myTaxiService

Nanni Luca
Giacomo Servadei

Important remark

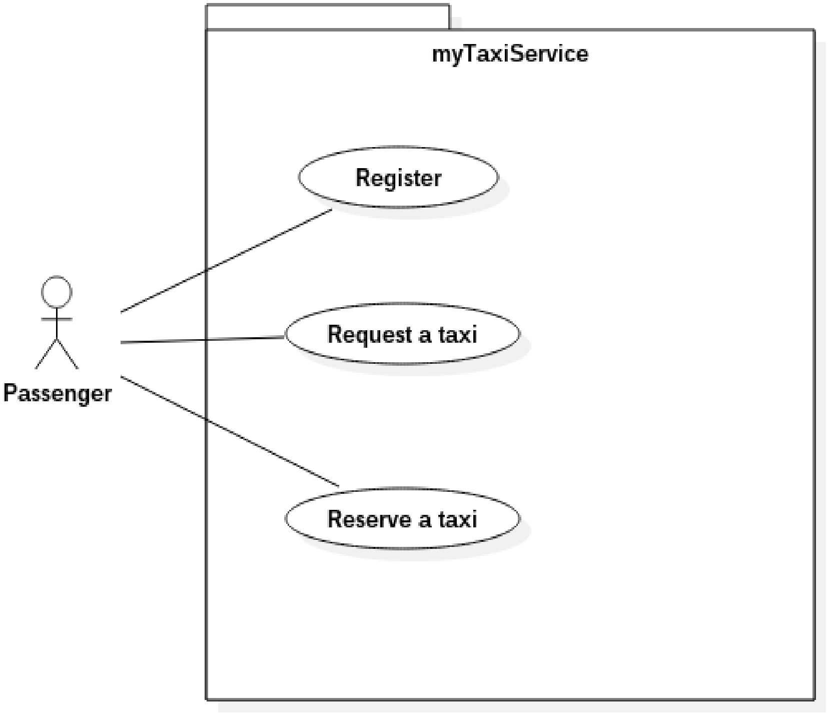
In this presentation we will cover and show only the **main** features of our project.
Everything is explained with **more detail in the documentation**.

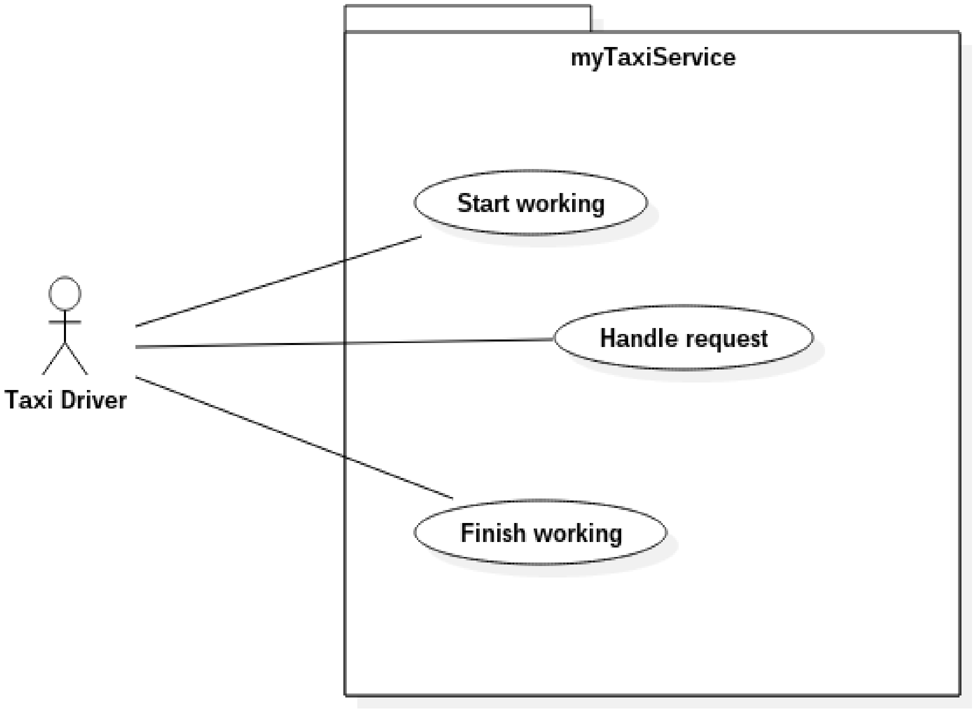
We have not included the *Code Inspection* assignment in this presentation. Its documentation is available in the GitHub repository

Outline of the project

1. **Requirement analysis and specification**
2. Design
3. Integration testing definition
4. Project planning

Requirements analysis





Assumptions

Passenger side:

- Once a passenger made a request and it has been accepted by a taxi, he’s going to wait for it until it arrives and he will take the ride
- There is no support for an in-app payment system
- **Passengers, once submitted a request, will never try to cancel it. ☹**

Assumptions

<u>Taxi Driver side:</u>

- The taxi drivers will set their availability only when they can accept new passengers
- There is no support for taxi driver registration in the system.
- There is absolutely no overlapping between taxi zones
- **Each taxi driver has only one taxi: this means that each taxi driver has a fixed identification number for his taxi**

Functional requirements

Passenger side:

- Login/logout/registration (username, password,ecc...) and error notification
- If a passenger makes a request and the corresponding taxi zone is not empty, then the system must respond *positively* to him, soon or later
- A passenger request must be refused if and only if there are no taxi driver available in the corresponding taxi queue or the number of passengers of the ride is greater than 3 (*)

Functional requirements

(*) We notice immediately that if all the taxi drivers in a queue refuse to accept the Passenger request, this will be forwarded again to the first taxi driver in the queue, who is the first one that refused it.

We think that this behavior is fair but can cause a loop and the Passenger could wait for a response a lot of time.

