**2.4 Sequence of component**

We have identified four different subsystems in the organization of our tests.

***Car*** Server

It represents the most critical part for our system because it concerns all functionalities and configurations about cars and their business logic. It takes care for creating communication channels between the system server side, cars and vice versa. Inside itself reservations, cars’ requests, decisions and timers are handled.

***Car*** Client

We have decided to test deeply both server and client side particularly because the communication among each other is a fundamental aspect for the operation mode of the entire system. Indeed, we want to be sure (as more as possible) that the communication works properly. Inside itself driving decisions are taken following server instructions via actuators, sensors and user interfaces are handled.

***Operator*** Server

It is the main part concerning business logic of operators on the server side. We do not spent time on the client side because it is basically a thin client and simple unit tests should be sufficient at this level also because we assume that the business logic is correct. Moreover, operator clients are inside the local private network of our system and we rely on simple local communication protocols. Operator server side developed in this document mainly concerns the notifications dispatcher, paperworks handling, ride report managing and minor issues report managing.

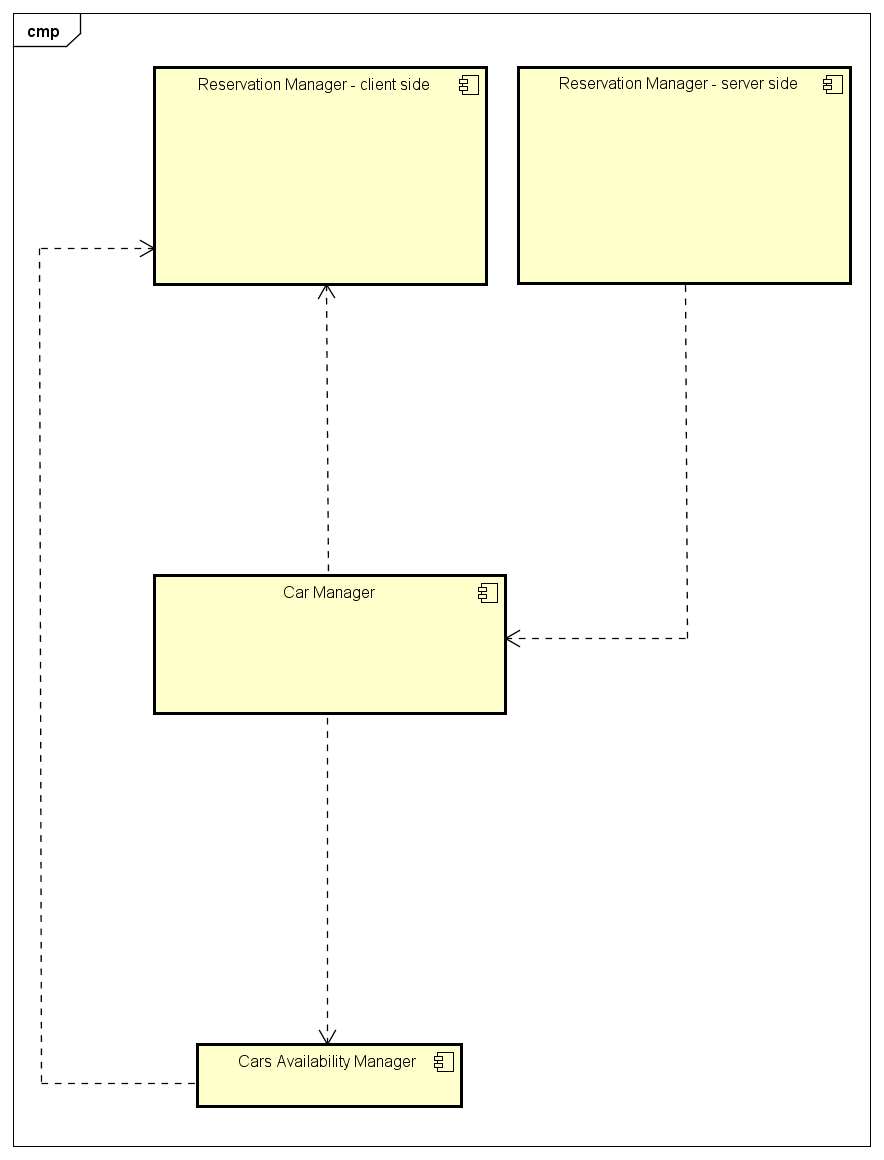
***App User – Web User*** Server

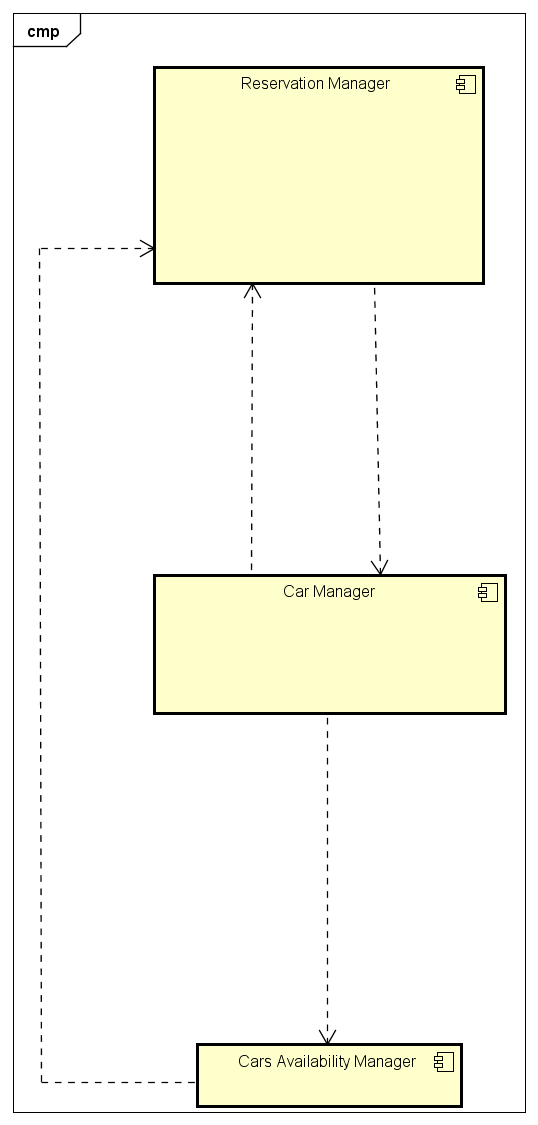
