COMPUTER SCIENCE AND ENGINEERING

PROJECT OF SOFTWARE ENGINEERING 2

Requirements Analysis and Specification Document

POLITECNICO
MILANO 1863

Author: Riccardi Vincenzo 793241

Reference Professor: Mirandola Raffaela

Summary

Summary……1.

Introduction……2.

Overall Description……5.

Requirements……8.

Scenarios……13.

UML Models……17.

Alloy……38.

1.Introduction

Purpose

The present document is written to describe the functionalities of a new application for helping people to find and call taxi rapidly. The document concern is to inform the potential developers and the stakeholder of the project, to make the functionality and requirements of web application.

Goals

The software will have the following goals:

- Allow user to log in to application.
- Allow user to look for an available taxi.
- Allow a visitor to became a registered user.
- Allow user to call a taxi.
- Allow user to insert his position manually or automatically with GPS.
- Show the user the waiting time and the code of taxi.

- Allow user to insert a destination and show him an estimate of the cost.
- Allow user to delete the account.
- Send an email to user to confirm the email used in registration.
- Allow user to resend the confirmation email .
- Allow user to insert information about the number of baggage.
- Allow taxi driver to inform the system about their availability and to confirm
  that they are going to take care of a certain call.

Limitation

The software will have the following limitations:

- Only registered user can use the application.
- It is not possible stop a call in progress.
- It is not possible change the email or other account information.
- It is not possible use the application in the outside of city.
- It is not possible choose the taxi.

Actors

- Visitors: all people who can only see the login page and if complete the registration form to be able to access to the functionality of the application.
- Registered user: this people, after registration, can use the web or mobile application to look for taxi and call it.
- Taxi driver: all people who are always connected to mobile application.

Reference

• IEEE Recommended Practice for Software Requirements Specifications: http://standards.ieee.org/findstds/standard/830-1998.html .

• Alloy code – "myTaxiService.als" for more information follow the link http://alloy.mit.edu/alloy/).

• UML Class Diagram (attached to the document) - "UML Class Diagram.png".

2.Overall Description

Assumptions

Our goal is to create a project free of ambiguity, so we had to assume some facts.

We assume that:

- The taxi drivers do not have a registration form but they are added automatically by the system.
- Each taxi have a device on which it is installed the mobile application.
- Each taxi have a GPS system to identify it in the city.
- Do not exist two taxi with the same code.
- For each zone the queue of taxi is never empty.
- Each taxi has a capacity of baggage that may contain, specified with an integer number.
- Each taxi has a capacity of people that may contain.

Architecture

The architecture is based on Client-Server structure. The client (user) sends the request to the server that communicates with client (taxi-driver). Taxi-driver confirms or not the availability and sends the answer to the server that communicate again with the user.

Interfaces

- User interfaces :

  Client and server have an interface. The server interface is a simple terminal written by the developers. Client interface is a set of screen with a different buttons that allow the navigation. In particular the main page would contain:

  - A screen containing the Log In and Register buttons for people who are not registered or not logged in.
  - A screen containing the Call Taxi button, the Insert Position button, code and waiting time box for people who are registered and logged in.

  In taxi driver devices there is not a GUI but only physical buttons.

- Software interfaces:

  The principal client software interface is web browser.
  The browser that the application supports are Google Chrome, Mozilla Firefox, Internet Explorer, Safari and Microsoft Edge.
  The principal server software interface is Database management system, in particular MySQL.
  http://www.mysql.com/downloads/mysql/

For the connection between server and client is used TCP (Transmission Control Protocol), while HTTPS is used for the exchange of data.

Programming Language

The high language imposed is Java 8 (Oracle); DBMS used is MySql Community Edition. HTML and CSS for the web interface.

3.Requirements

Functional requirements

The functional requirements describe what the software have to do in the interaction with the user.

The application:

- Allow a visitor to became a registered user.
- Allow user to log in to application.
- Allow user to call a taxi.
- Allow user to insert a position with or without GPS.
- Allow user to specify the number of baggage and people.
- Allow user to see the waiting time and the code of taxi.
- After the registration, the system will send the user a confirmation email .
- Allow taxi driver to confirm or not his availability.

Non-functional requirements

User interface

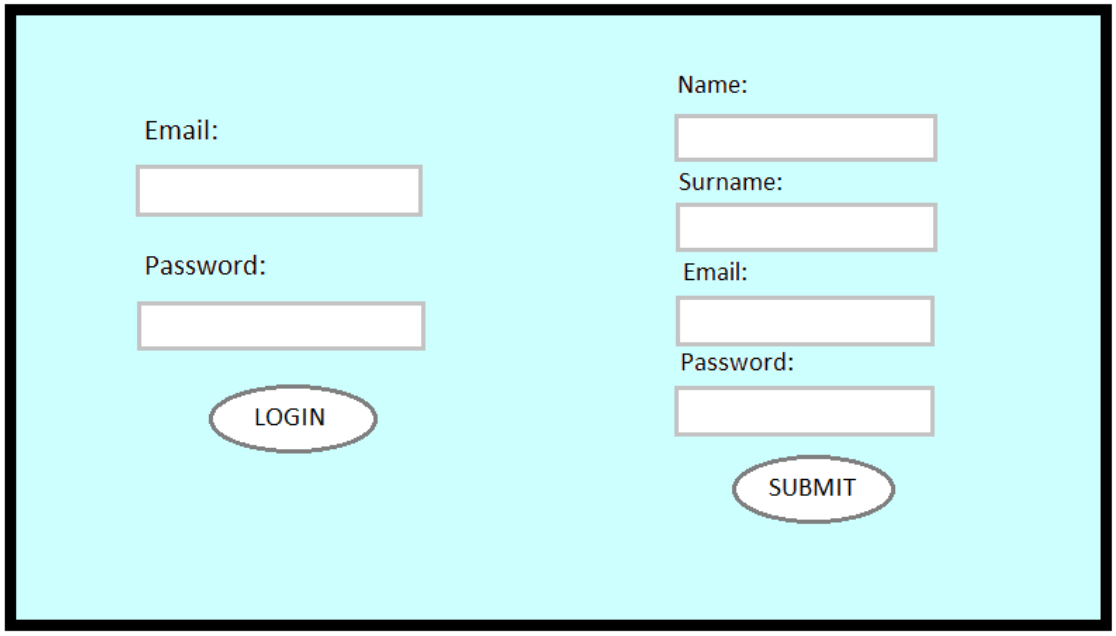
To describe the User interface we want to provide three sketches of the most important pages. With this sketches we want to focus on the structure of the user interface and not on the design.

The principal non-functional requirements are:

- Software System Attributes :
  a) Availability, the application will be accessible online anytime.
  b) Portability, the application could be used on any SO which support Java Virtual Machine and Database Management System.
  c) Security, the system must protect the information of users (password, username, email). For example, the password must contain 8 character with number, letter and special character.