Functional requirements

Passenger side:

- A reservation must be refused if and only if:
    - Origin and destination are the same location
    - The number of passengers of the ride is greater than 3
    - time(meeting time) - time(reservation) < 2 hours
- When the system looks for a taxi driver for serving a reservation (10 minutes before the meeting time), if the corresponding taxi queue is empty, it will wait until there is at least one taxi driver in the queue.

Functional requirements

Taxi driver side:

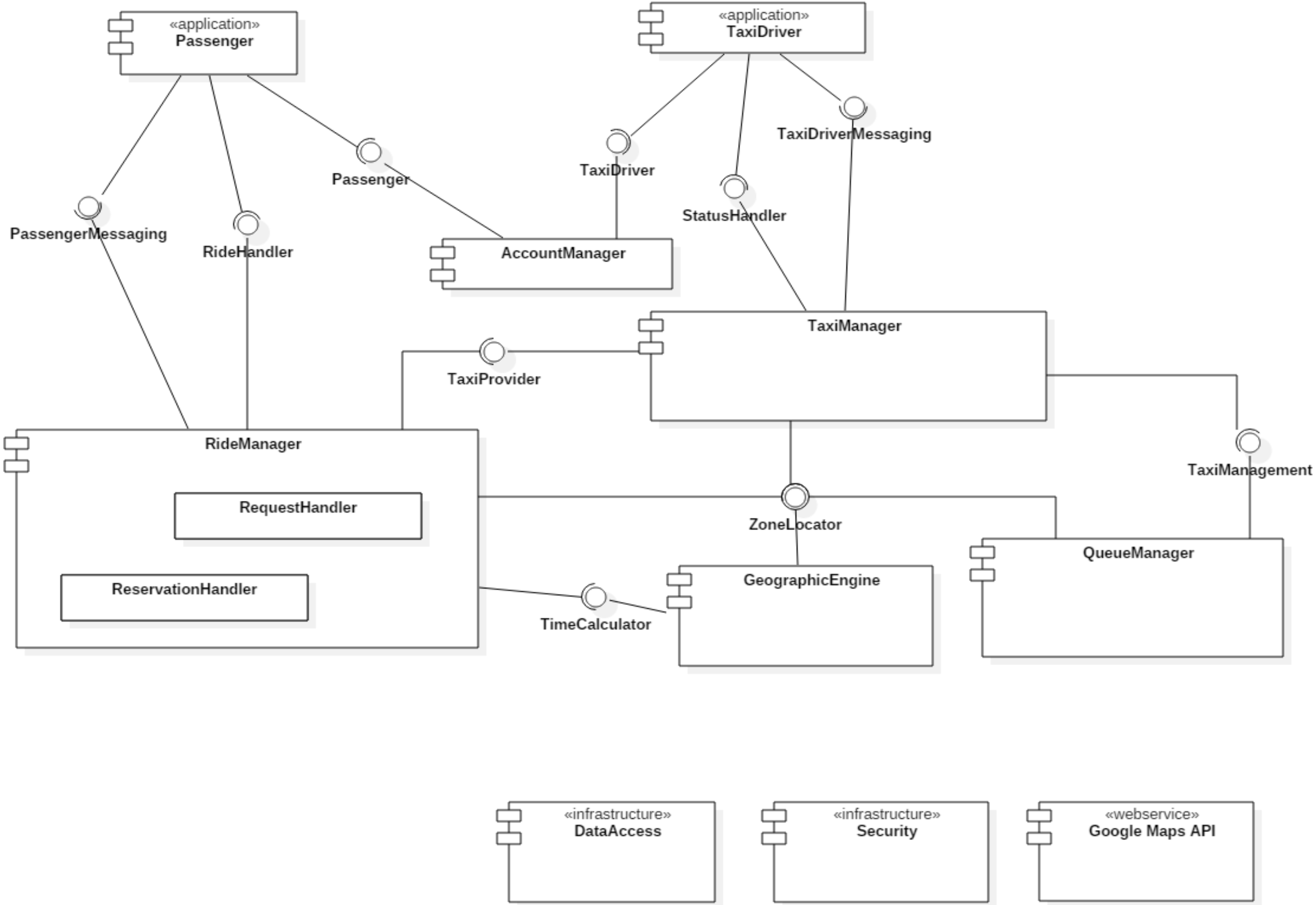
- Each queue can be empty or have a finite positive number of taxis
- The system must put the taxi driver at the bottom of the queue if:
  - He refuses a Request
  - He does not respond to a request within the 10 seconds from the reception of it
- Can receive requests only if he is at the top of the taxi queue
- For each location of the taxi driver must correspond exactly one taxi zone
- When a Taxi driver accepts a request from a Passenger, he must be removed from the corresponding taxi queue and set as busy

Outline of the project

1. Requirement analysis and specification
2. **Design**
3. Integration testing definition
4. Project planning

Design

We have already done this presentation, but in order to better understand the following parts, we will do a very short recap of our design choices!



