Concordia University

COMP 6231 Distributed System

Summer 2019

Project

Tian Wang: 40079289

Minxue Sun: 40084491

Yilin Li: 40083064

Rui Li: 40091912

6231 Project Design Documentation

1. Techniques and Architecture

In this project, we enhance our CORBA implementation of the Distributed Event Management System (DEMS) developed in Assignment 2 to be software failure tolerant and highly available under process crash failure using process replication.
To complete this project, we use Eclipse 4.6.2 with JDK & JRE 1.8.0. The core techniques of this DEMS system is using CORBA and UDP programming.

1.1 CORBA:

The connection between the client and FE is through the Common Object Request Broker Architecture (CORBA) which allows distributed objects to interoperate in a heterogeneous environment.

1.2 UDP/IP:

FE communicates with Sequencer by UDP/IP.
Sequencer communicates with four RMs by UDP/IP.
Four RMs send messages to each other by UDP/IP.
RMs send reply messages to FE by UDP/IP.

1.3 Multicast:

Sequencer sends messages to RMs by reliable basic Multicast.

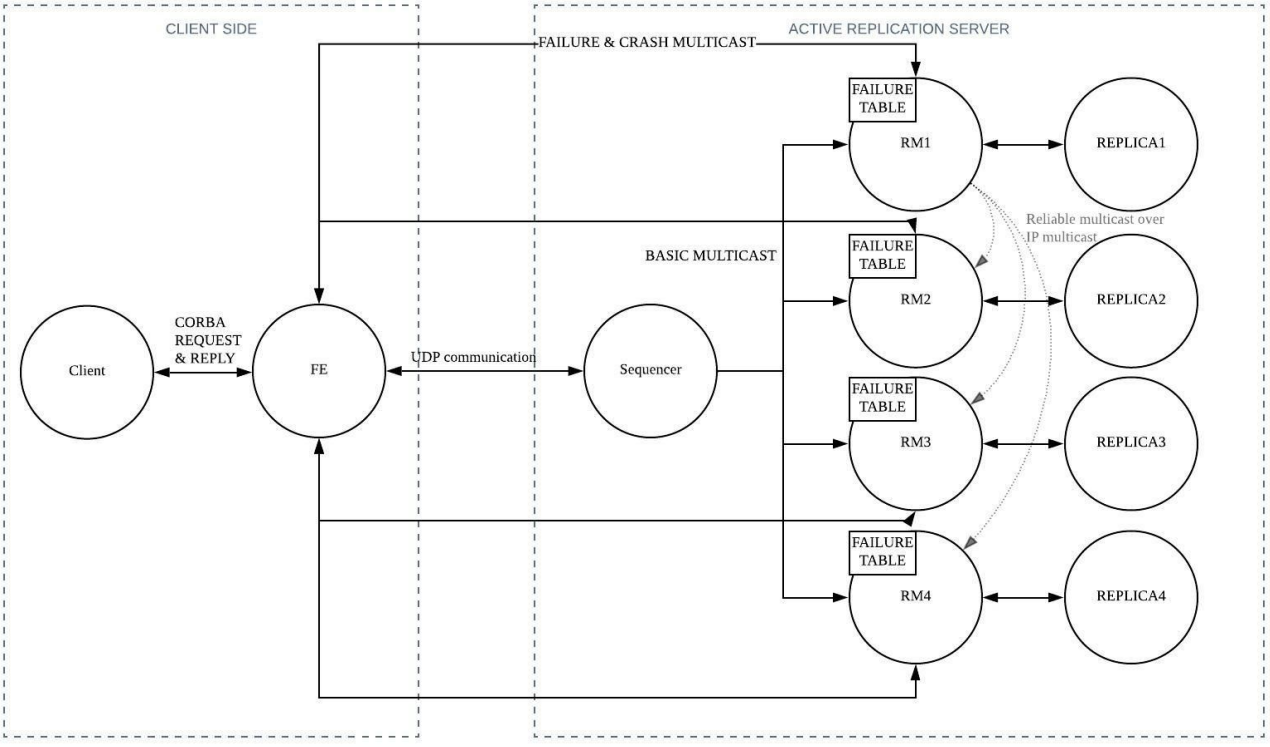


Fig.1 System Architecture

Fig.1 shows the architecture of the System. Here are some key points need to be mentioned besides the requirement of the project description:

- To make UDP/IP communication reliable, When an RM receives a Failure or Crash message from FE, it would automatically send the message to the other three RMs without modification.
- In our system, we need four replicas, but there are three different implementations of the DEMS In our group so far. So we decide to duplicated use one of them.

