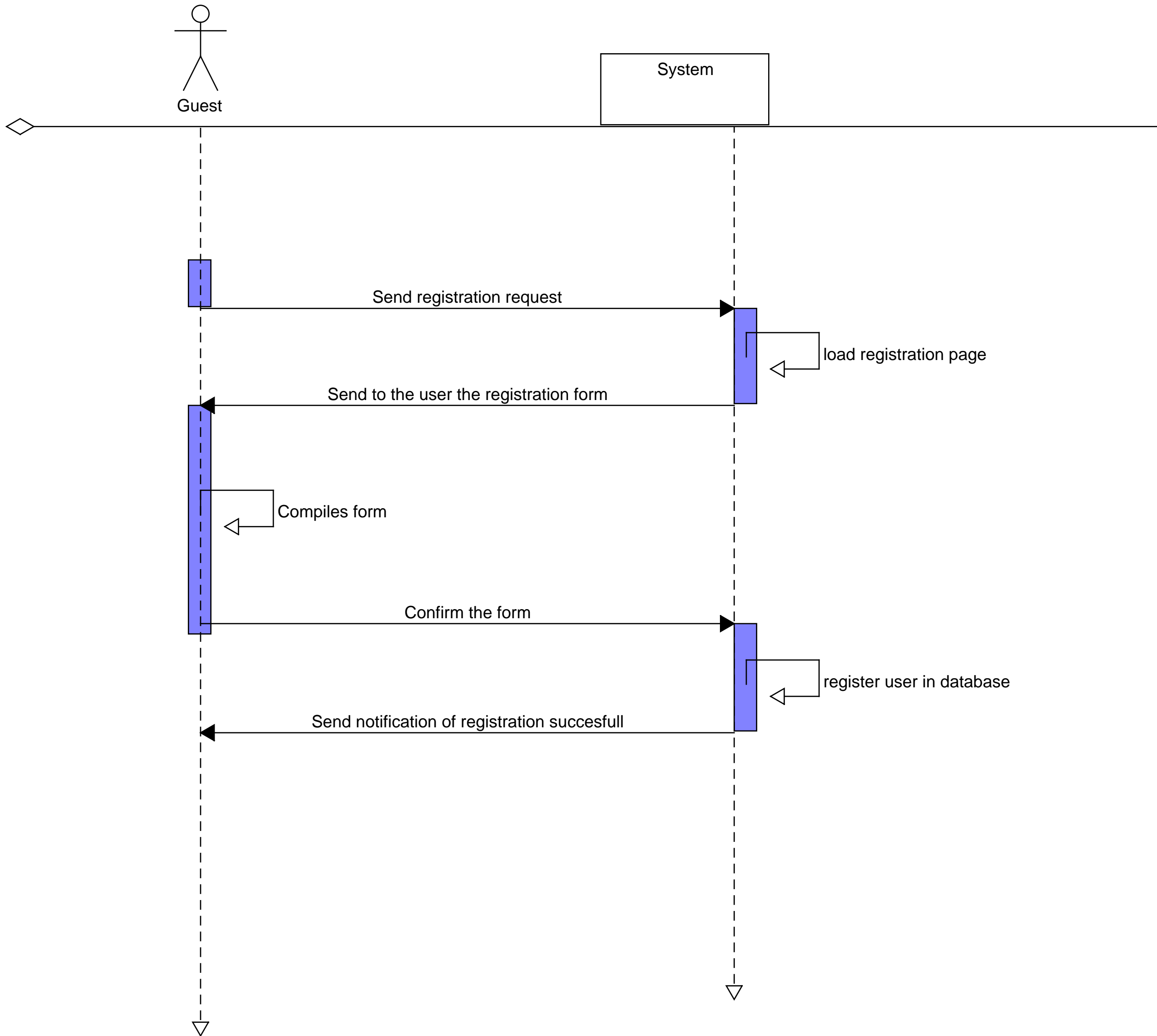start drive

Send available state

Put taxi driver in queue

Scenarios

Martin is in town center and he needs a taxi to come back home. So he log in the application and click on the button "call for a taxi", and he choose an immediate call. Then he decides to use his actual position as pick up address. The system use the gps signal of the mobile phone to find out his position and send the call to the first taxi driver in the queue of the corresponding zone. The taxi driver receives a notification of the incoming call, and he sees that the caller is named Martin Muller and he has 2 past calls, and none of them is blank. So he decides to accept the call, and click on the "accept" button on his device. The system calculates that the taxi will need approximately 6 minutes to reach the pick up place, and send a notification to Martin with this estimated arriving time. Martin can wait the 6 minutes so he decides to confirm the call, pressing the button on the application. The system removes the taxi driver from the queue and send to him a notification of the confirmed call. So he drives to the pick up place and then he brings the user to his final destination. Finally, he press the button "available" on his device to signal to the system that the ride is completed and he can accept new passengers again. The system use the gps signal of the taxi to find out the new Zone of the taxi, and then put the driver at the end of the corrisponding queue.