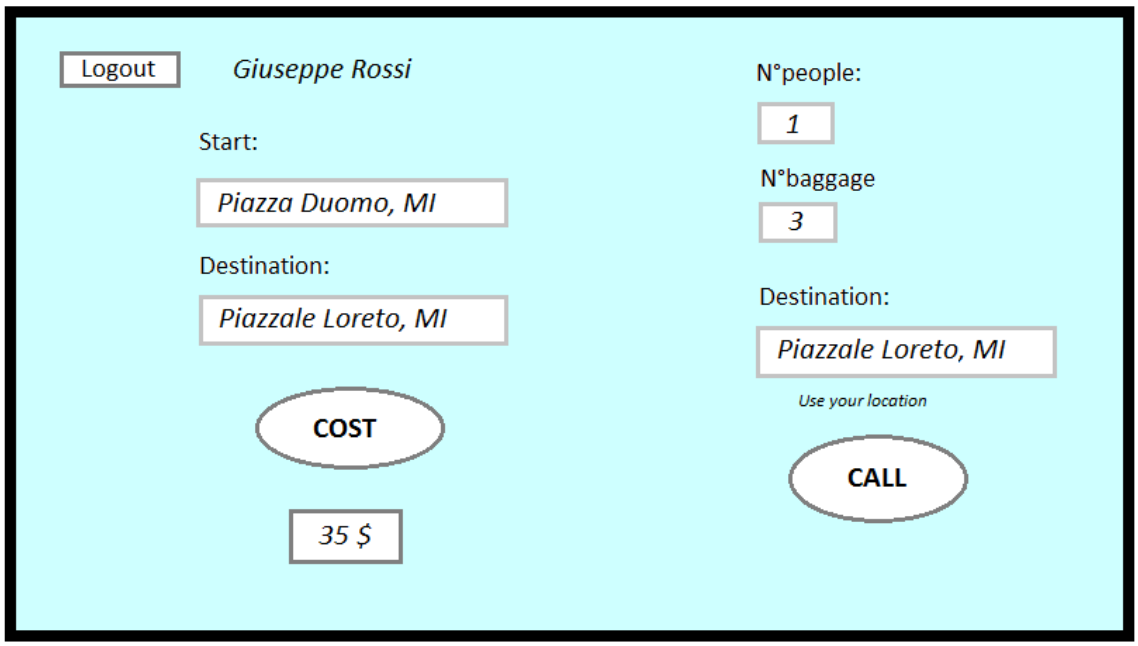
The first is the login/registration page. In this page the user can log in the application or can become a new user.

We can see below the first sketches.



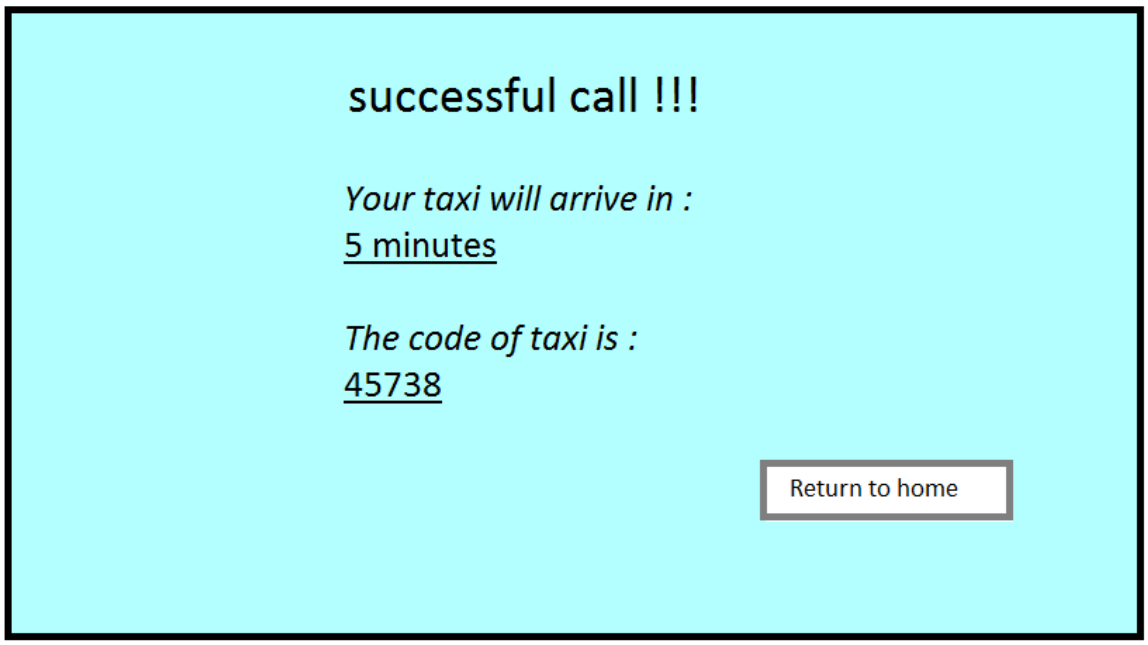
The second is the home page. In this page the user can call a taxi or can see an estimate cost of its journey, also the user can use the GPS system to insert its location.

We can see below the second sketches.



The last sketches is the confirmation page. In this page the user can see the time to wait and the code of his taxi.

We can see below the sketches.



4.Scenarios

1. Mario and Alessandro are two guys who arrive in Milano Garibaldi at nine o'clock, because of the strike of ATM agency they can't take metro and tram, so Alessandro decided to use myTaxiService application in his mobile phone to call a taxi.
   Two days ago Alessandro completed registration to the application, so insert the correct email and password in their box and click "LOGIN" button.
   Alessandro logs in the application, but meantime the mother of Mario decided to go get them by car, so immediately Alessandro click "LOGOUT" button end close the application.

2. Alessia is a girl who attends the high school, so one day Katia who is her schoolmate recommended to download myTaxiService application.
   When Alessia returns to home download the new application, so she decided to complete the registration, at first she inserts an incorrect email and the system alerts her with a message. So she tries another email and insert a password which contain 8 letter, a special character and a number, this time everything is correct and the system sends an email confirmation.
   Alessia logs in her webmail account and click the link content in the email received to complete the registration. Now Alessia can use the application.

3. Ten people with lots of baggage arrive at the central station in Milan to visit the city. One of them decides to use myTaxiService to call a taxi, but he have not a mobile telephone and uses his computer to connect at the web site of application. After login, he inserts the number of people, the number of baggage and his location (writing it in the text box) and click “CALL” button. Automatically the system receives the informations about people, baggage and location and confirm the reservation, in fact in the display the guys could have seen the waiting time and the codes of taxi which will come.

