

myTaxiService

Software Engineering 2 -Project

Alberto Maria Metelli Riccardo Mologni

Politecnico di Milano
M. Sc. in Computer Science and Engineering

RASD Presentation, 11th November 2015

Outline

- 1 Introduction
 - Actors
 - Goals identification
 - Product perspective
- 2 Requirement analysis
 - Functional requirements
- 3 Requirement specification
 - Use cases
 - Other UML diagrams
 - Non functional requirements
 - Alloy model

Outline

- 1 Introduction
 - Actors
 - Goals identification
 - Product perspective
- 2 Requirement analysis
 - Functional requirements
- 3 Requirement specification
 - Use cases
 - Other UML diagrams
 - Non functional requirements
 - Alloy model

Actors

- Passenger
 - Registered passenger
 - Unregistered passenger
- Taxi driver
- Call center operator



Outline

- 1 Introduction
 - Actors
 - Goals identification
 - Product perspective
- 2 Requirement analysis
 - Functional requirements
- 3 Requirement specification
 - Use cases
 - Other UML diagrams
 - Non functional requirements
 - Alloy model

Goals

Jackson Zave approach - 1



- [G1] Allow a passenger to request a taxi for its current position without registration.
- [G2] Allow the passenger to visualize the waiting time and the code of the incoming taxi for confirmed requests.
- [G3] Allow a registered passenger to have a personal area.
- [G4] Allow a registered passenger to reserve a taxi.

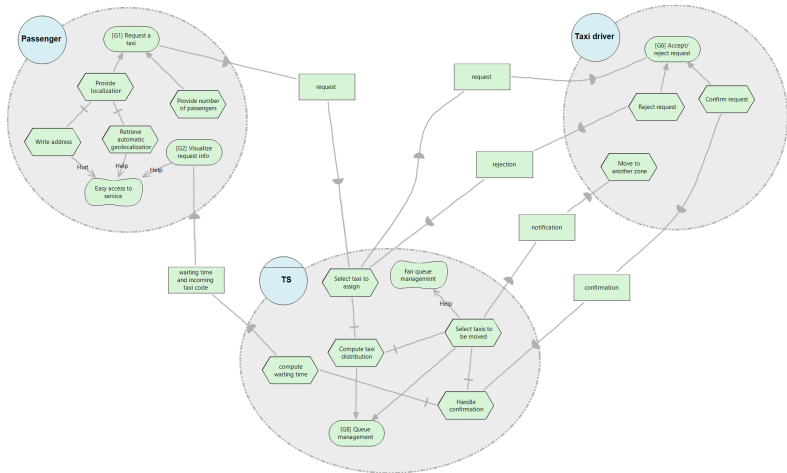
Goals

Jackson Zave approach - 2

- [G5] Allow a registered passenger to cancel or modify a previous reservation.
- [G6] Allow a taxi driver to either accept or reject a request coming from the system.
- [G7] Allow a taxi driver to inform the system about his/her availability.
- [G8] Ensure that available taxi queues enjoy “fair properties” specified in sub paragraph 1.6.2.



i* model

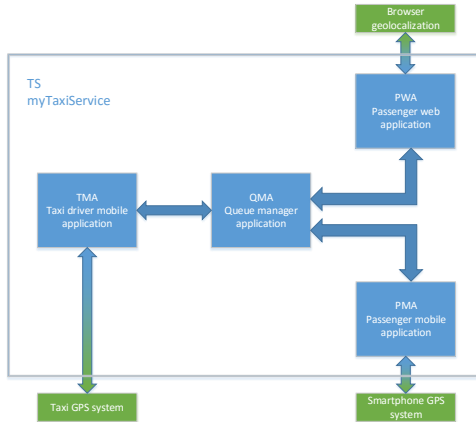


Outline

- 1 Introduction
 - Actors
 - Goals identification
 - Product perspective
- 2 Requirement analysis
 - Functional requirements
- 3 Requirement specification
 - Use cases
 - Other UML diagrams
 - Non functional requirements
 - Alloy model