«application»
Passenger
«application»
TaxiDriver
TaxiDriverMessaging
TaxiDriver
Passenger
StatusHandler
PassengerMessaging
RideHandler
AccountManager
TaxiManager
TaxiProvider
RideManager
TaxiManagement
RequestHandler
ZoneLocator
QueueManager
GeographicEngine
ReservationHandler
TimeCalculator
«infrastructure»
DataAccess
«infrastructure»
Security
«webservice»
Google Maps API

Components

Account Manager

- Account related actions (login, logout, registration, deletion)

Ride Manager

- Communication with passengers
- Lifecycle of Requests and Reservations

Taxi Manager

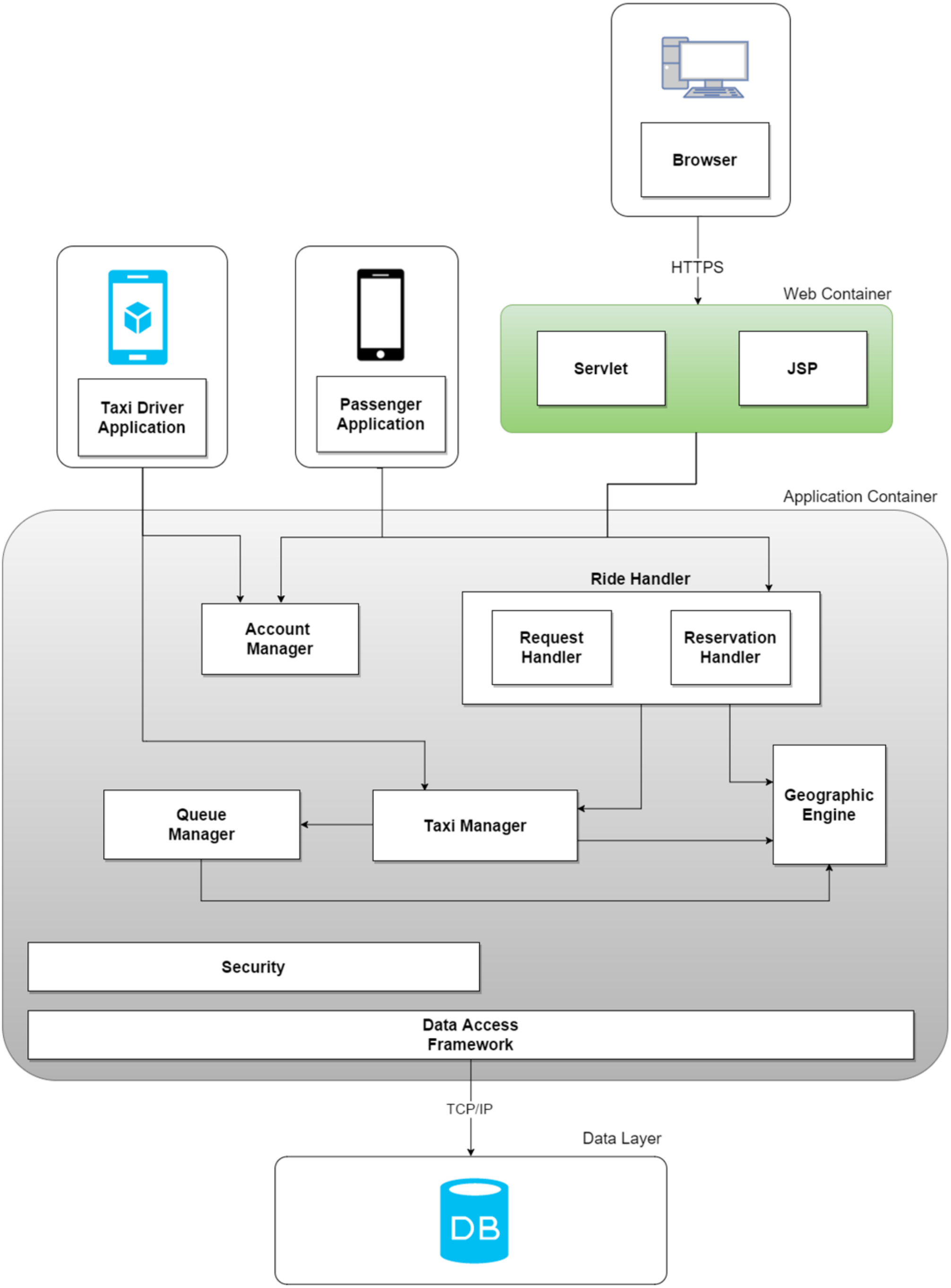
- Communication with taxi drivers and their shifts

Geographic Engine

- Geographic related actions (compute zone by location, compute waiting time)

Queue Manager

- Management of available taxi drivers (store and retrieve them by certain policy)


Browser
HTTPS
Web Container
Servlet
JSP
Taxi Driver Application
Passenger Application
Application Container
Ride Handler
Account Manager
Request Handler
Reservation Handler
Geographic Engine
Queue Manager
Taxi Manager
Security
Data Access Framework
TCP/IP
Data Layer
DB