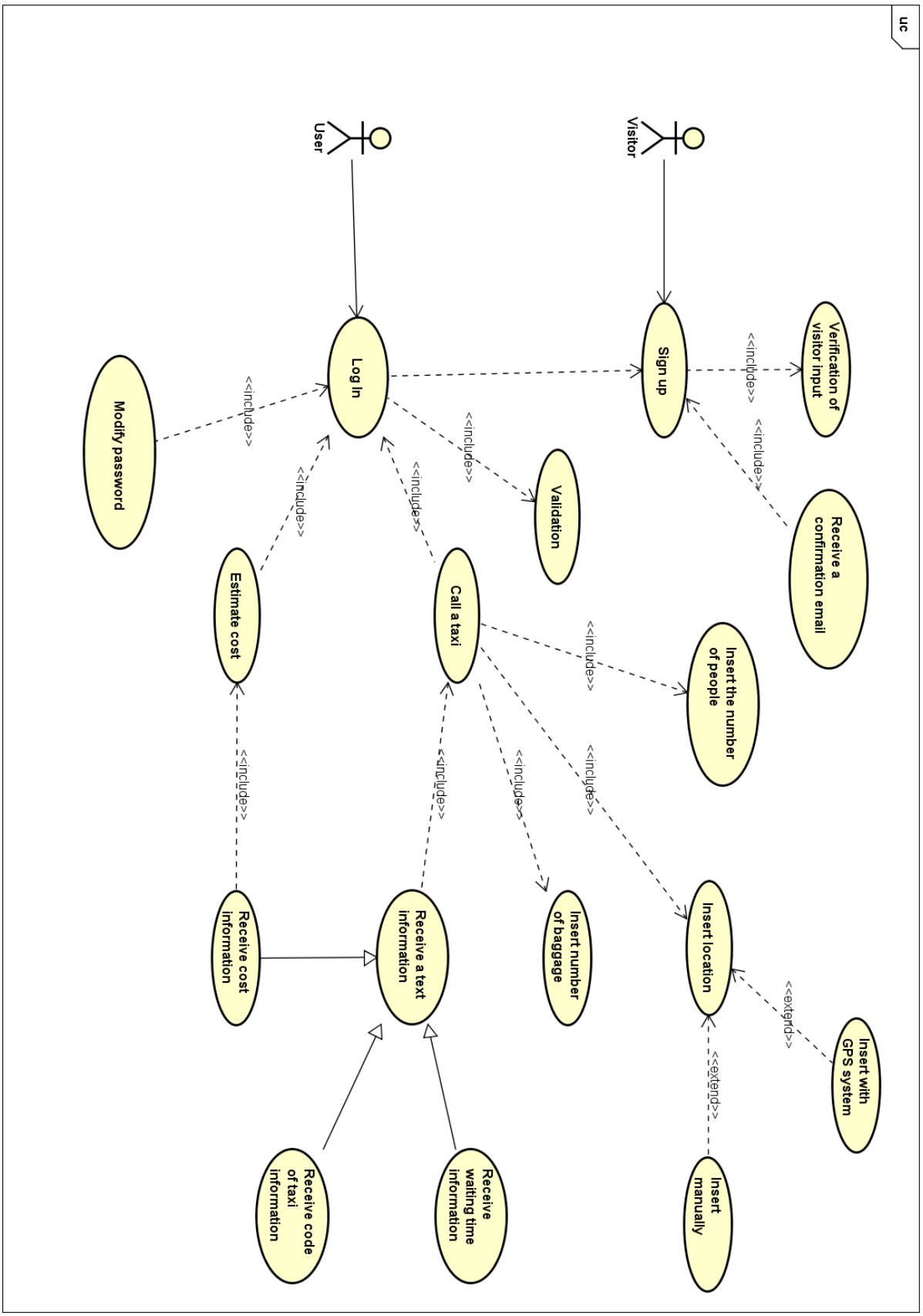
4. Marco arrives at central station in Milan at 8 o'clock, he needs to take a taxi but Marco have not much money. Fortunately he remembers that myTaxiService application can estimate the fare, so after the login, he inserts the start and the destination location and press the "COST" button. The system elaborates the information and Marco can see the cost of his journey in the text box. In the end Marco inserts his location by clicking on the GPS icon and he press "CALL" button to call the taxi.

5. A group of friends is out of town when suddenly run out the gasoline in the car. A guy tries to connect in internet but his phone gets bad, so he can't log in the myTaxiService application. Another guy logs in the application but when he inserts location and press "CALL" button the system notifies him that the application does not work out of the town.

5.UML Models

Use Case Diagram

A **use case diagram** at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.


uc
Visitor
User
Sign up
Log In
Verification of visitor input
Receive a confirmation email
Validation
Modify password
Call a taxi
Estimate cost
Insert the number of people
Insert location
Insert with GPS system
Insert manually
Insert number of baggage
Receive a text information
Receive cost information
Receive code of taxi information
Receive waiting time information
<<include>>
<<extend>>

Use Case Description

We described in a detailed way the use cases.

We refine the use case “Login”.

NAME	Log in
ACTORS	User
PRE-CONDITION	User has successfully registered to the application.
FLOW OF EVENTS	1. The user enters in web site or mobile application. 2. The user writes in the fields the username and password. 3. The user click on the “LOGIN” button.
POST-CONDITION	The system shows the home page.
EXCEPTIONS	The password or username inserted are wrong.

We refine the use case “Sign Up”.

NAME	Sign up
ACTORS	Visitor
PRE-CONDITION	User is not registered to the application.
FLOW OF EVENTS	1. The visitor enters in the web site or in mobile application. 2. The visitor write in text fields: • Name • Surname • Email • Password 3. The visitor click on the “SUBMIT” button.
POST-CONDITION	The system sends an email confirmation.
EXCEPTIONS	The password or email do not meet the requirements.

We refine the use case “Call a taxi”.

NAME	Call a taxi
ACTORS	User
PRE-CONDITION	The user logs in the home page.
FLOW OF EVENTS	1. The user enters in home page of application. 2. The user specifies in the box: • Number of people • Number of baggage 3. The user write his location in text box or use GPS to find it clicking on the GPS icon. 4. The user click on the “CALL” button.
POST-CONDITION	The system shows the confirmation page.
EXCEPTIONS	If the user not specifies number of people and baggage, the system uses the default value (number of people and baggage = 1).

We refine the use case “Estimate Cost”.

NAME	Estimate cost
ACTORS	User
PRE-CONDITION	The user logs in the home page.
FLOW OF EVENTS	1. The user enters in home page of application. 2. The user specifies in the box: • The start of journey. • The destination of journey. 3. The user click on the “COST” button.
POST-CONDITION	The system show him the cost of journey.
EXCEPTIONS	The system does not recognize the start or destination.

We refine the use case “Receive Cost Information”.

NAME	Receive cost information
ACTORS	User
PRE-CONDITION	The user fill in all fields and click on the “COST” button.
FLOW OF EVENTS	The system shows him the estimated cost of the journey.
POST-CONDITION	The user remains logged in the home page.
EXCEPTIONS	

We refine the use case

“Receive Waiting Time Information”.