Product perspective

Conceptual system decomposition



Product perspective

Interoperability

- Integration with the previous taxi management system based on phone calls
- APIs to allow programmers to access the main functionalities for future requirements extensions




Outline

- 1 Introduction
 - Actors
 - Goals identification
 - Product perspective
- 2 Requirement analysis
 - Functional requirements
- 3 Requirement specification
 - Use cases
 - Other UML diagrams
 - Non functional requirements
 - Alloy model

Functional requirements

Request - 1



- **[G1] Allow a passenger to request a taxi for its current position without registration.**
- [R1.1] TS shall provide the passenger with a form in which he/she has to insert the total number of passengers and accept terms and conditions 
- [R1.2] TS shall retrieve automatically passenger's position if GPS or browser geolocalization is available, otherwise user has to specify his address.

Functional requirements

Request - 2

- **[G1] Allow a passenger to request a taxi for its current position without registration.**
- **[R1.3] After confirmation, TS shall store the request and**
 - [R.1.3.1] Assign it to the first available taxi in the queue of the zone.
 - [R.1.3.2] If the queue is empty, TS shall look for taxis in the queues of adjacent zones and, if necessary, repeat the process for the other adjacent zones.
 - [R.1.3.3] If no taxi is found, TS shall inform passenger and put request on hold.



Functional requirements



Reservation

- **[G4] Allow a registered passenger to reserve a taxi.**
- [R4.1] TS shall provide the registered passenger with a form in which he/she has to insert the total number of passengers, the origin and the destination of the ride, the date and time of the meeting.
- [R4.2] TS shall accept only reservations made at least two hours in advance.
- [R4.3] TS shall store the reservation, allocate a request 10 minutes before the meeting time.



Functional requirements

Request evaluation

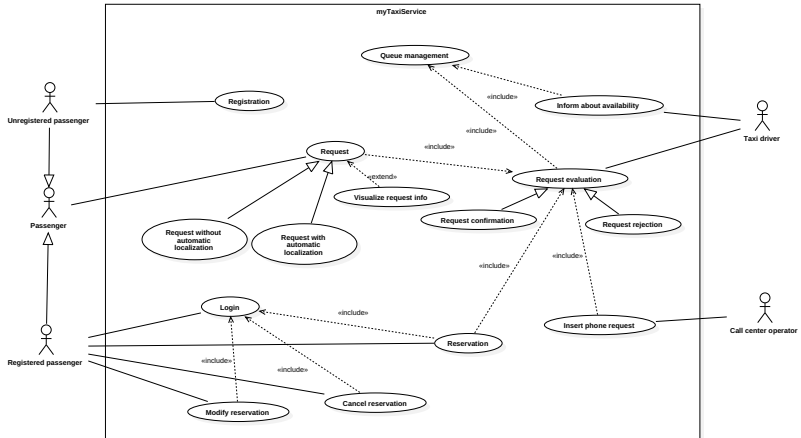
- **[G6] Allow a taxi driver to either accept or reject a request coming from the system.**
- [R6.1] TS shall show to the chosen taxi driver the request indicating coordinates of the passenger and total number of passengers.
- [R6.2] TS shall provide the taxi driver with a form allowing him to choose if accept or reject the request.
- [R6.3] TS prevents taxi driver to reject twice the same request 
- [R6.4] In case of acceptance, TS shall put the taxi driver into state *busy*, otherwise put taxi driver at the end of the queue and repeat [R1.3]. If no answer from the taxi driver in one minute it is interpreted as a rejection. 

Outline

- 1 Introduction
 - Actors
 - Goals identification
 - Product perspective
- 2 Requirement analysis
 - Functional requirements
- 3 **Requirement specification**
 - **Use cases**
 - Other UML diagrams
 - Non functional requirements
 - Alloy model

Use cases

UML Use Case diagram



Reservation

Use case description - 1

<i>Name</i>	Reservation
<i>Related goals</i>	[G4]
<i>Actors</i>	Registered passenger
<i>Entry condition</i>	Passenger is logged in.
<i>Flow of events</i>	<ol style="list-style-type: none"> ➊ Passenger accesses to the reservation area. ➋ Passenger inserts the required data (origin, destination, date, time, number of passengers). ➌ Passenger confirms the reservation. ➍ TS system whether data are valid. ➎ QMA creates a new reservation and the related request is allocated.