They concern business logic parts present in the server that communicate with the final user. They can be accessed via User Mobile App or via Web Browsing Pages. They allow to exploit functionalities from the system client side. We decided to test deeply these parts because some of client’s decisions may affect their cost affairs and indeed are to be considered critical tasks. The client side of this part is not covered by integration tests because they are basically thin clients and simple unit tests should be sufficient at this level also because we assume that the business logic is correct.

**2.4.1 Software Integration Sequence**

**Loop Dependency**

Da aggiungere didascalia per i diagrammi a sx c’è diagramma dipendenze così come nella sezione 2.2 e a dx come è intesa realmente nella sezione 3 (nelle tabelle) per evitare la dipendenza ciclica.

****



That dependency is because both components need to send asynchronous messages in order to request services to the other one. Indeed, the taxi driver client both asks for a service but provides services also.

At testing layer, we’ve solved the dependency problem splitting the component “Reservation Manager” into two different ones. Each sub-component exports and provides functionalities according to the side with which it interfaces. In that way the component is tested following a functionalities approach.

In section 3, sometimes we used the name “Reservation Manager” to refer to both server and client sides.

**Parallelization Approach**

As stated in section 2.3 in which we state that we use a bottom-up approach for integration tests and as it can be seen in dependences diagrams in section 2.2 we start testing starting from the bottom of diagrams and then we go above integrating focusing on dependences among components. In this way we can easily highlight how integration tests may be parallelized in order to make this task more efficient using different integration tester teams and speeding up the entire testing task.