NAME	Receive waiting time information
ACTORS	User
PRE-CONDITION	The user fill in all fields and click on the “CALL” button.
FLOW OF EVENTS	The system shows him the information page and the waiting time.
POST-CONDITION	The user remains logged in the information page.
EXCEPTIONS	

We refine the use case

“Receive Code of Taxi Information”.

NAME	Receive code of taxi information
ACTORS	User
PRE-CONDITION	The user fill in all fields and click on the “CALL” button.
FLOW OF EVENTS	The system shows him the information page and the code of taxi.
POST-CONDITION	The user remains logged in the information page.
EXCEPTIONS	

We refine the use case “Insert Location”.

NAME	Insert location
ACTORS	User
PRE-CONDITION	The user is on home page of application.
FLOW OF EVENTS	The user writes the destination manually or he uses the GPS to find his location clicking on the GPS icon.
POST-CONDITION	The user remains logged in the home page.
EXCEPTIONS	The system does not recognize the location.

We refine the use case “Insert Number of Baggage”.

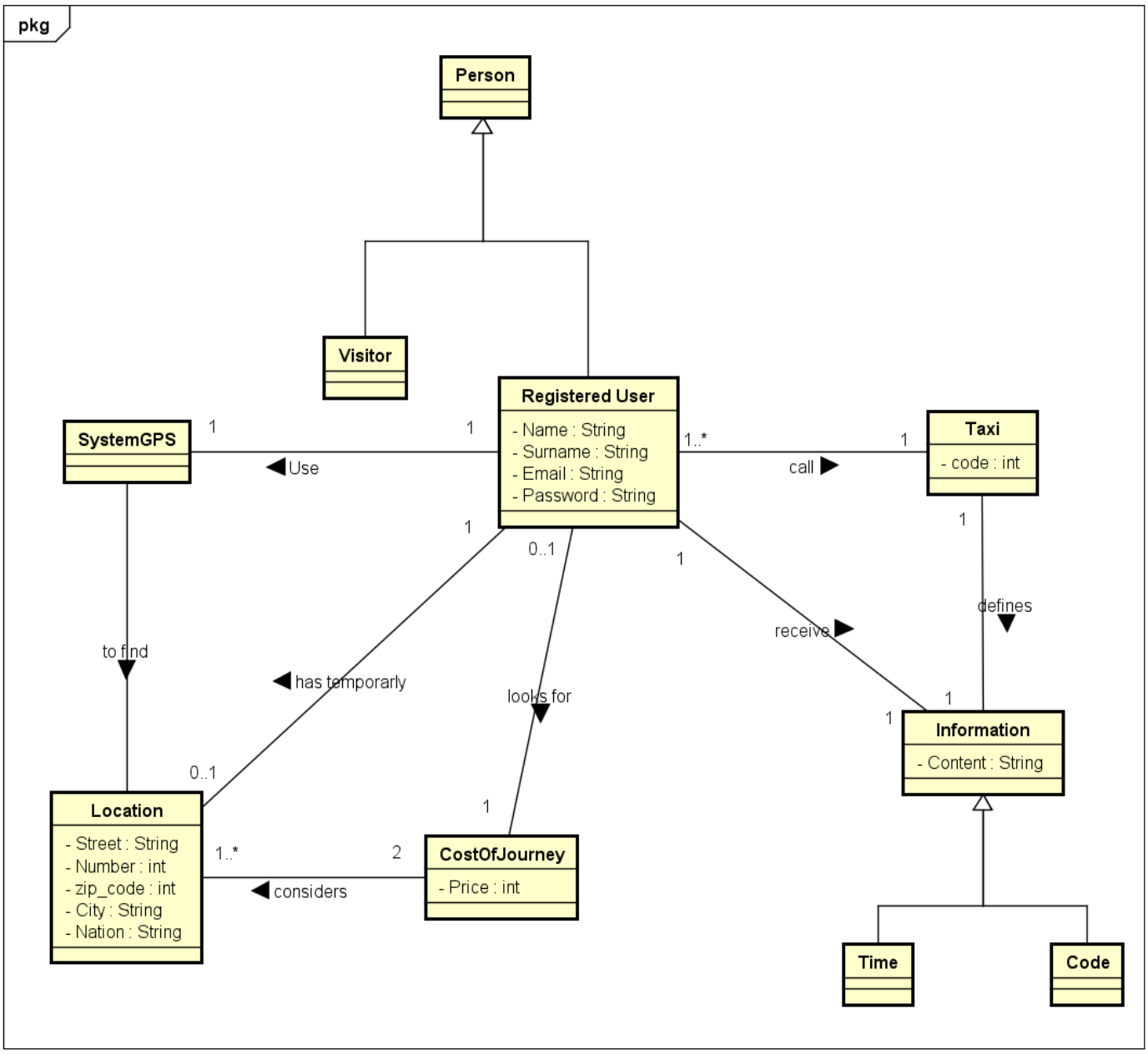
NAME	Insert number of baggage
ACTORS	User
PRE-CONDITION	The user is on home page of application.
FLOW OF EVENTS	The user writes the number of baggage in text box.
POST-CONDITION	The user remains logged in the home page.
EXCEPTIONS	If the user not specifies number of baggage, the system uses the default value (number of baggage = 1).

We refine the use case “Insert Number of People”.

NAME	Insert number of people
ACTORS	User
PRE-CONDITION	The user is on home page of application.
FLOW OF EVENTS	The user writes the number of people in text box.
POST-CONDITION	The user remains logged in the home page.
EXCEPTIONS	If the user not specifies number of people, the system uses the default value (number of people = 1).