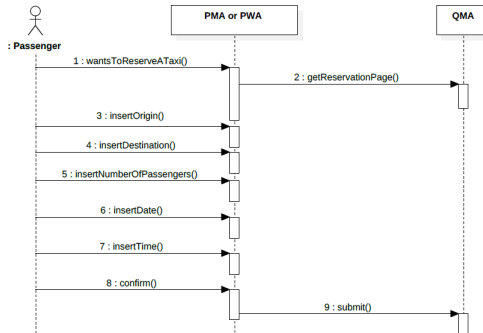
Reservation

Use case description - 2

<i>Exit condition</i>	The reservation is added to the TS system.
<i>Exceptions</i>	<ul style="list-style-type: none">● If passenger does not confirm the operation is not performed.● If the data are not valid (origin, destination, number of passengers) an error message is shown to passenger and the operation is not performed. Passenger can repeat the process.● If the date and time are such that the reservation is not made at least two hour in advance an error message is shown to user and the operation is not performed. Passenger can repeat the process.

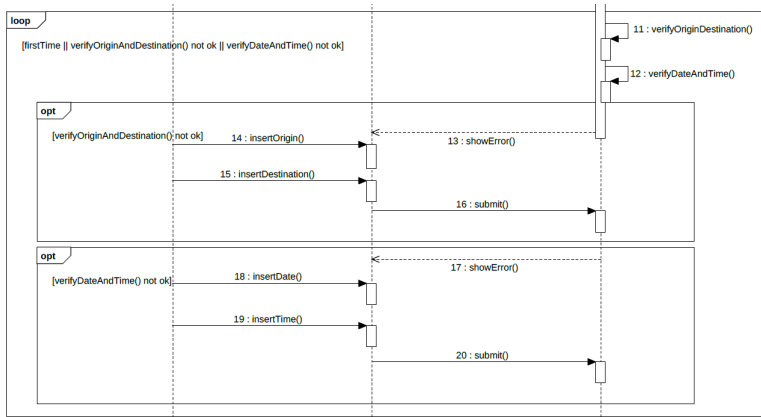