Outline of the project

1. Requirement analysis and specification
2. Design
3. **Integration testing definition**
4. Project planning

Strategy decision

- Top Down

  No Hierarchy

- Sandwich

  No Hierarchy from the top

- Thread

  Not worth dividing all the components

- Critical Modules

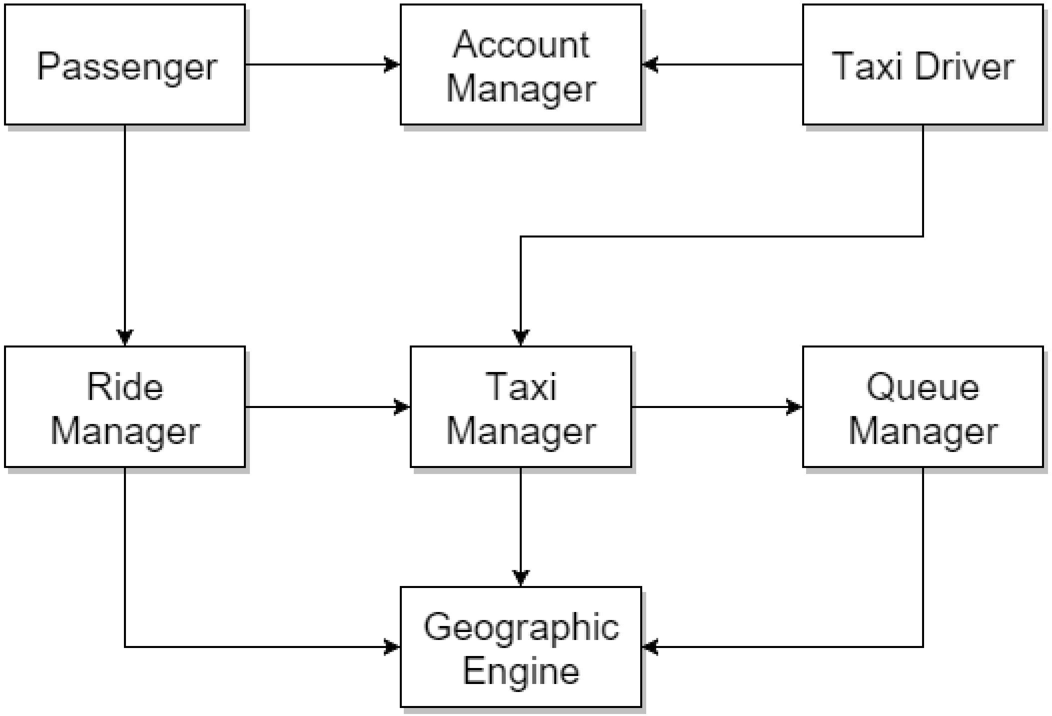
  Not complex enough

- **Bottom Up**

  The choosen one

  Integrate single components in an iterative way.

Components

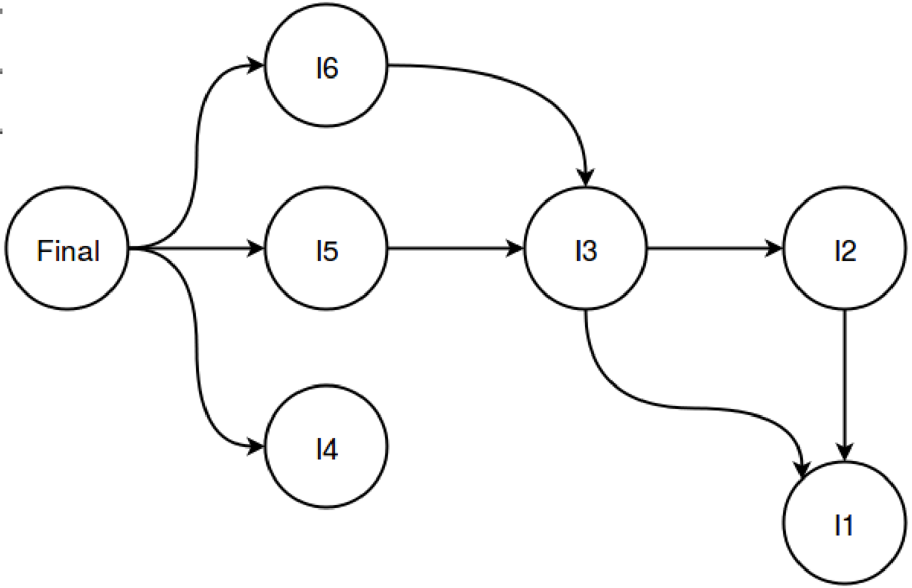

Passenger
Account Manager
Taxi Driver
Ride Manager
Taxi Manager
Queue Manager
Geographic Engine

Integration Plan

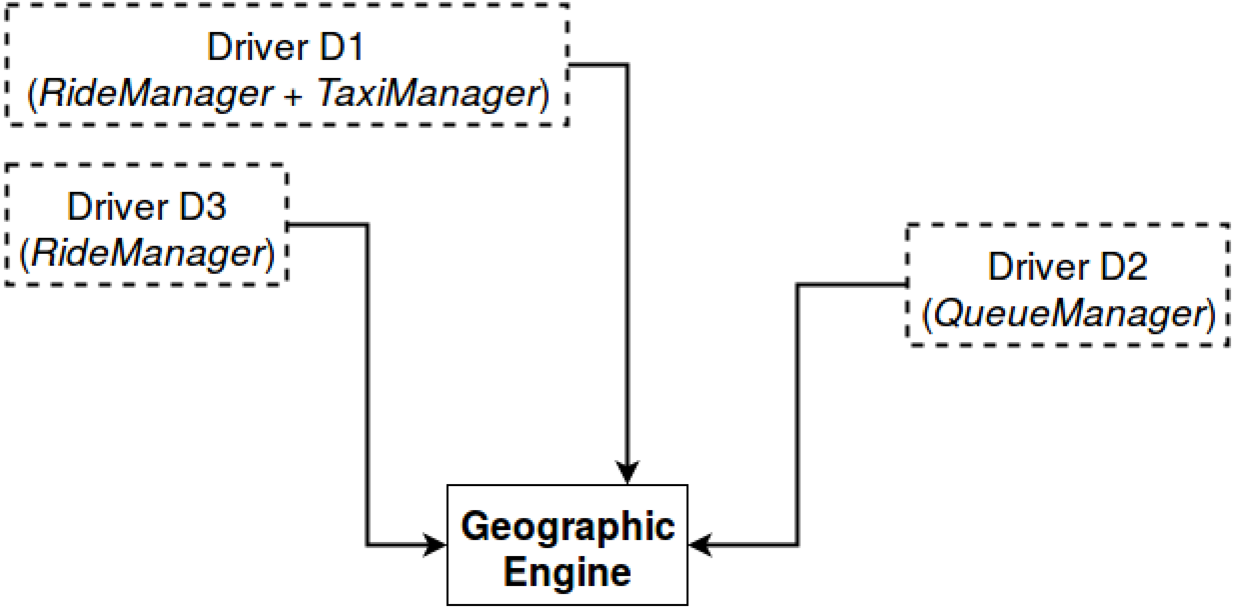
ID	Components interaction
I1	RideManager, TaxiManager, QueueManager $\Longrightarrow$ GeographicEngine
I2	TaxiManager $\Longrightarrow$ QueueManager
I3	RideManager $\Longrightarrow$ TaxiManager
I4	Passenger, TaxiDriver $\Longrightarrow$ AccountManager
I5	Passenger $\Longrightarrow$ RideManager
I6	TaxiDriver $\Longrightarrow$ TaxiManager