Class Diagram

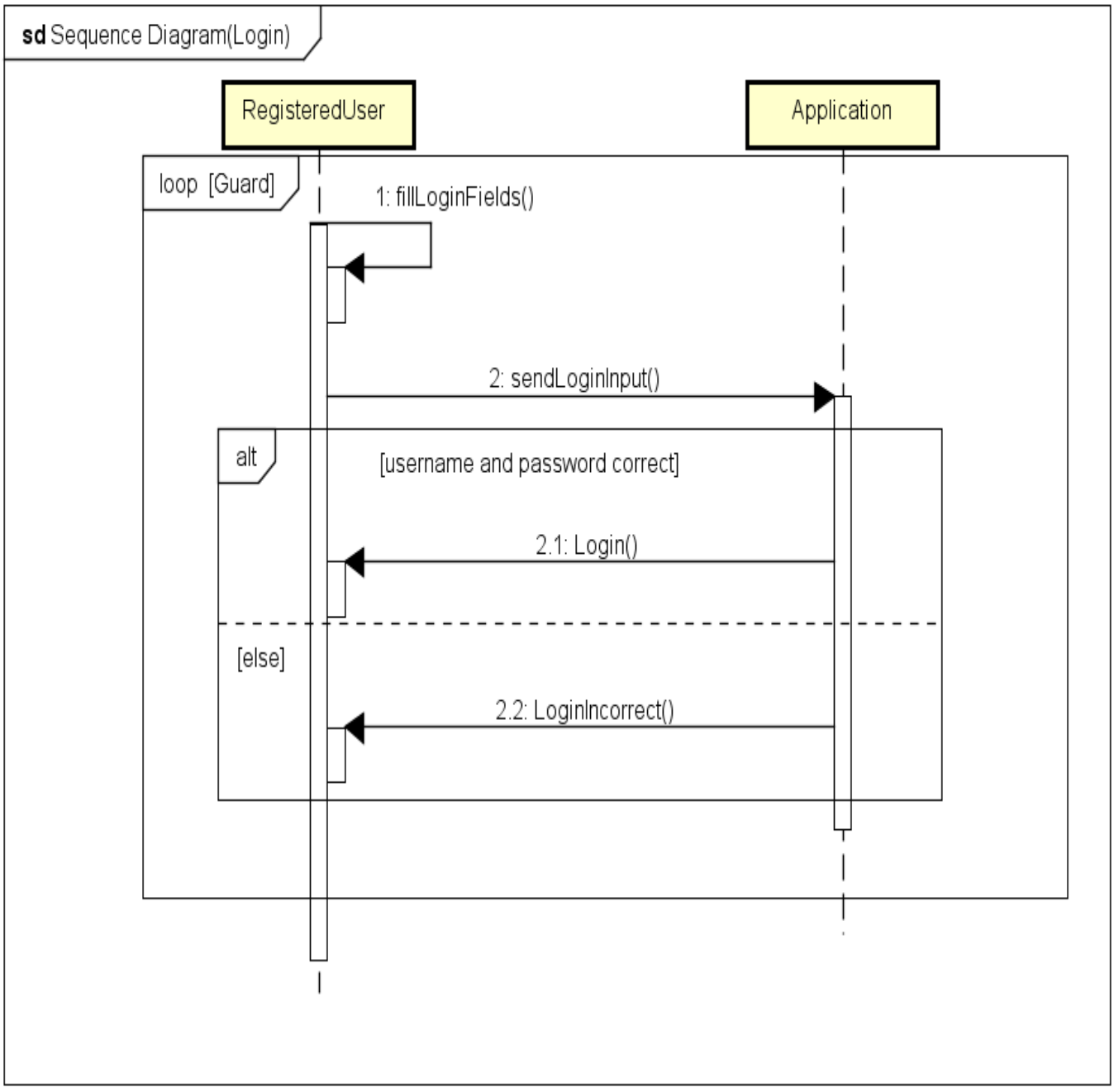
A **class diagram** in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.


pkg
Person
Visitor
Registered User
- Name : String
- Surname : String
- Email : String
- Password : String
SystemGPS
Taxi
- code : int
Use
call
to find
has temporarly
looks for
receive
defines
Location
- Street : String
- Number : int
- zip_code : int
- City : String
- Nation : String
CostOfJourney
- Price : int
considers
Information
- Content : String
Time
Code
1
1
1..*
1
1
0..1
1
1
1
0..1
1..*
2
1
1

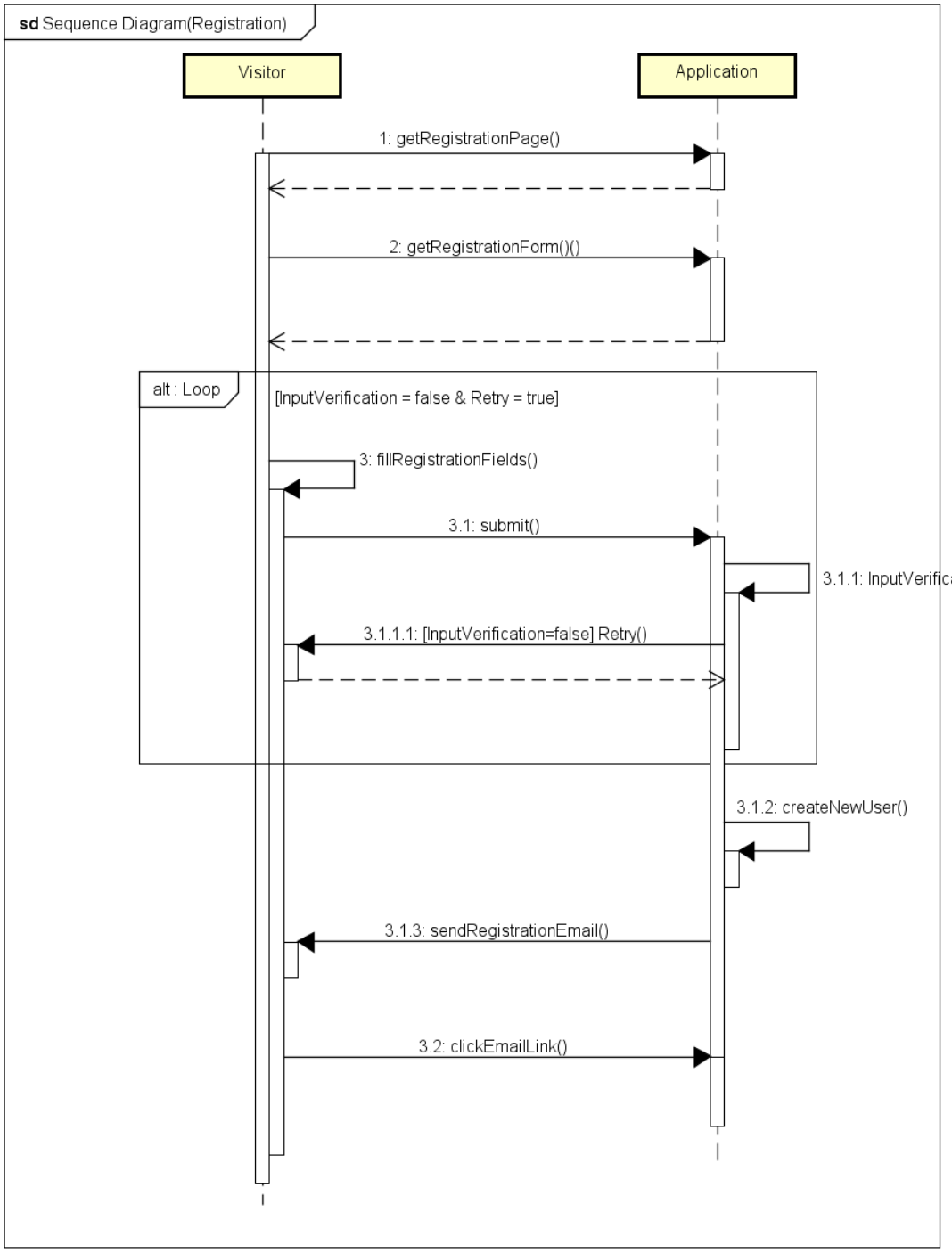
Sequence Diagram

A **Sequence diagram** is an interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called **event diagrams** or **event scenarios.**

