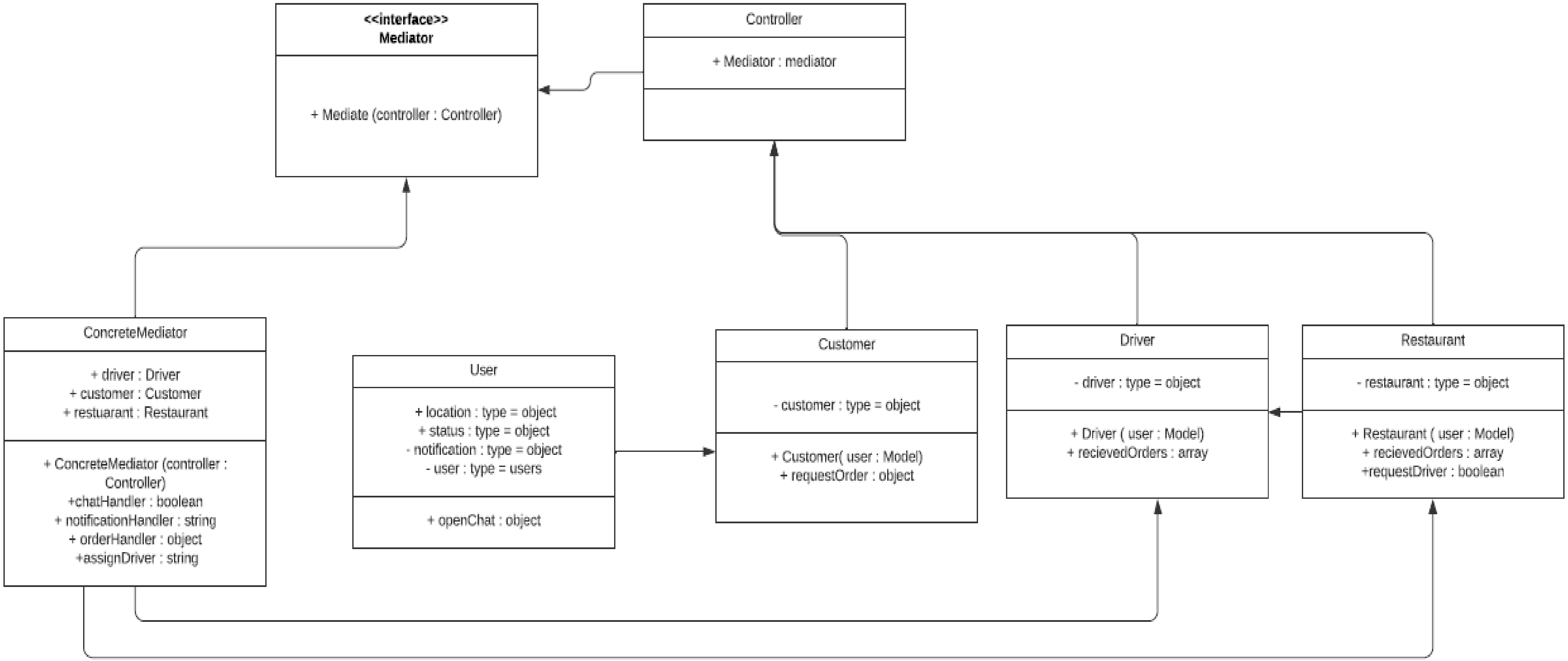
Design Pattern Introduction

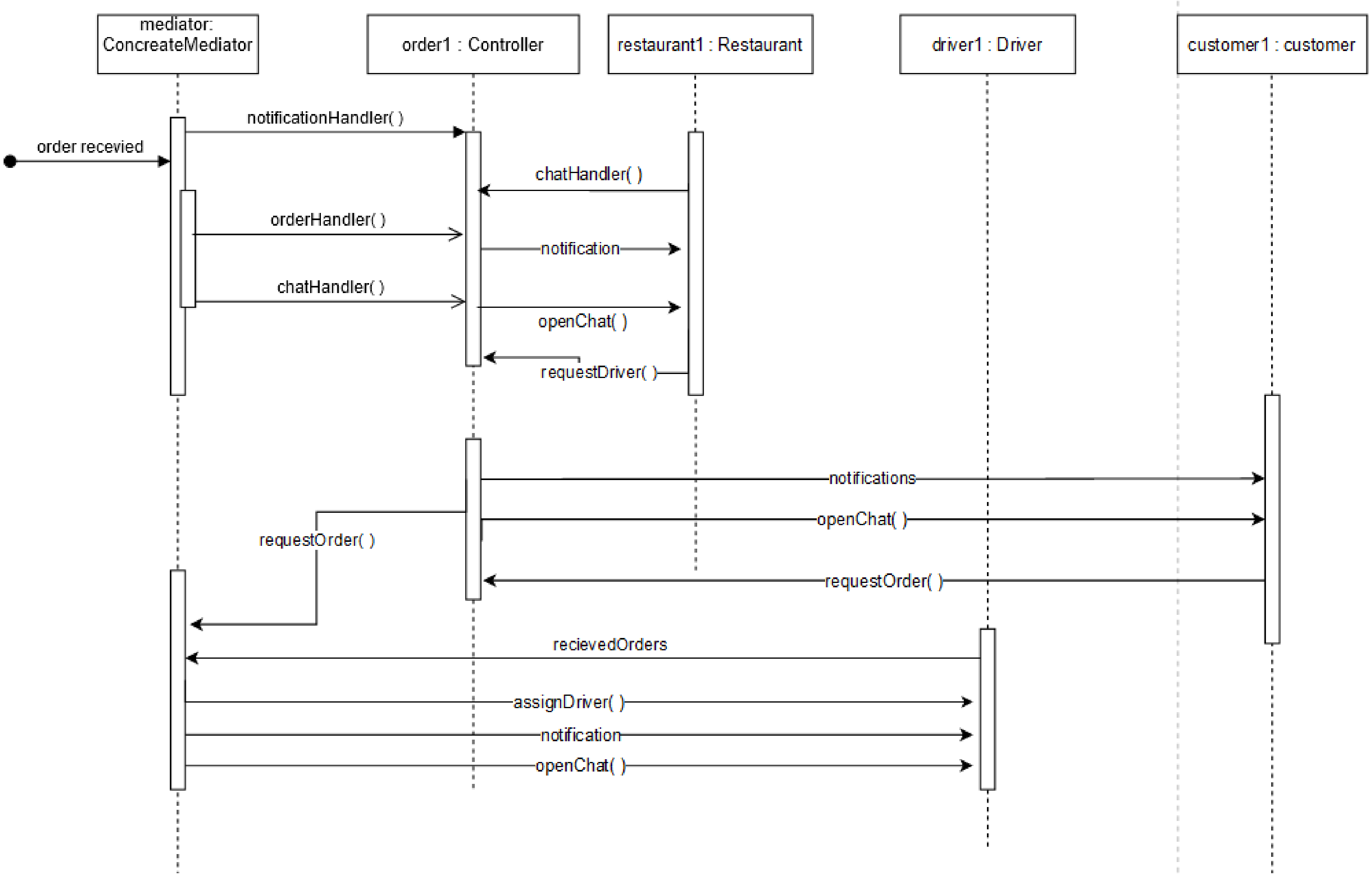
For the message and networking layer, we are leaning more towards the mediator design pattern. It is a behavioural design pattern that has an object that encapsulates how a set of objects interact. When working on networking between different users, i.e. drivers, restaurants and customers, connecting these classes for communication will get complicated with the increase in their numbers, especially during refactoring and maintenance. This makes the program harder to read and maintain. Furthermore, it can become difficult to change the program since any change may affect code in several other classes. With the mediator pattern, communication between objects is encapsulated within a mediator object. Objects no longer communicate directly with each other but instead communicate through the mediator. This reduces the dependencies between communicating objects, thereby reducing coupling.