Dependency graph between the integration steps →

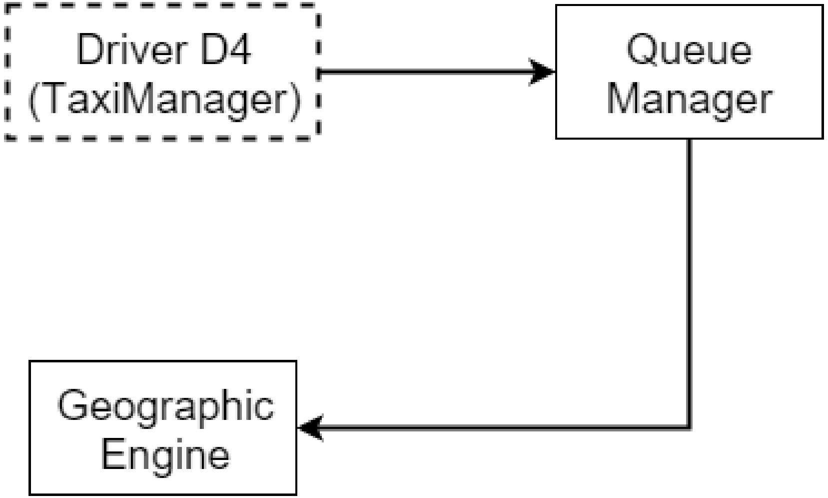
there is an arc A → B if A *needs* the previous execution of B *in order to* execute.