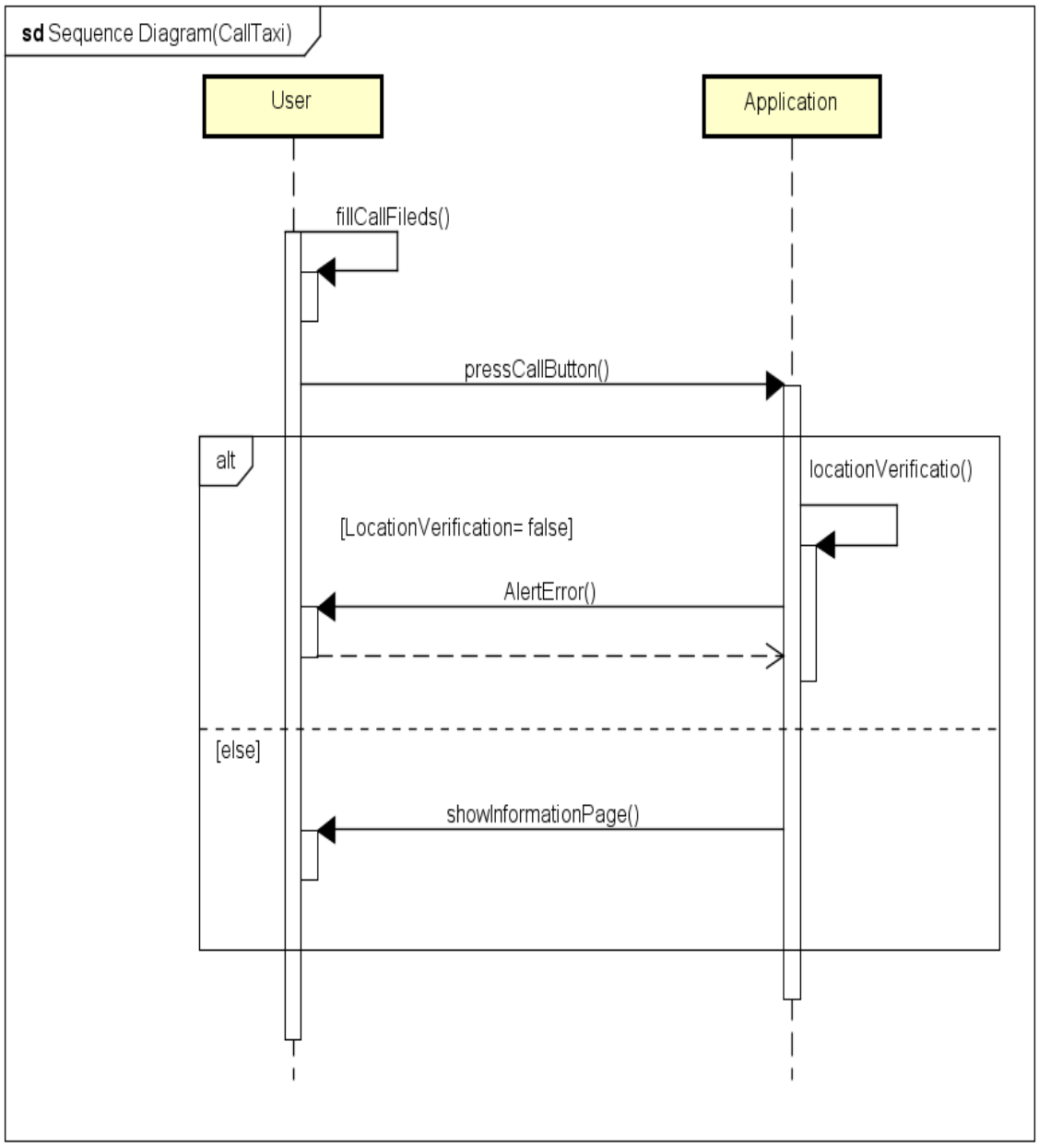
Log In


sd Sequence Diagram(Login)
RegisteredUser
Application
loop [Guard]
1: fillLoginFields()
2: sendLoginInput()
alt
[username and password correct]
2.1: Login()
[else]
2.2: LoginIncorrect()