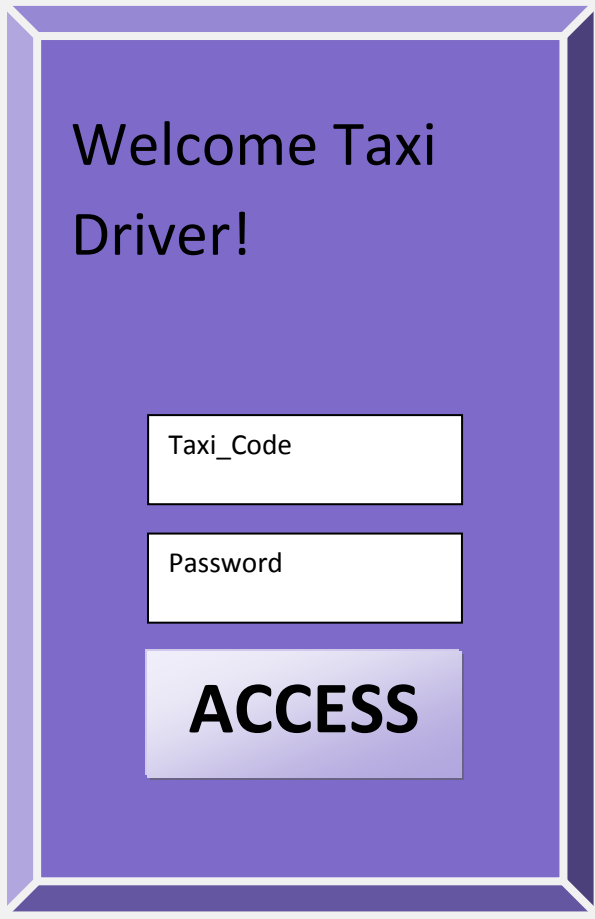
Silvia has just bought a flight ticket for next Monday, 9th of November, and she realizes that she will need a taxi to reach the airport. So, after logging in the application, she decide to reserve a taxi for that day. She press the button "call a taxi" and then she chooses the option "Delayed call". In the next view she have to write the date and time of the call, and she decides to ask for the taxy on Monday, obviusly, at 1:00 PM. Then she has to put in the pick up address, and she choose in her personal locations the location "Home", that she had already put in the system after signing up. So the location contains her home address. Then she has to put in the destination address, and she does it manually. Then she confirms the call. The system waits until monday at 12:50 PM, and then it sends an incoming call to the first taxi in the queue of the zone that contains the pick up address. In this case, no confirmation from the user is necessary. So the first taxi driver, named Jhon, decides to accept the call, that is automatically confirmed by the system, which removes Jhon from the queue. He starts from his position and reach the pick up address, waiting for Silvia to come in. She arrives few minutes after 13, and they can finally go to the Airport. When the taxi arrives there, he click on the "available" button on his device and the system puts him in the queue of the zone he reached with this ride.