Reservation

Sequence diagram - 1



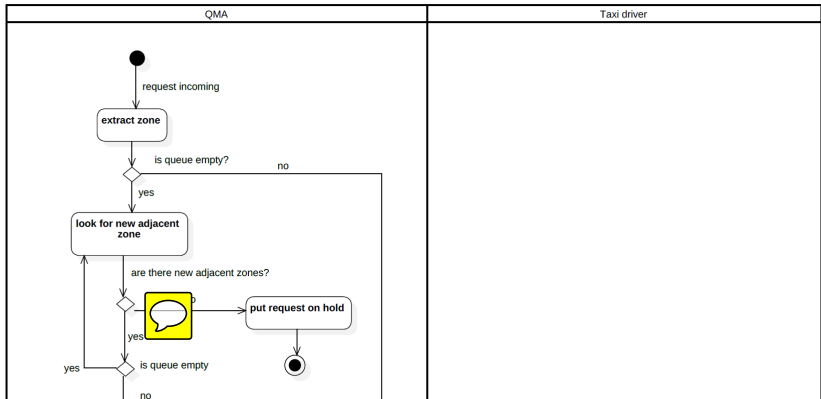
Reservation

Sequence diagram - 2



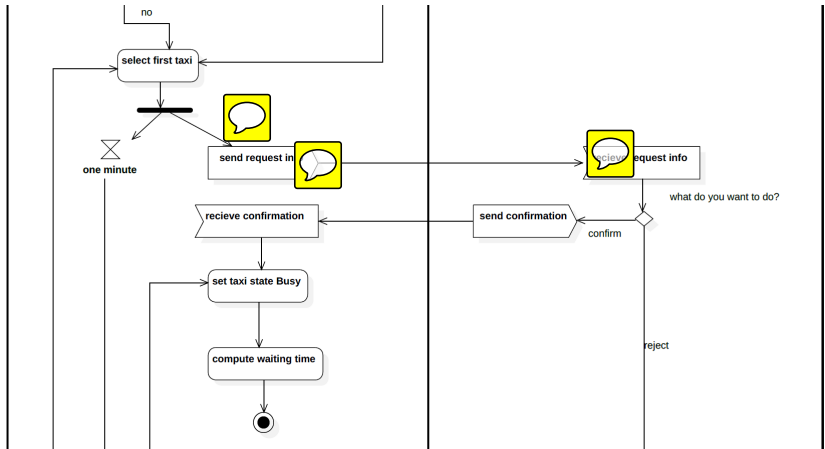
Request evaluation

Activity diagram - 1



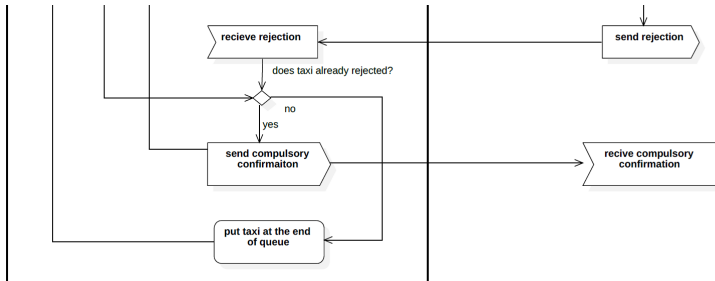
Request evaluation

Activity diagram - 2



Request evaluation

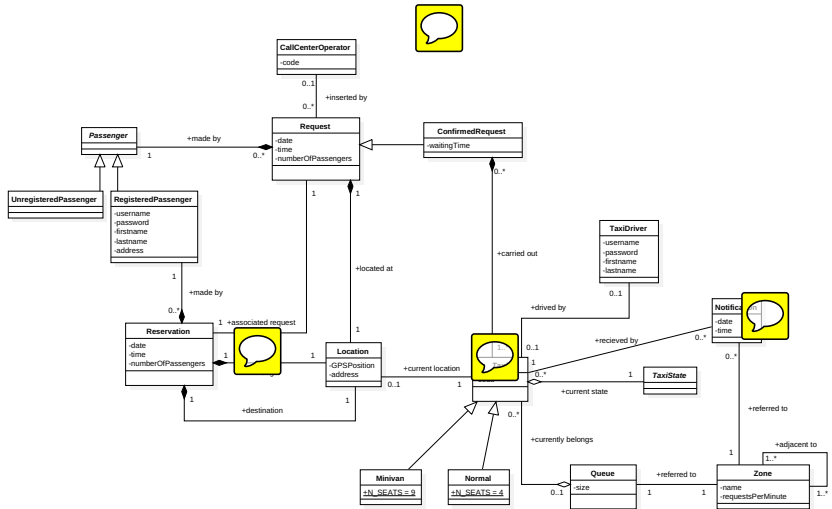
Activity diagram - 3



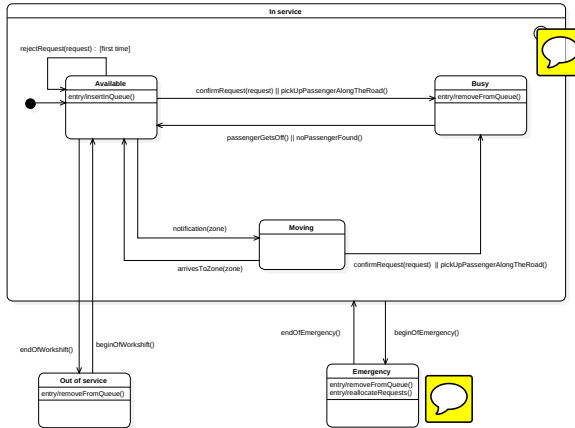
Outline

- 1 Introduction
 - Actors
 - Goals identification
 - Product perspective
- 2 Requirement analysis
 - Functional requirements
- 3 **Requirement specification**
 - Use cases
 - **Other UML diagrams**
 - Non functional requirements
 - Alloy model