Another design pattern that we can look at is the composite design pattern. The composite design pattern is a structural design pattern that lets you compose objects into tree-like structures and then work with these structures as if they were individual objects. The composite design pattern allows you to work with complex tree structures more conveniently. It also follows the open/close principle. You can introduce new element types into the app without breaking the existing code, which now works with the object tree. The composite allows objects to use other objects' fields/properties/members through inheritance and composition. With our group's code needing to have objects like the driver, restaurant, and customer, using the composite design pattern be an ideal situation. This will allow the objects such as driver, customer, and restaurant to use methods and variables that are similar across these objects.

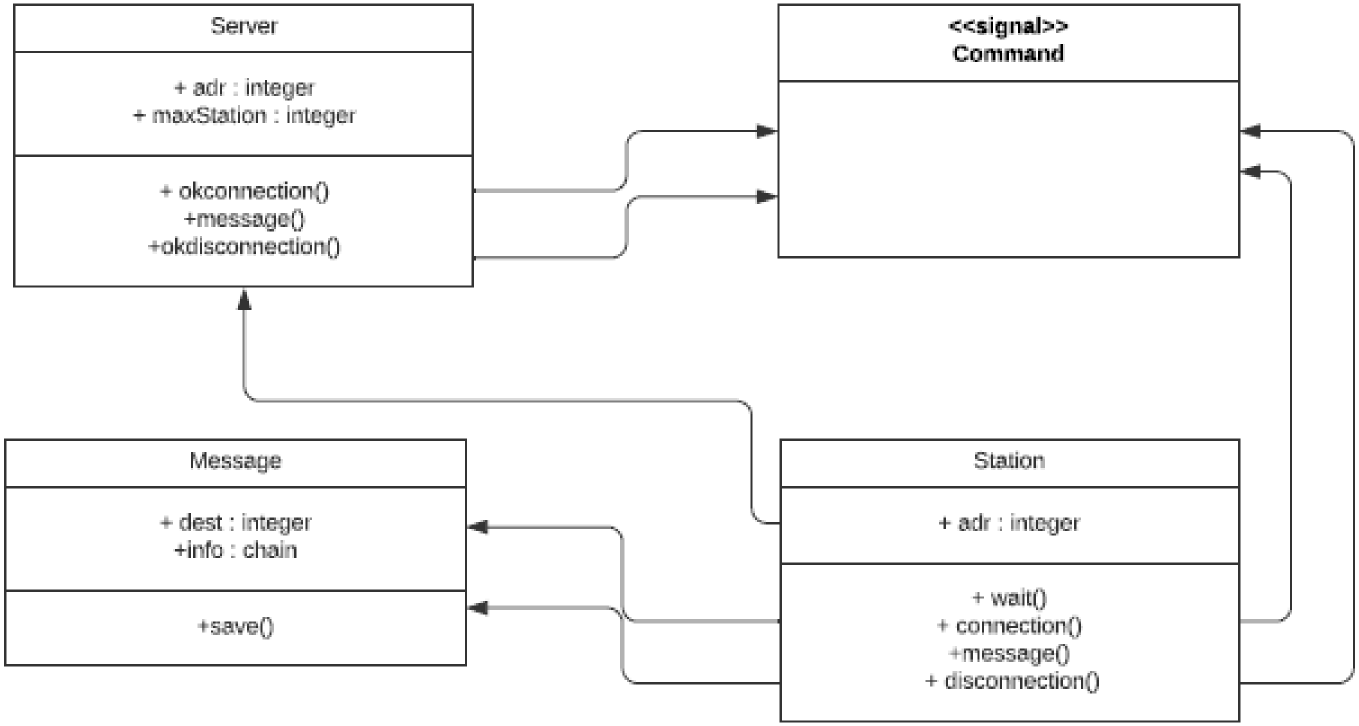
UML diagram for Messaging/Networking and notification implementing the Mediator Design Pattern