Teo Gazz has just send an immediate call for a taxi in his actual position: he just finished work and he wants to go home. So Jenny, the taxi driver who received the call and accepted it, is coming to the pick up place he chose. Unfortunately, a Teo's coworker comes out form the office, searching for him: there is a huge problem in the office and he needs him to resolve it. So Teo runs back to the office with him, forgetting to cancel the call. So when Jenny arrives a the pick up place, noone is there. She waits 10 minutes, the she tries to call Teo: she saw his number in the call's informations. But noone is answering: Teo is too busy now to care aboute the phone ringing. So Jenny has no choice. She signals a blank call to the System, that put the last call of Teo in the list of his blank calls. Then the system automatically free jenny and puts her back into the queue of the zone, at the first place.

Taxi app: Taxi driver

Unique Taxi code provided by the system Password cannot be saved
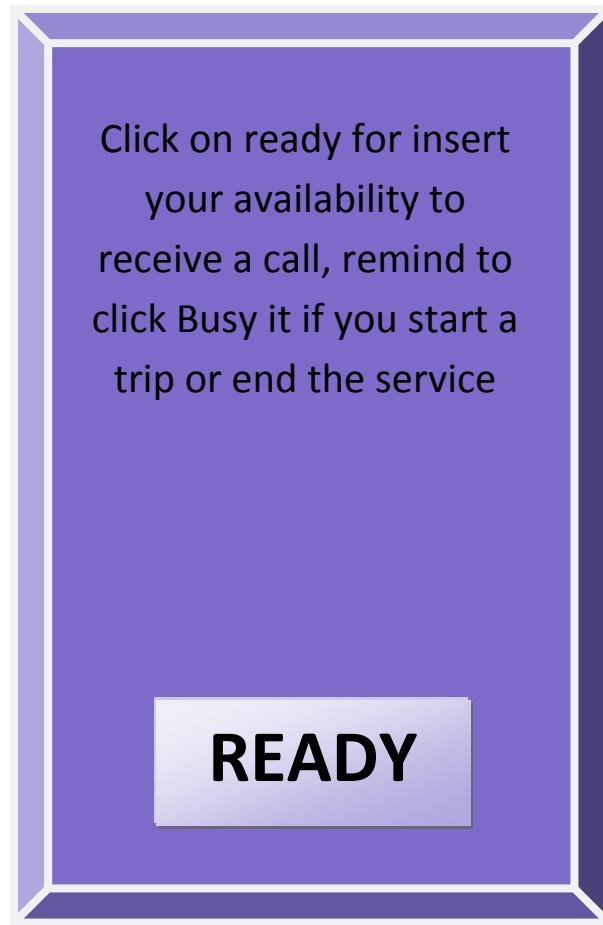
Further implementations: change password

Welcome Taxi
Driver!

Taxi_Code

Password

ACCESS

The caption of command Ready will be Busy after click

Click on ready for insert your availability to receive a call, remind to click Busy it if you start a trip or end the service

READY

⚠ INCOMING CALL from *street_name, house_nr* asked by *user_name* (*feedback*)