Class diagram



State Chart



Outline

- 1 Introduction
 - Actors
 - Goals identification
 - Product perspective
- 2 Requirement analysis
 - Functional requirements
- 3 **Requirement specification**
 - Use cases
 - Other UML diagrams
 - **Non functional requirements**
 - Alloy model

Non functional requirements

- Performance
- Reliability
- Availability
- Security
- Maintainability
- Portability
- Documentation
- User interface and human factors

Outline

- 1 Introduction
 - Actors
 - Goals identification
 - Product perspective
- 2 Requirement analysis
 - Functional requirements
- 3 **Requirement specification**
 - Use cases
 - Other UML diagrams
 - Non functional requirements
 - **Alloy model**



Alloy model

Some signatures - 1



```

1  abstract sig Passenger{}
   sig UnregisteredPassenger extends Passenger{}
3  sig RegisteredPassenger extends Passenger{}

5  abstract sig TaxiState{}
   one sig OutOfService, Emergency extends TaxiState{}
7  one sig Available, Busy, Moving extends TaxiState{}

9  abstract sig Taxi {
    driver: lone TaxiDriver,
11     state: one TaxiState,
    numberOfSeats: one Int,
13 }

15 sig MinivanTaxi extends Taxi {} {numberOfSeats = 9}
   sig NormalTaxi extends Taxi {} {numberOfSeats = 4}

```


Alloy model

Some signatures - 2

```
sig Request{
2     passenger: one Passenger ,
        date: one Date ,
4     time: one Time ,
        numberOfPassengers: one Int ,
6     location: one Location ,
} {numberOfPassengers >=1}
8
sig ConfirmedRequest extends Request{
10     waitingTime: one Time ,
        taxis: some Taxi ,
12 }
```

Alloy model

Some facts - 1

```
//Origin and destination for each request must be
different
2 fact originAndDestinationDifferent {
    all r: Reservation | r.origin != r.destination
4 }

//In each request the number of seats must be
sufficient wrt number of passengers
2 fact numberOfSeatsSufficient {
    all r: ConfirmedRequest | sum r.taxis.
        numberOfSeats
4     >= r.numberOfPassengers
}
```

Alloy model

Some facts - 1

```
1 //Origin and destination for each request must be
   different
   fact originAndDestinationDifferent {
3       all r: Reservation | r.origin != r.destination
   }

   //In each request the number of seats must be
   sufficient wrt number of passengers
2 fact numberOfSeatsSufficient {
       all r: ConfirmedRequest | sum r.taxis.
       numberOfSeats
4       >= r.numberOfPassengers
   }
```

Alloy model

Some facts - 2

```
1 //The number of taxis sent are the minimum requested
   to pick up all passengers
fact numberOfSeatsAreTheMinimumRequired {
3   all r: ConfirmedRequest | no taxiSubset: set
      Taxi | taxiSubset in r.taxis and
      taxiSubset != r.taxis and sum taxiSubset.
      numberOfSeats >= r.numberOfPassengers
}

fact numberOfSeatsAreTheMinimumRequired {
2   all r: ConfirmedRequest | no t: Taxi | t in r.
      taxis and sum r.taxis.numberOfSeats - t.
      numberOfSeats >= r.numberOfPassengers
}
```

Alloy model

Some facts - 2

```
1 //The number of taxis sent are the minimum requested
   to pick up all passengers
fact numberOfSeatsAreTheMinimumRequired {
3     all r: ConfirmedRequest | no taxiSubset: set
        Taxi | taxiSubset in r.taxis and
        taxiSubset != r.taxis and sum taxiSubset.
        numberOfSeats >= r.numberOfPassengers
}

fact numberOfSeatsAreTheMinimumRequired {
2     all r: ConfirmedRequest | no t: Taxi | t in r.
        taxis and sum r.taxis.numberOfSeats - t.
        numberOfSeats >= r.numberOfPassengers
}
```