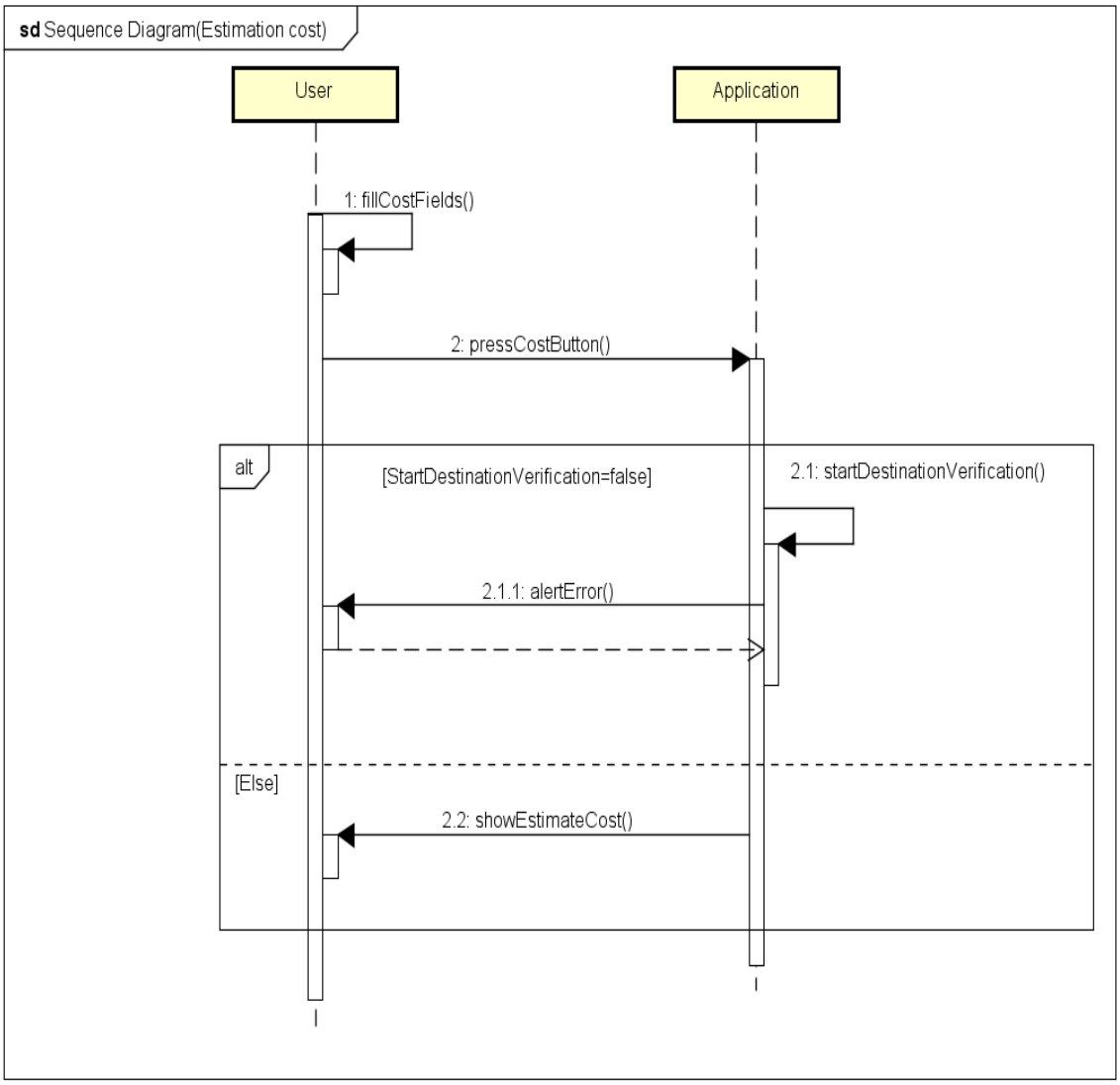
Registration



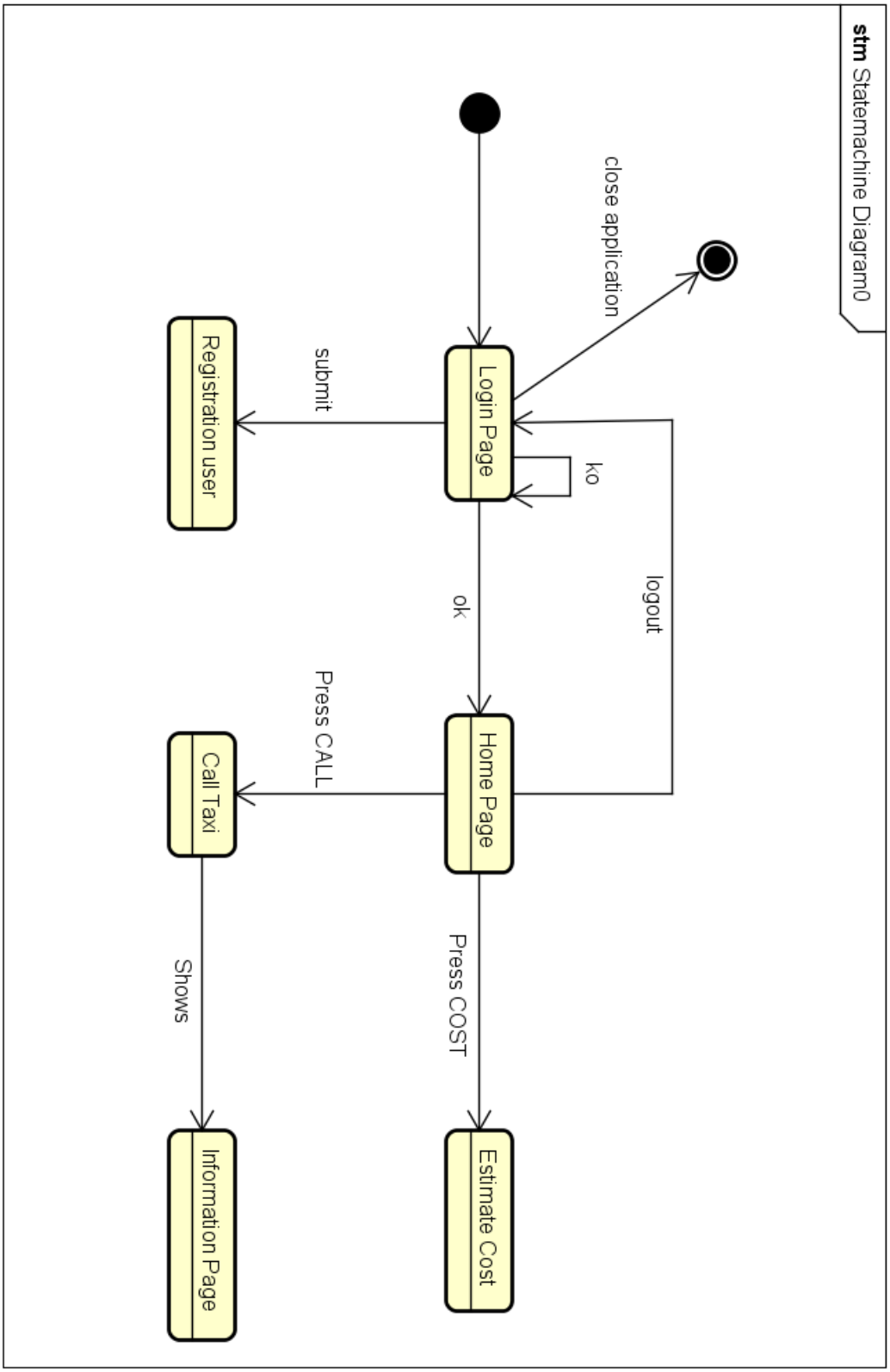
Call a Taxi


sd Sequence Diagram(CallTaxi)
User
Application
fillCallFileds()
pressCallButton()
alt
locationVerificatio()
[LocationVerification= false]
AlertError()
[else]
showInformationPage()

Estimation Cost



State Machine Diagram



6.ALLOY

Alloy is a declarative specification language for expressing complex structural constraints and behavior in a software system. Alloy provides a simple structural modeling tool based on first-order logic. Alloy is targeted at the creation of *micro-models* that can then be automatically checked for correctness. Alloy specifications can be checked using the Alloy Analyzer.