ACCEPT

REJECT

⚠️ INCOMING **SHARED** CALL from s*treet_name*, *house_nr* asked by *user_name* (*feedback*) to *street_name*, *house_nr* in *stop_nr* stops, for *km_nr* km
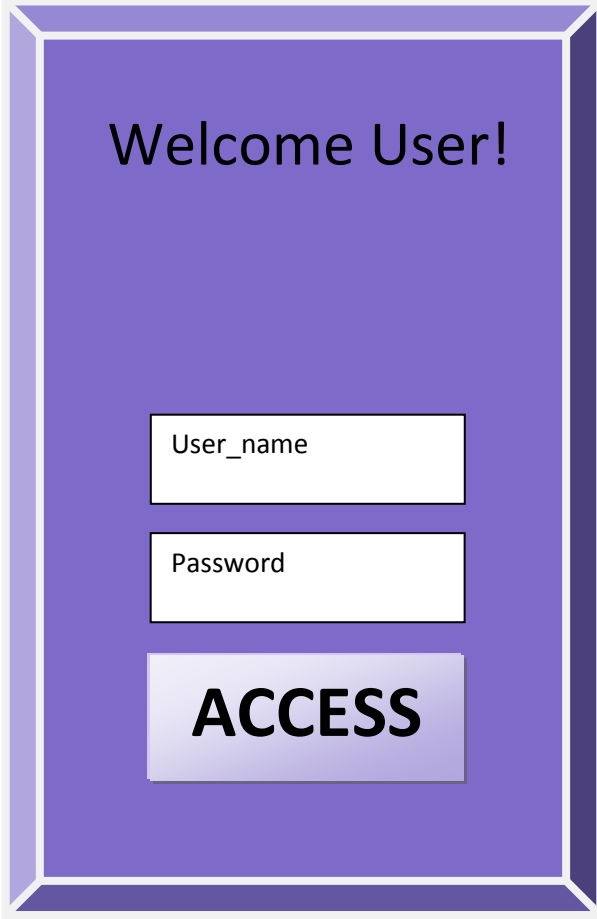
ACCEPT

REJECT

Taxi app: User

In the first tab I have choosen Access, instead of New Registration

Further implementations: save password

Welcome User!

User_name

Password

ACCESS

BOOK A TAXI

PAST CALLS

ADDRESS

ACCOUNT

SETTINGS

BOOK A
TAXI **NOW**

BOOK A
TAXI **LATER**

SHARE A
TAXI

In share a taxi, after chosen an address, system load the tab Arrive to, that have the same structure In favorite address I can select from a list In current location, after connect with GPS, I can see the map (like google maps)

In new address I can choose a new address

Departure from:

- **Favorite address**
- **Current location**
- **New address**

After the end of the booking process: the taxi driver accept the call and I accept the estimed time

⚠️At *estimed_time* the taxi driver *taxi_driver_code* (*phone_nr)* will arrive, be at the pick up point in time!
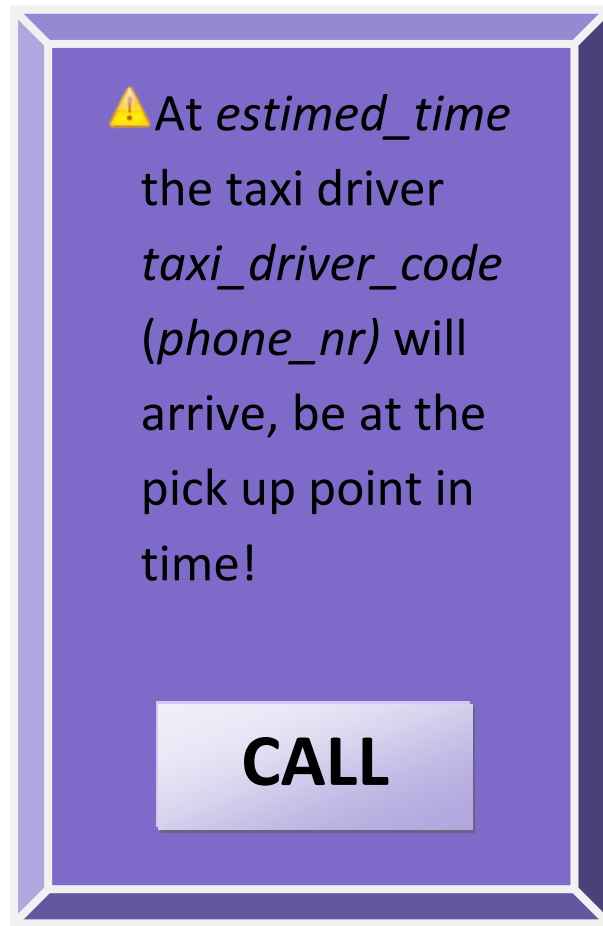
CALL