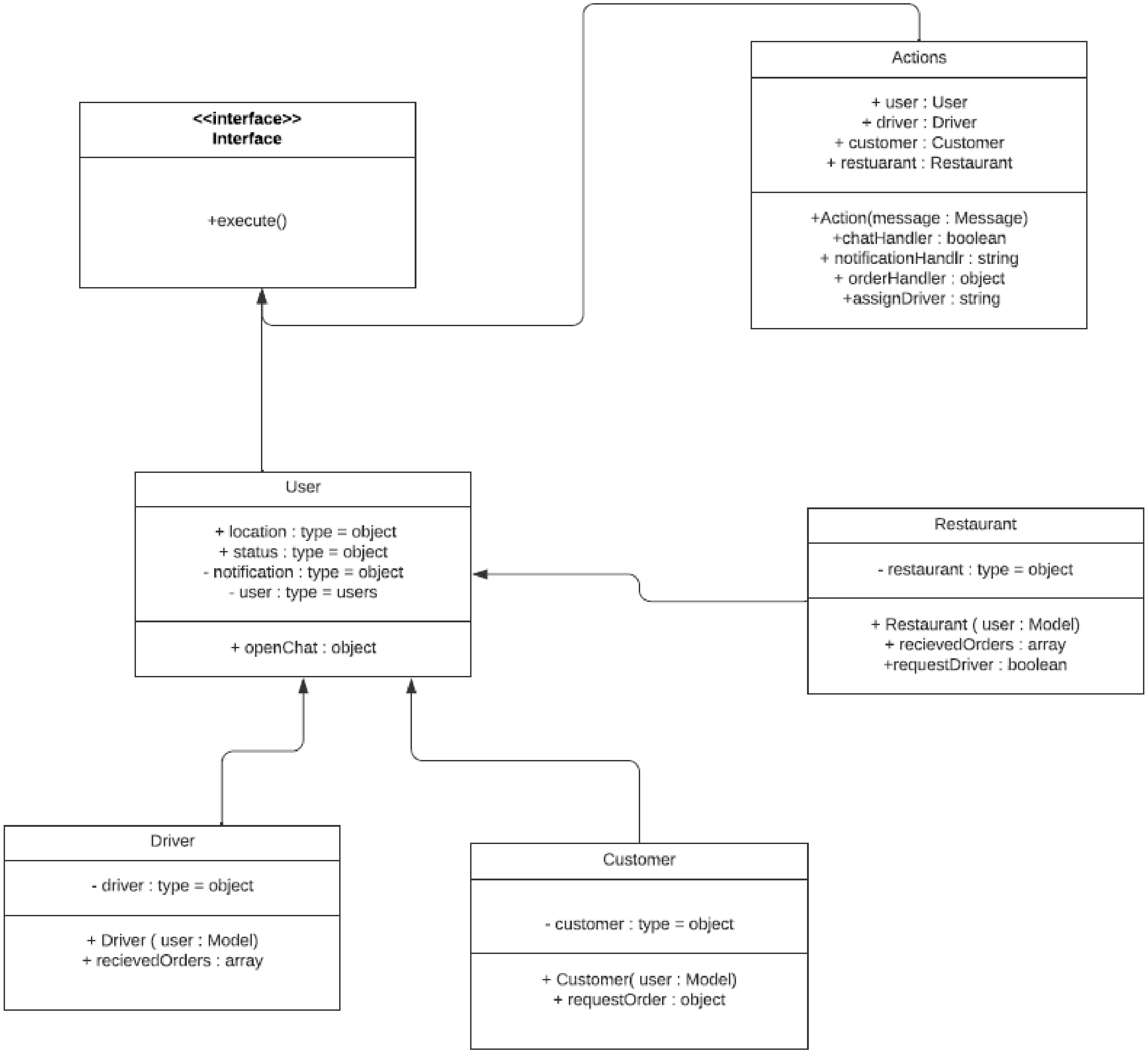
Sequence diagram for Messaging/Networking and notification implementing the Mediator Design Pattern