Alloy Code

```
sig Person {}

sig Visitor extends Person{}

sig RegisteredUser extends Person{
	call: set Taxi,
	receive: set Information,
	receiveCost: set JourneyCost,
	useGPS: set GpsSystem,
	location: set Location,
	name: one String,
	surname: one String,
	email: one String,
	password: one String
}

sig Location{
	street: one String,
	number: one Int,
	zipCode: one Int,
	city: one String,
	nation: one String
}

sig Taxi{
	caller: one RegisteredUser,
	code: one Int,
	send: set Information
}

sig Information{
	content: set String,
	visibleTo: set RegisteredUser
}

sig WaitingTime extends Information{}

sig TaxiCode extends Information{}

sig JourneyCost{
	price: set Int
}

sig GpsSystem{}
```

```
//FACTS

fact OneUserPerTaxi{
	//Every taxi is called by one user
	all t: Taxi | ( one u: RegisteredUser | u.call = t )
}

fact OneLocationPerUser{
	//Every user has only a position
	all u: RegisteredUser | ( one p: Location | u.location = p)

}

fact InformationProperty{
	//The information about waiting time and code are visible only to the taxi caller.
	all i: Information | ( all t: Taxi | i.visibleTo in t.caller )
}

fact noEmptyLocation{
	//no Empty Location
	all p: Location |(#p.street = 1) and ( #p.number = 1) and (# p.zipCode = 1 ) and (#p.city = 1) and ( #p.nation = 1)
}

fact noEmptyTaxiCode{
	//no empty taxi code
	all t: Taxi | ( #t.code = 1)
}

fact noEmptyPrice{
	//no empty price of the journey
	all c: JourneyCost | (#c.price = 1)
}

fact noEmptyTime{
	// no empty waiting time information
	all w: WaitingTime | (#w.content = 1)
}
```

```
//ASSERTIONS


assert noMultiCall{
	//check no a taxi is called by two or more user
	no u1, u2: RegisteredUser, t: Taxi | u1 in t.caller and u2 in t.caller
}
check noMultiCall


assert noMultiLocation{
	// check no users have two or more location
	no li: Location, lu: Location, u: RegisteredUser | ( u.location = li) and (u.location = lu)
}
check noMultiLocation

assert noViolationProperty{
	// check that the information about waiting time and code are visible only to the taxi caller.
	no i: Information | ( i.visibleTo in RegisteredUser)
}
check noViolationProperty


//PREDICATES

pred showTaxiInformation{
//Show a world in which there is a taxi that sends waiting time and code information to a registered user.
all u1: RegisteredUser, t1: Taxi, i1: WaitingTime, i2 : TaxiCode | (i1 in t1.send implies i1 in u1.receive)
 and (i2 in t1.send implies i2 in u1.receive) and (t1.caller = u1) and (u1.call = t1)
}
run showTaxiInformation

pred showEstimationCost{
//Show a world in which there is an user that use the estimation cost function with GPS sistem.
all u1: RegisteredUser, j1:JourneyCost, g1: GpsSystem | u1.useGPS = g1  and u1.receiveCost = j1
}
run showEstimationCost for 5

pred showCall{
//Show a world in which there are two user and one taxi, so only one user can call the taxi.
all t: Taxi, u1, u2 : RegisteredUser | u1 != u2 and u1 in t.caller and u2 not in t.caller
}
run showCall
```

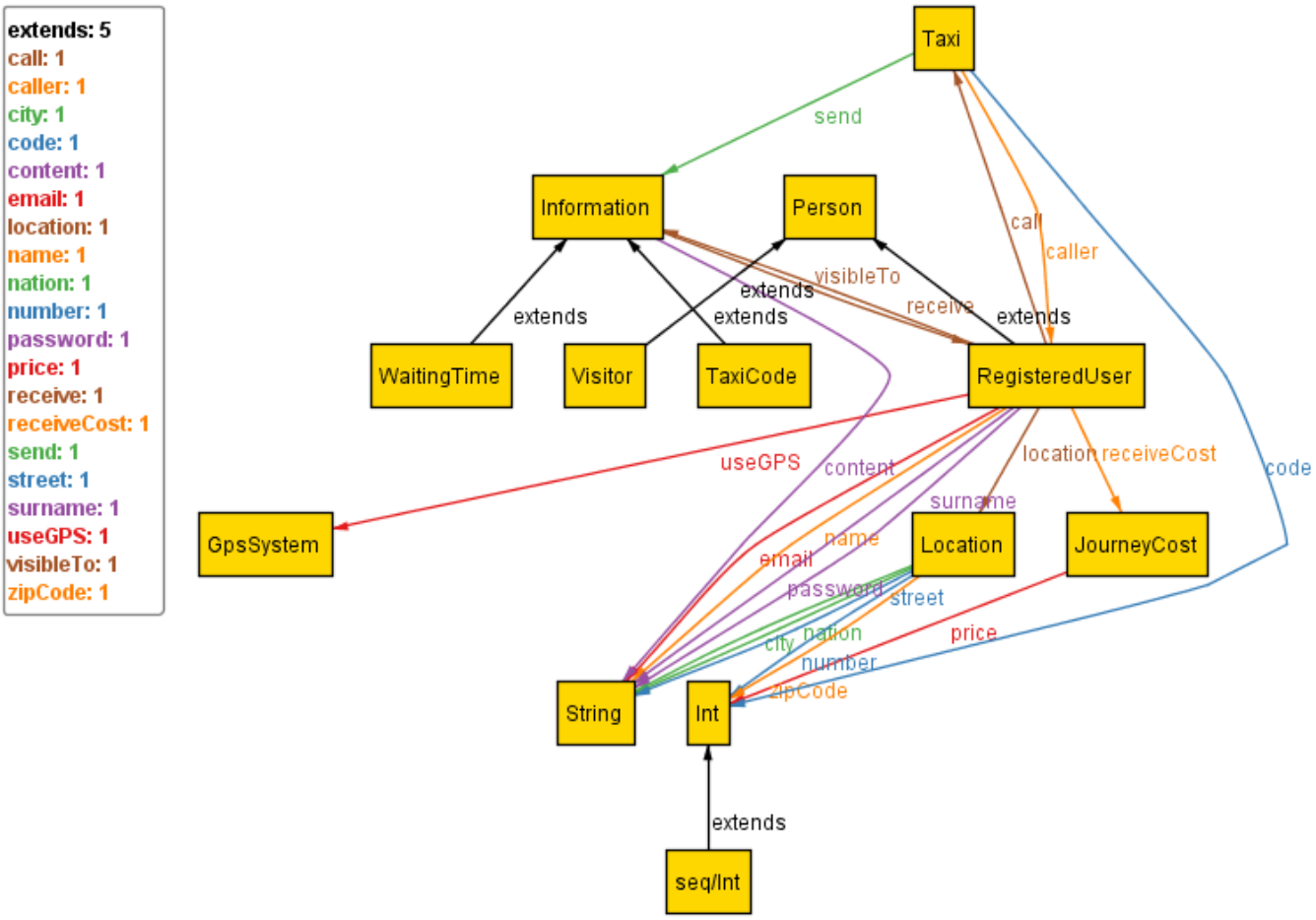
Result

The result of the analysis:

```
#1: No counterexample found. noMultiCall may be valid.
#2: No counterexample found. noMultiLocation may be valid.
#3: Counterexample found. noViolationProperty is invalid.
#4: Instance found. showTaxiInformation is consistent.
#5: Instance found. showEstimationCost is consistent.
#6: Instance found. showCall is consistent.
```

Alloy Worlds

General World:



Used tools

The tools we used to create the RASD document are:

- Microsoft Office Word 2011.
- Astah: to create UML Models.
- Alloy Analizer 4.2.

I spent 55 hours to redacting and writing this document.