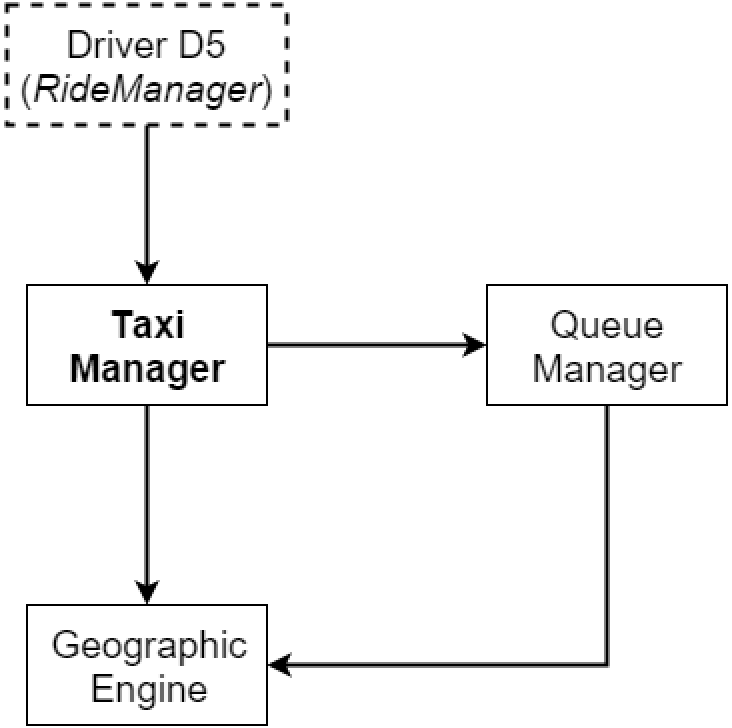
Integration I1



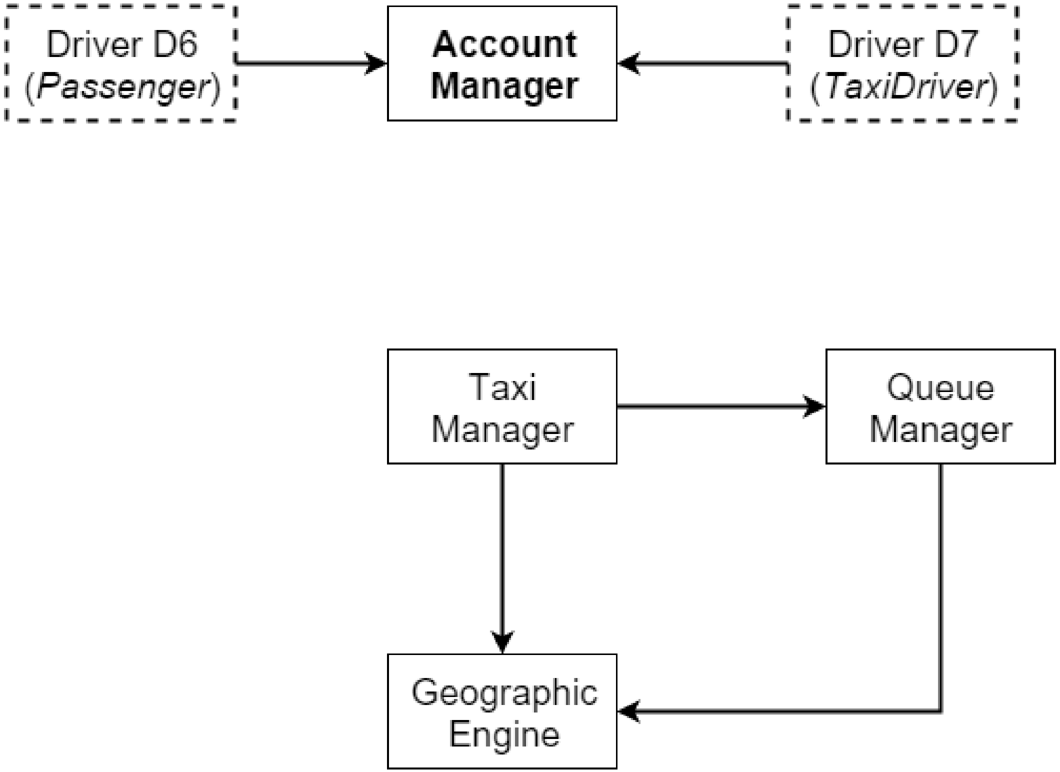
Integration I2



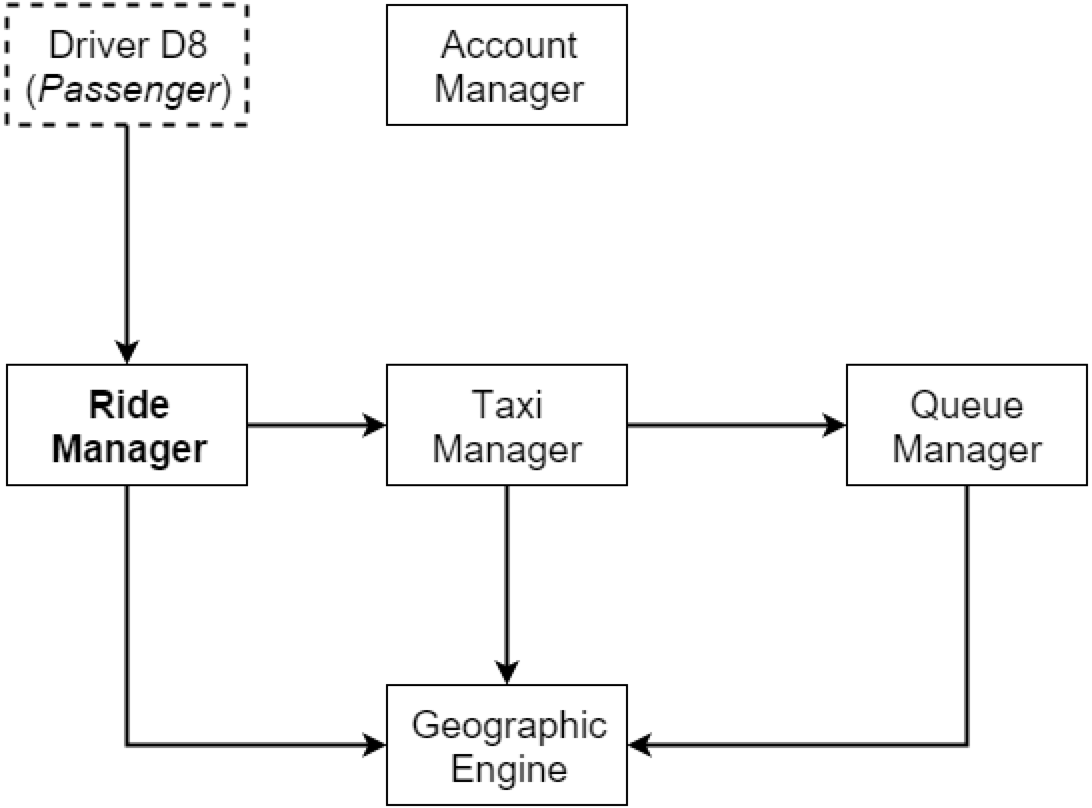
Integration I3



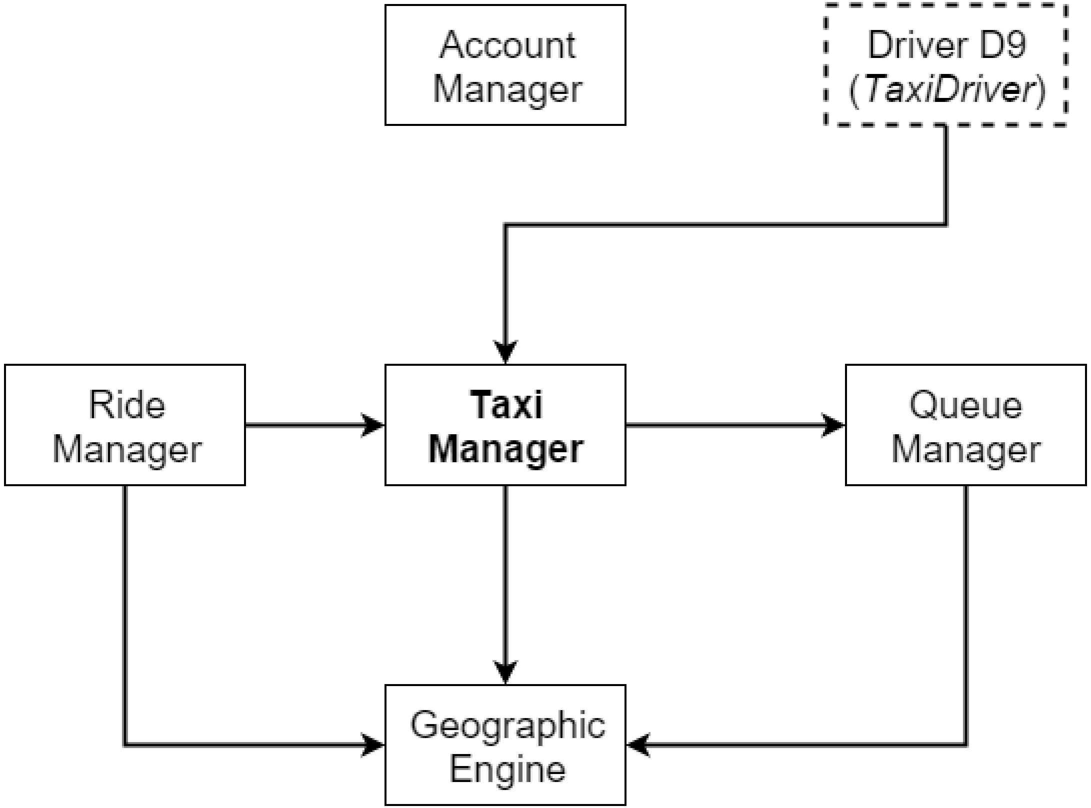
Integration I4



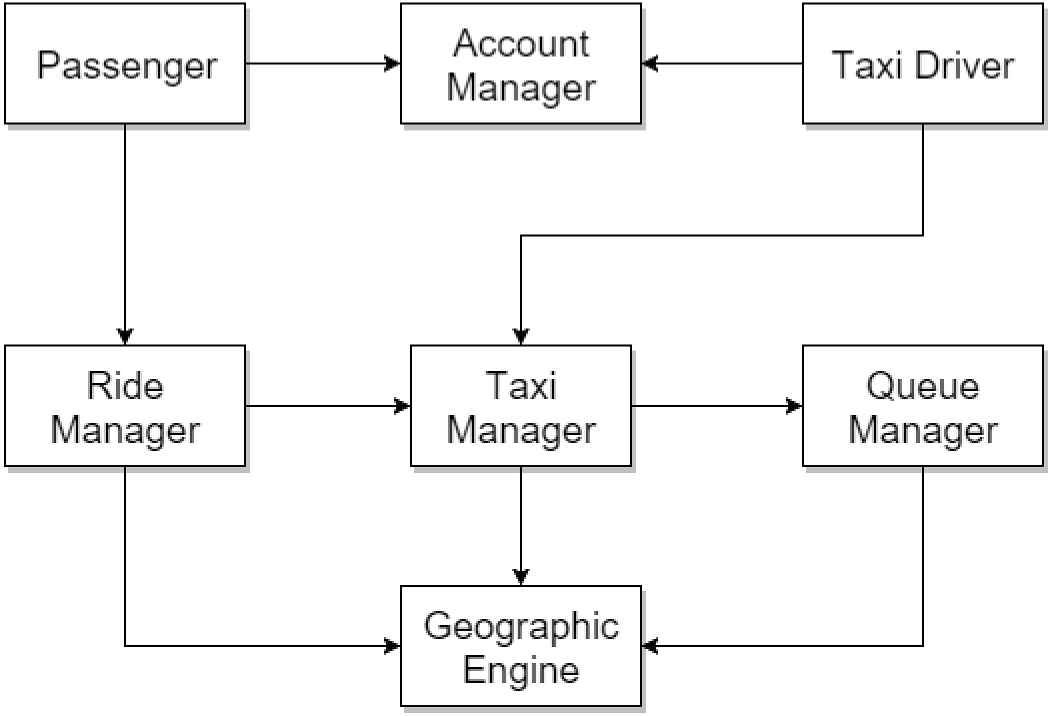
Integration I5



Integration I6



Final integration - *full test*



Scheduling of integrations

1. I1 and I4 in parallel
2. I2
3. I3
4. I5 and I6 in parallel
5. Final integration

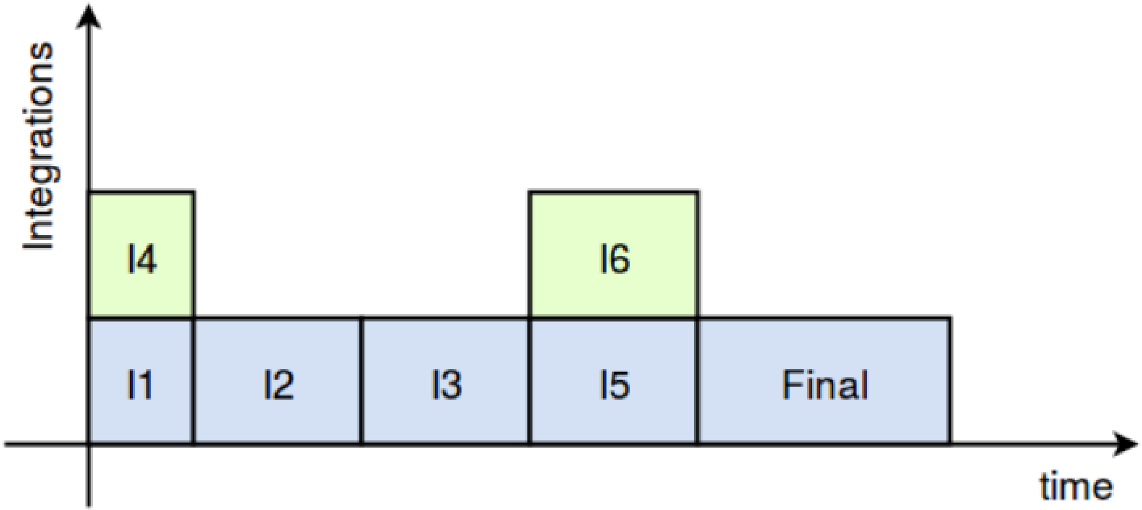


Figure 3.9: Schedule of a possible integration plan

Outline of the project

1. Requirement analysis and specification
2. Design
3. Integration testing definition
4. **Project planning**

Project Planning

Outline:

1. Function points calculation
2. COCOMO II (effort calculation → duration + dimensioning of the team)
3. Task identification and scheduling
4. Resource allocation
5. Risk analysis

1.Function points calculation

ILF	**Complexity**	**Rationale**
Requests	Simple	The requests of a passenger are to be stored, and they present a very simple structure (name of passenger, date of execution, location, ecc...)
Reservations	Simple	Similar to requests with some more attributes
Passenger accounts	Medium	Managing of all the data of a passenger account
TaxiDriver accounts	Simple	The accounts of taxi drivers are much simpler than the one of passengers. They consist simply in a username and password and the id of the taxi.
Taxi zones	Complex	The representation of taxi zone implies some geometry a probably the files containing them will be pretty complex
Taxi Queue	Medium	A taxi queue is composed by a taxi zone and the set of available taxi drivers that are in it.
Total FP	**56**	