ALLOY

```
abstract sig Person{
    }

sig Taxi_driver extends Person{
    incoming:lone Call  }

sig User extends Person{
    location:set Location,
    past_call:set Call,
    blank_call:set Call}

abstract sig Call{

    }

sig Delayed_call extends Call{
    caller:one User,
    start: one Address,
    destination: one Address
    }

sig Immediate_call extends Call{
    caller:one User,
    start: one Address
    }

sig Shared_call extends Call{
    start: some Address,
    destination: some Address,
    caller: some User}

sig Zone{
    address: some Address    }

sig Queue {
    zone:one Zone,
    drivers: set Taxi_driver}

sig Location{
    address: one Address}

sig Address{
    }

//FACTS

fact OneDriverPerCall{
    //ad ogni call deve corrispondere un tassista se attiva oppure essere un past call di un
    utente
    all c:Call | (lone t:Taxi_driver | t.incoming=c )
    all c:Call| (some u:User |(no t:Taxi_driver | c=t.incoming <=> (c in u.past_call)))
}

fact BlankCalls {
    //le chiamate perse non possono essere più di quelle effettuate e devono essere
    contenute in quelle passate
    all u:User|(all c: Call |(c in u.blank_call implies c in u.past_call))}
```

```
fact MaxOneActiveCall {
    //un utente può avere solo una chiamata attiva
    all u:User, i1:Immediate_call, i2:Immediate_call | ((i1!=i2) implies (i1.caller=u
    implies (i2.caller != u or i2 in u.past_call)))
    all u:User, s1:Shared_call, s2:Shared_call | ((s1!=s2) implies (u in s1.caller implies
    (u not in s2.caller  or s2 in u.past_call)))
    all u:User, s:Shared_call, i:Immediate_call | ((u in s.caller implies (u != i.caller  or
    i in u.past_call))and (u = i.caller implies (u not in s.caller  or s in u.past_call)))
    }

fact DestinationNotStart {
    //una destinazione non può essere la partenza
    all i:Delayed_call | i.start!=i.destination
}

fact NotEmptyQueue{
    //in ogni coda deve contenere almeno un tassista
    all q:Queue | (some t:Taxi_driver | t in q.drivers)
}

fact NumberAdressesShared{
    //partenze e arrivi al massimo uguali al numero di utenti
     all c:Shared_call | (#c.start <=#c.caller && #c.destination<=#c.caller)}

fact SameStartZone{
    //tutti gli utenti devono partire dalla stessa zona
    all c:Shared_call|(one z:Zone |c.start in z.address)
}

fact QueueInZone {
    //ogni zona deve avere una e una sola queue
    all z:Zone |(one q:Queue |q.zone=z)
}

fact OnlyMyCalls {
    //ogni utente ha in lista solo le sue calls
    all u:User |(all c:Immediate_call|(c in u.past_call implies c.caller=u))
    all u:User |(all c:Delayed_call|(c in u.past_call implies c.caller=u))
    all u:User |(all c:Shared_call | (c in u.past_call implies u in c.caller))
    }

fact OneQueuePerTaxi{
    //un tassista deve essere in una sola coda
    all t:Taxi_driver |(lone q:Queue |t in q.drivers)
}

fact LocationInUser{
    //una location di una chiamata deve essere nelle location dell'utente
    all l:Location|(one u:User | (l in u.location))}

fact NoOrphanAddress{
    //non ci sono indirizzi senza zona
    all a:Address| (one z:Zone |(a in z.address))}


//ASSERTIONS
```

```
assert NoOrphanCalls {
    //controlla che non ci siano chiamate senza tassista o utente
    no c: Call | ((no u: User | c in u.past_call) && (no t: Taxi_driver | t.incoming=c))
}
check NoOrphanCalls

assert NoDifferentStart {
    //controlla che non ci siano utenti che partono da zone diverse
    all c : Shared_call, a1,a2 : Address |(a1 in c.start and a2 in c.start implies (one
    z:Zone| a1 in z.address and a2 in z.address))
}
check NoDifferentStart

assert MaxBlank {
    //controlla che le chiamate perse non siano maggiori di quelle effettuate
    all u:User | (#u.blank_call<=#u.past_call)}
check MaxBlank



//PREDICATES

//un mondo con tassisti non in coda
pred DriverNotinQUeue {
some t:Taxi_driver| (all q: Queue | t not in q.drivers)
some Location
some Immediate_call
 }
run DriverNotinQUeue


//un mondo con utenti che chiamano ma non prendolo il taxi
pred UserNoRide{
all c:Call|(some u:User | c in u.blank_call)
}
run UserNoRide

//un mondo con almeno 2 utenti che chiamano un taxi condiviso
pred Sharing{
some c:Shared_call | #c.caller>1 and (no u:User | c in u.past_call)
}

run Sharing

//un mondo in cui un utente ha prenotato una chiamata da una zona preferita ad un'altra
pred Favorite {
one u:User| (some c:Delayed_call |( c.start in u.location.address and c.destination in
u.location.address and c not in u.past_call))
}

run Favorite

//un mondo con 2 zone
pred TwoZones{
#Person>2
#Zone>1
```

```
}
run TwoZones

//partenza e arrivo in zone diverse
pred DifferentZone{
one c:Delayed_call | some z1,z2:Zone | (c.start in z1.address and c.destination in
z2.address and z1!=z2)
}
run DifferentZone
```