2. Project Structure and Implementation

2.1 Front End (FE):

Responsibilities:

FE - Client:
- receives a client request as a CORBA invocation.
- sends a single correct result back to the client.

FE - Sequencer:
- forwards the request to a failure-free sequencer.

FE - RM:
- receives the results from four RMs.
- finds the majority of the receiving results and reply to the client.
- If anyone of the Replica produces an incorrect result, the FE informs all the RMs about that replica.
- if FE does not receive a response from any one of the RMs during a certain time slot. FE will automatically multicast to all the RMs that this Replica has crashed.

2.2 Sequencer:

Sequencer - FE:
- receives a client request from FE and assigns a unique sequence number to the request.

Sequencer - RM:
- multicasts the request with the sequence number to all the replicas reliably.

2.3 Replica Manager (RM):

RM:
- Creates and initializes the actively replicated server system.
- Detects and recovers from a failure.

(1) One replica produces incorrect results for three client requests, then the RM fixes the bugs.

(2) One replica timeout. There are 6 steps as follows:

a. FE checks one replica that did not produce the result in a reasonable time.
b. FE send crash message to all RMs.
c. All RMs send “IfCrash” messages to the RM that did not produce the result in a reasonable time.
d. The RM check if it was crash, if it is crash then will send “DidCrash” to other RMs, else will send “NotCrash” messages.
e. If one RM receive “DidCrash” from the crash replica’s RM then send “RestartReplica” message back to the RM.
f. If the crash replica’s RM has received three “RestartReplica” message from all other RMs then restart its replica.

RM - FE:
- Return the results back to the FE.

RM - Sequencer:

- executes the client requests in total order according to the unique sequence number.

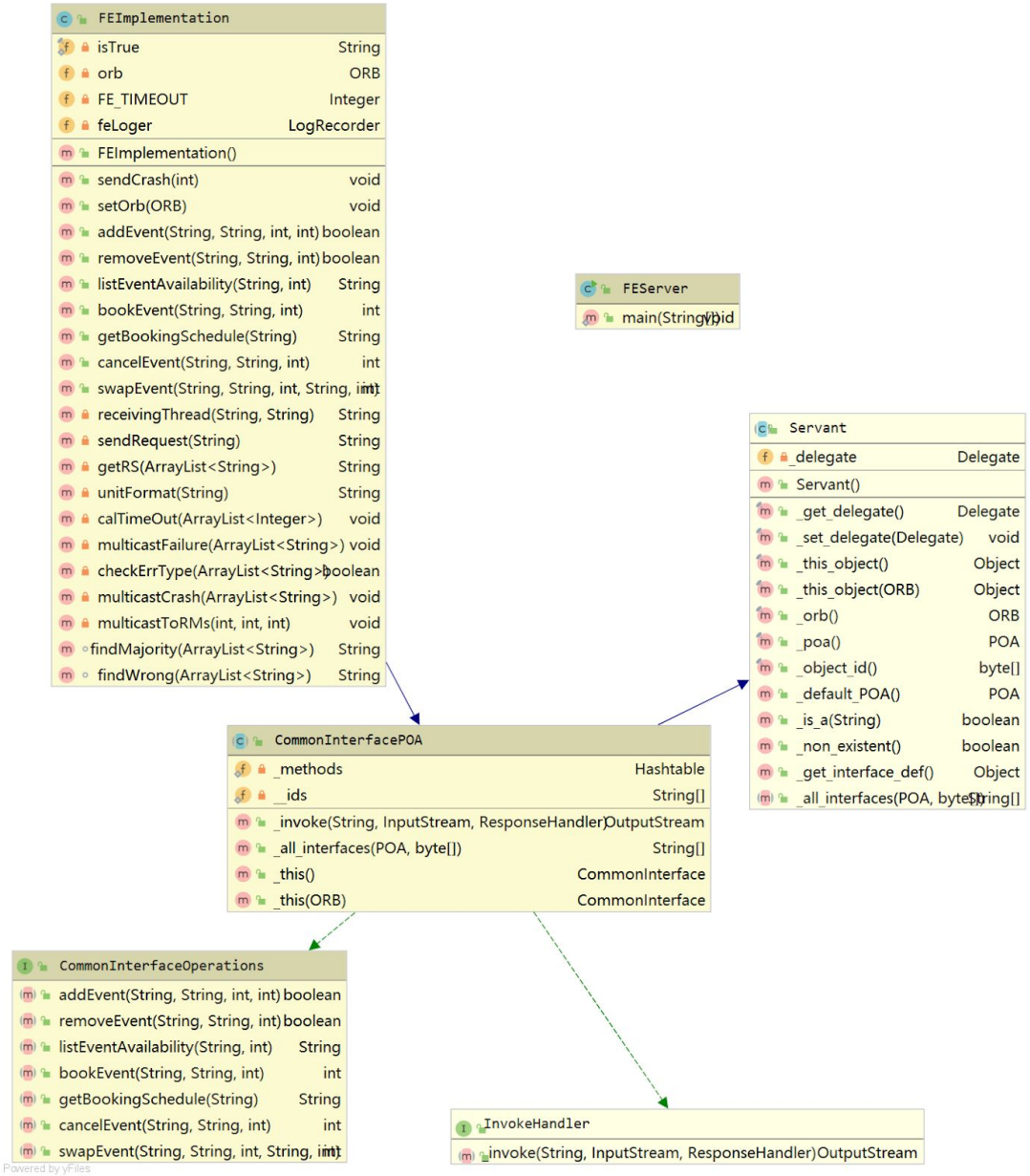
RM - RM:

- forwards Crash or Failure messages sent from FE to other RMs.

2.4 Replicas

There are four replicas which consist of three different implementations. One of the replicas returns error results after executing the requests, one crashes, and the remaining two always keep the correct execution of the requests. When two of the four replicas fail at the same time (one failure, one crash), the FE can still get the correct information and pass it to the client. This guarantees the fault tolerance of the system.

2.5 ClassDiagram of Client-Side



3. Important Algorithms

3.1 Reliable Multicast

The RMs will forward all received messages from the FE and Sequencer to other RMs. The RM will check the message when it receives that, and ignore the message which was received before, otherwise, the message will be executed.

```
On initialization
   Received := {};

For process p to R-multicast message m to group g
   B-multicast(g, m);          // p ∈ g is included as a destination

On B-deliver(m) at process q with g = group(m)
   if (m ∉ Received)
   then
            Received := Received ∪ {m};
            if (q ≠ p) then B-multicast(g, m); end if
            R-deliver m;
   end if
```

pseudo code of Reliable Multicast[1]

3.2 Total Order

Sequencer: The Sequencer will add a sequence id to the message received from the FE and send it to the four RMs.

RMs: RMs will put the received messages into the queue according to their sequenceIDs, which ensures that all servers execute messages according to the total order.

3.3 Timer

We implemented a timer to refresh the timeout whenever FE receives a request from one of the clients. By using an array to store the time of every replica replying the certain request. The global variable, the timeout for detecting the server crashes will be updated as two times of longest one of the previous arrays. In other words, FE can automatically adjust the timeout according to the network environment.

4. Data Structures

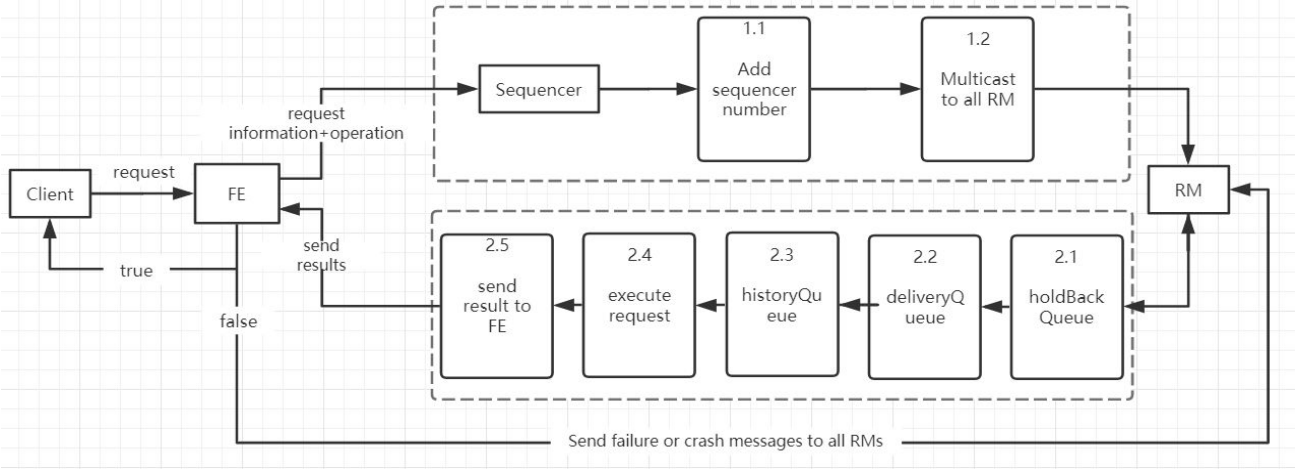
There are several important data structures in the RM, such as holdBackQueue, deliveryQueue, historyQueue, failureQueue and Counter.

historyQueue: When the message is going to be executed, the request will be stored in the historyQueue in order to recover the previous operation when the replica is restarted.

failureQueue: In order to ensure the reliability of UDP transmission, when RM receives the failure message by FE, it will send the message to all the remaining RMs only when the RM receives the message at the first time. The message will be placed in the failure queue. This queue not only guarantees the failure message transmission between RMs, increases the reliability of transmission but also ensures that each RM maintains the same failure number statistics table.