1.Function points calculation

EIF	Complexity	Rationale
GoogleMaps API	Medium	The system uses the external service offered by Google in order to parse the locations and to transform them into geographic/geometric objects. This task is estimated not to be trivial
Total FP	**7**	

1.Function points calculation

External Input	Complexity	Rationale
Login of passenger	Simple	From the requirements
Logout of passenger	Simple	” ”
Registration of a passenger	Simple	” ”
Deletion of passenger account	Medium	Differently from the other basic operations on the account of a passenger, this one involves the elimination of all the requests and reservations.
Login of taxidriver	Simple	From the requirements
Logout of taxidriver	Simple	” ”
Make a request	Medium	Not an easy operation: it involves the storing of the request, the validation of the input and the parsing of the locations, ecc...
Make a reservation	Complex	This operation involves a lot of components in our system and probably is the most demanding one.
Setting of availability of taxi driver	Medium	It involves the setting of the zone and the pop/push into a taxi queue
Accepting/Refusing a ride	Medium	It involves the re-forwarding of the request/reservation and the notification to the passenger in case of problems.
Total FP	**37**	

1.Function points calculation

External Output	Complexity	Rationale
Sending of a request of a ride to a taxi driver	Medium	We need to retrieve the correct taxi driver and send him all the data regarding the ride
Acknowledgment of reservation to a passenger	Simple	It only require the generation of a string on the base of the outcome of the reservation process.
Sending to the passenger the meeting data of a request	Medium	It involves the calculation of the estimated waiting time
Total FP		**14**

1.Function points calculation

External Inquiry	**Complexity**	**Rationale**
Visualize the list of reservations of a passenger	Simple	From the requirements
Total FP		**3**

1.Function points calculation

FP type	Number of FP
ILF	56
EIF	7
External Input	37
External Output	14
External Inquiry	3
Total FP	**117**

Table 1.6: Total number of function points calculation

2. COCOMO II

- *Source line of code* (SLOC)
    - average conversion factor (JEE) = 46

$$SLOC = 46 \times 117 = 5382$$

2. COCOMO II

- ***Scale drivers***

From which we calculate the *scale exponent*:

$$E = 0.91 + 0.01 \times \sum_{j=1}^{5}(SF_j)$$

NB:: in the documentation there is the explanation of all of them

Scale driver	**Value**
PREC	Very Low
FLEX	High
RESL	Low
TEAM	Nominal
PMAT	Nominal

Table 1.7: Scale drivers values

2. COCOMO II

- ***Cost drivers***

From which we calculate the *effort adjustment factor*: →

$$EAF = \prod_{j=1}^{17} EM_j$$

NB:: in the documentation there is the explanation of all of them

Cost driver	Value
RELY	Nominal
DATA	Nominal
CPLX	Nominal
RUSE	High
DOCU	High

TIME	Nominal
STOR	High
PVOL	Nominal
TOOL	High
SITE	Low
SCED	High

ACAP	High
PCAP	High
AEXP	Low
PEXP	Low
LTEX	High
PCON	Very High

2. COCOMO II

- ***EFFORT CALCULATION***

$$Effort = 2.94 \times EAF \times \left(\frac{SLOC}{1000}\right)^{E}$$

$$Effort_{myTaxiService} = 15.9 \text{ Person-Months}$$

2. COCOMO II

- ***DURATION***

$$Duration = 3.67 \times (Effort)^{SE}$$

$$SE = 0.28 + 0.2 \times (E - 0.91)$$

$$Duration_{myTaxiService} = 11.9 \text{ Months}$$

2. COCOMO II

- ***DIMENSIONING THE TEAM***

$$People = \frac{Effort}{Duration}$$

$$People_{myTaxiService} = 1.36 \approx 2$$

2. COCOMO II

- Monetary cost estimation
    - Lets suppose to pay each member of the team 2500$/month
- Total cost of the project: 39716$ → **40.000$**

3. Task identification and scheduling

PHASE	DEADLINE
RASD	06/11/2015
DD	04/12/2015
ITPD	20/01/2016
PM	02/02/2016
DEVELOPMENT	//
TESTING	//

PHASE	TASKS
RASD	T1: Requirement Specification
	T2: UML Diagrams
	T3: Alloy Model
DD	T4: Architectural Design
	T5: Algorithm Design
	T6: Requirement Traceability
ITPD	T7: ITPD
PM	T8: FP
	T9: COCOMO
	T10: Task Identification
	T11: Resources Allocation
	T12: Risk Management
DEVELOPMENT	T13: Backend
	T14: Frontend
TESTING	T15: Unit Testing
	T16: Integration Testing

3. Task identification and scheduling
4. Resource allocation

- Gantt diagram
- Resource allocation diagram

We need to show them through an appropriate software!

We used the software *Gantt project*.

5. Risk analysis and strategies

- RK0: *Marketing of the solution* (< 60%) → serious
- **RK1: *Lack of personnel* (> 60%) → catastrophic**
- RK2: *Illness of one or more developers* (< 40%) → serious
- RK3: *Lack of experience of personnel* (< 30%) → medium
- RK4: *Client's abandonment of the project* (<10%) → catastrophic
- **RK5: *Lack of budget* (> 30%) → serious**
- RK6: (substantial) *change of requirements* (>20%) → serious
- RK7: *Client's need of a cheap solution* (>70%) → small

5. Risk analysis and strategies

Strategies:

RK1, RK2, RK3 → *staff formation and hiring*
RK4, RK5, RK6, RK7 → *advance of money at the begining of the project. Make a law valuable version of RASD document.* ***Explain to the client the tradeoff between a cheap solution and its quality!***
RK0 → (remote solution) *hire a marketing expert*

Thank you