ALLOY Tests

Executing "Check NoOrphanCalls"
 Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
 2413 vars. 192 primary vars. 3984 clauses. 10ms.
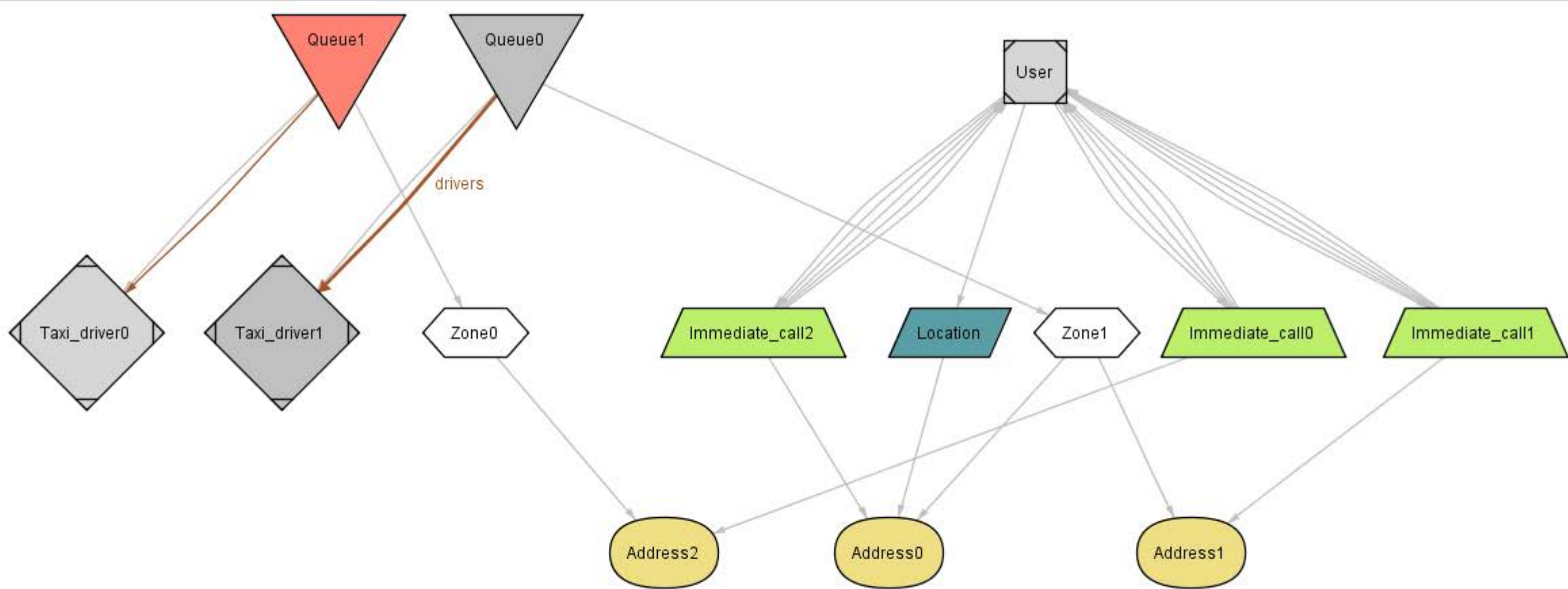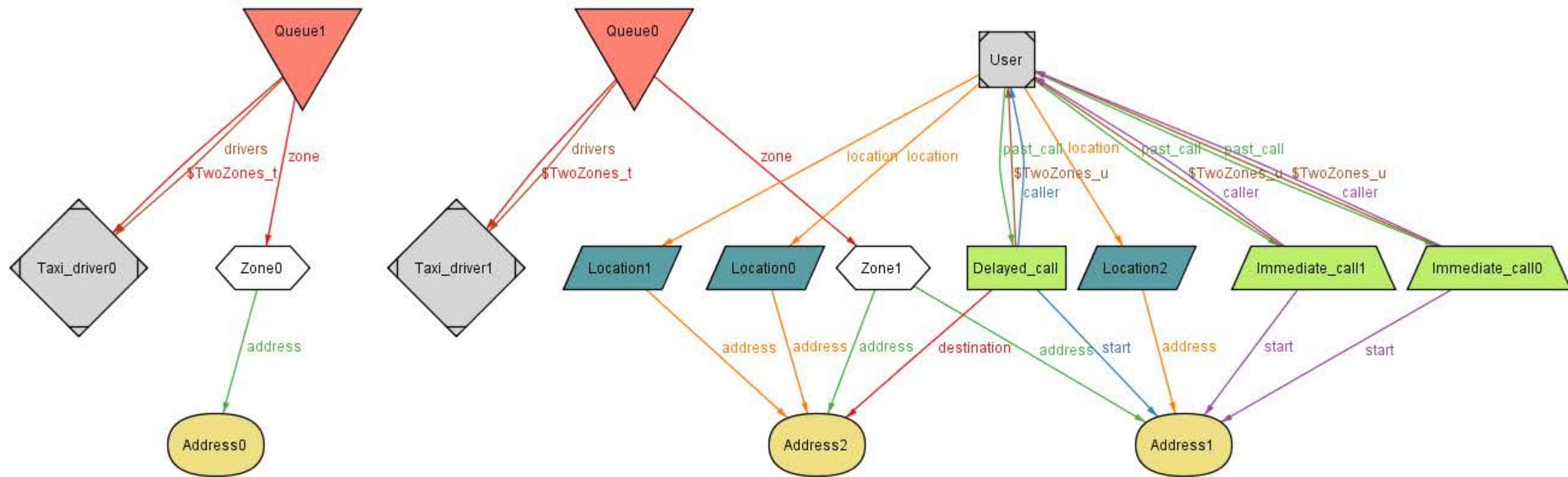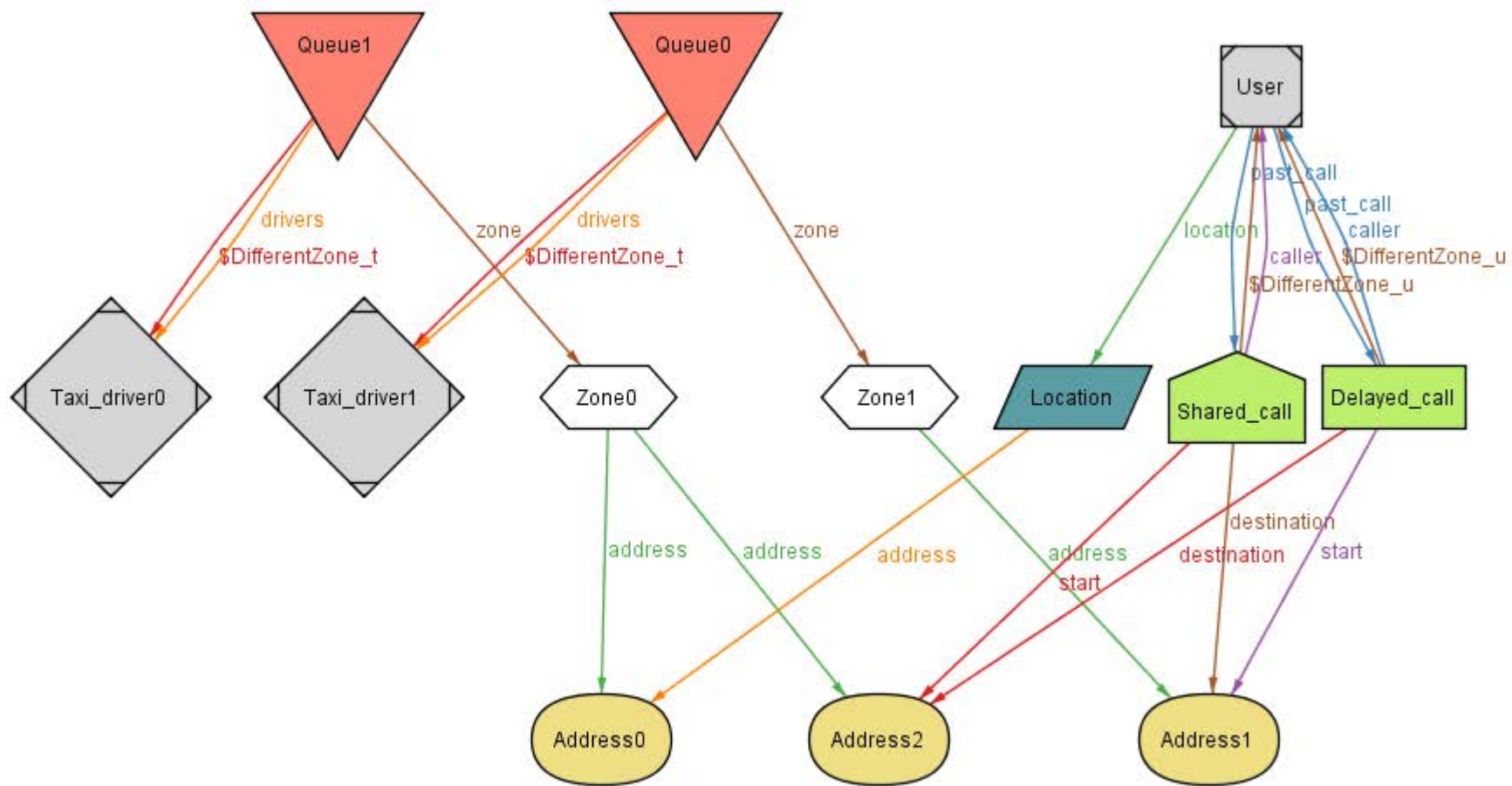 No counterexample found. Assertion may be valid. 3ms.

Executing "Check NoDifferentStart"
 Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
 2438 vars. 198 primary vars. 4067 clauses. 8ms.
 No counterexample found. Assertion may be valid. 2ms.

Executing "Check MaxBlank"
 Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
 2412 vars. 192 primary vars. 4049 clauses. 9ms.
 No counterexample found. Assertion may be valid. 2ms.

$UserNoRide_t: 2
$UserNoRide_u: 3
$UserNoRide_u': 3
address: 1
address: 3
blank_call: 3
caller: 3
drivers: 2
location: 1
past_call: 3
start: 3
zone: 2

Queue1

Queue0

User

drivers

Taxi_driver0

Taxi_driver1

Zone0

Immediate_call2

Location

Zone1

Immediate_call0

Immediate_call1

Address2

Address0

Address1