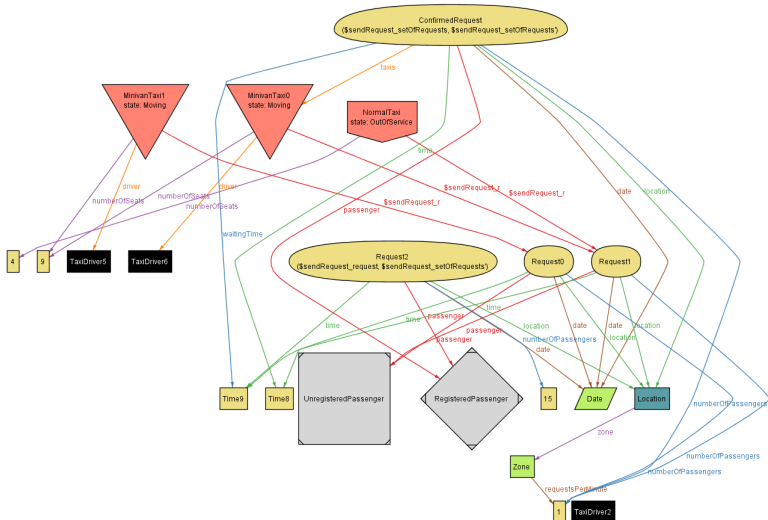
Alloy model

Predicates

```
1 //Predicate for sending a request: if the request is
   not already confirmed and it is not already in the
   set it is added to the set
   pred sendRequest[setOfRequests, setOfRequests': set
     Request, request: Request] {
3       no ((ConfirmedRequest + setOfRequests) &
         request) implies
           setOfRequests' = setOfRequests +
             request
5       else
           setOfRequests' = setOfRequests
7     }
```

Alloy model

World generated by pred sendRequest



Alloy model

Assertions

```

1 //Request carried out by the same taxi driver must be
   different in date or time
   assert allRequestOfTheSameTaxiDriverDifferentInTime {
3       all td: TaxiDriver | all disj r1,r2:
           ConfirmedRequest | td in (r1.taxis.driver
           & r2.taxis.driver) implies not
           atTheSameTime[r1,r2]
   }

```

Executing "Check allRequestOfTheSameTaxiDriverDifferentInTime for 10"
 Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
 50129 vars. 2338 primary vars. 125356 clauses. 109ms. No
 counterexample found. Assertion may be valid. 219ms.

Alloy model

Queue modeling - 1

```
sig Zone{
2       queue: one Queue,
       adjacentZones: some Zone,
4  }

6  //Queue definition
sig Queue {
8       root: lone Node
  }

10
sig Node {
12       taxi: one Taxi,
       next: lone Node
14 }
```

Alloy model

Queue modeling - 2

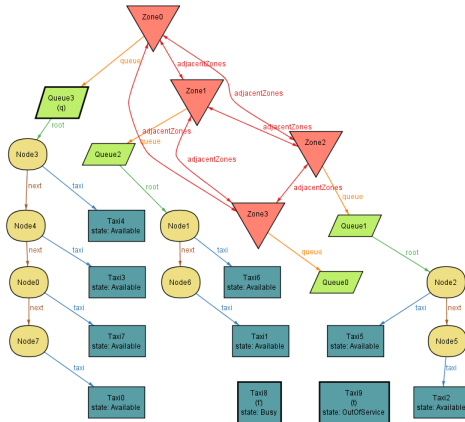
```
//Structural properties
2 fact queueStructuralProperties {

4     //Each node belongs to exactly one queue
    all n: Node | one q: Queue | n in q.root.*next
6     //No cycles
    no n: Node | n in n.^next
8 }

10 //Adjacency relation between zones is simmetric but
    not reflexive
fact adjacencySimmetricButNotReflexive {
12     adjacentZones in ~adjacentZones
    no adjacentZones & iden
14 }
```

Alloy model

World generated



References

- IEEE Software Engineering Standards Committee, “IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications”, October 20, 1998.
- P, Zave, M. Jackson, Four dark corners of requirements engineering, TOSEM 1997.
- Software Abstractions: Logic, Language, and Analysis, revised edition Edition by Daniel Jackson, MIT Press.
- Software Engineering 2 course slides.

Questions