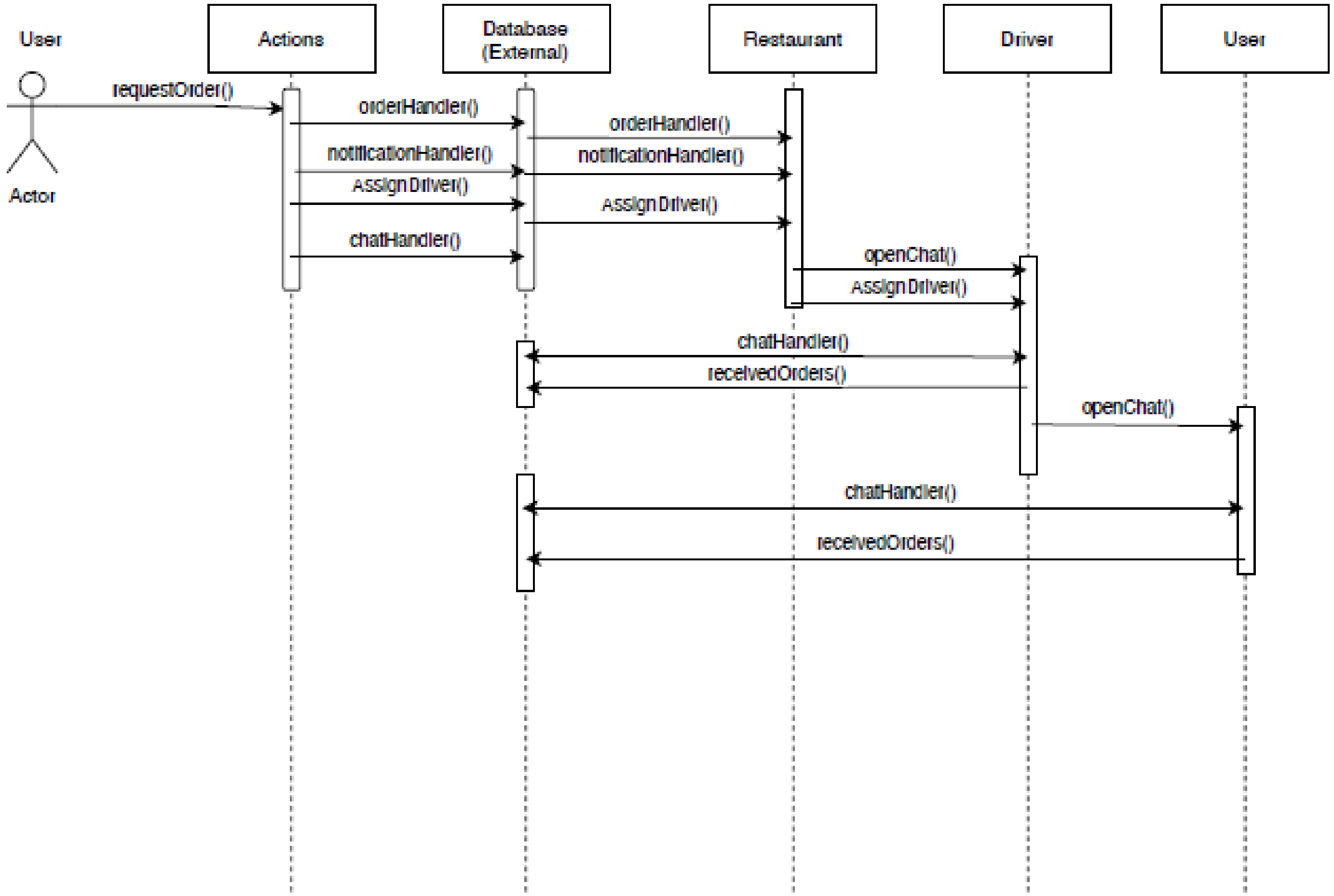
Messaging UML diagram



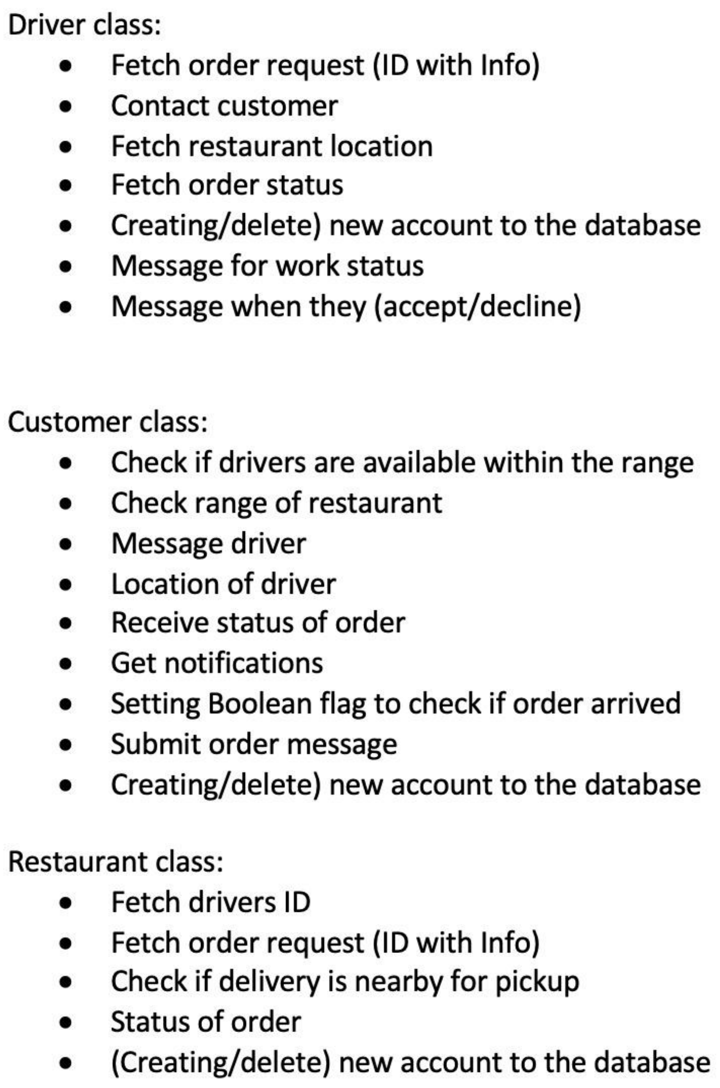
UML diagram for Messaging/Networking and notification implementing the Composite Design Pattern



UML diagram for Messaging/Networking and notification implementing the Composite Design Pattern



Use Cases for the Messaging/Networking Layer



Driver class:

- Fetch order request (ID with Info)
- Contact customer
- Fetch restaurant location
- Fetch order status
- Creating/delete) new account to the database
- Message for work status
- Message when they (accept/decline)

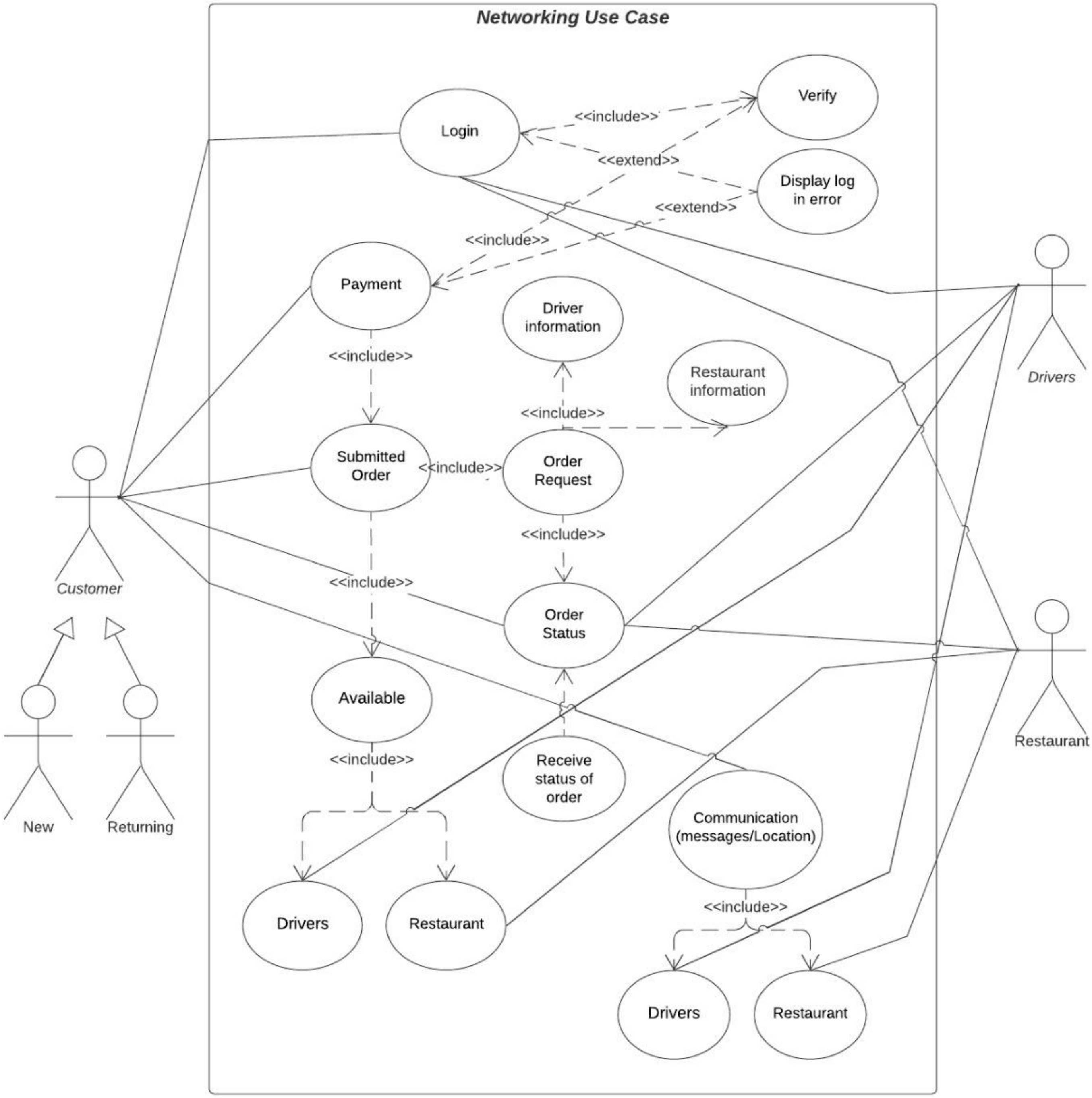
Customer class:

- Check if drivers are available within the range
- Check range of restaurant
- Message driver
- Location of driver
- Receive status of order
- Get notifications
- Setting Boolean flag to check if order arrived
- Submit order message
- Creating/delete) new account to the database

Restaurant class:

- Fetch drivers ID
- Fetch order request (ID with Info)
- Check if delivery is nearby for pickup
- Status of order
- (Creating/delete) new account to the database

Networking Use Case Diagram


Networking Use Case
Login
Verify
<<include>>
<<extend>>
Display log in error
<<extend>>
<<include>>
Payment
Driver information
Restaurant information
<<include>>
<<include>>
Submitted Order
<<include>>
Order Request
<<include>>
<<include>>
Order Status
Available
Receive status of order
<<include>>
Communication (messages/Location)
<<include>>
Drivers
Restaurant
Drivers
Restaurant
Customer
New
Returning
Drivers
Restaurant