Counter: The data structure is a HashMap, which is used to update the number of failures. When a failure message is received, the number of failures corresponding to the failure of the replicaId is increased by one. When it is accumulated to three times, it modified the error, then the value is reset to 0.

5. Dataflow



6. Important/Difficult Part

6.1 FE integrates responses from different RMs

There are three different background implementations in our group, so the return value type or format of each function is different. When the RM returns the execution result to the FE, the FE needs to compare the returned results to get the correct result and return the execution result to the client. Therefore, it is very important to unify the results of the three replica functions.

6.2 The communication between RMs

When the FE sends the error message to all RMs, the communication between the RMs uses reliable multicast. Each RM will only multicast to all the remaining RMs when the message is received for the first time, guaranteeing each The RM will receive the error message and receive it only once, ensuring the correctness of the error counter, thus achieving reliability and consistency. When the FE sends the crash message to all RMs, each RM checks the replicaId and sends the message to the RM corresponding to that id. If the remaining RMs confirm that the RM has been crashed, then notify the RM to restart the replica and recovery the data.

6.3 RMs recover from failure or crash

Recover from failure : The system defaults to the first three operations of one of the replicas (sequenceId is 0, 1, 2) and returns the wrong result. When the number of errors of this replica reaches three, the counter is automatically cleared, and when the fourth request is accepted, the system automatically fix the failure and return the correct result to the FE.

Recover from crash : The system will put the request into the historyQueue before executing each request. When the remaining RMs confirm that the RM has crashed, the RM will automatically clear all the data in the data structure and execute all the requests in the historyQueue, so the data consistency is guaranteed. And when recovering from a replica crash, the global variable of historyQueue is kept unchanged by setting the local variable and the global variable historyQueue respectively. Then, only the global variable of historyQueue is used to ensure that multiple crashes and multiple recovery can be achieved.

6.4 Make FE and client independent from Server

In our implementation, we need to make FE and client independent which means even if the FE and the client stop, the entire system will not crash and all data will not lose. The way to achieve it is keeping the message data in the system on the RM sides, such as the failure tables of the RMs

and the queues of all messages received.

7. Test Case

If log in succeed, the options that the manager and the customer can choose as below separately: Here are all test cases to ensure the robustness of DEMS：

7.1 Failure and Crash

(1) Failure only

No.	Case description	Precondition	Input data	Expected result	Test result
1	Only one replica returns wrong result, others execute correctly	when sequence number is from 0-2	any operation except the listEventAvailability and getBookingSchedule	the replica manager of failed replica sends back wrong results.	As expected

(2) Crash only

No.	Case description	Precondition	Input data	Expected result	Test result
2	One repica crash, and others execute correctly	FE setup crash number of intended crashed replica	any operation	nothing received from the crashed replica manager	As expected

RM: The figure describes that RM detects whether the replica crashes, if the replica crashes, RM restarts the replica (recover requests in the historyQueue).

```
Crash-------
UDP receive: Crash:5:2
This Replica maybe crash!
RM2 crash Listener receive: IfCrash
DidCrash
RM2 crash Listener receive: IfCrash
DidCrash
RM2 crash Listener receive: IfCrash
DidCrash
RM2 crash Listener receive: RestartReplica
RM2 crash Listener receive: RestartReplica
RM2 crash Listener receive: RestartReplica
RM 2 restart replica 2...
recover --> addEvent,MTLM0001,MTLM190324,1,2
recover --> addEvent,MTLM0001,MTLA190233,2,2
recover --> addEvent,MTLM0001,MTLA111122,3,2
recover --> listEventAvailability,MTLM0001,1
recover --> listEventAvailability,MTLM0001,2
recover --> listEventAvailability,MTLM0001,1
```

(3) Simultaneous occurrences

No.	Case description	Precondition	Input data	Expected result	Test result
3	One replica returns wrong result, one replica crash, the other two are fine	when sequence number is from 0-2 FE setup crash number of intended crashed replica	any operation except the listEventAvailability and getBookingSchedule	Front End received a wrong response from the failed replica manager and received nothing from the crashed replica.	As expected

RM: Each RM will receive four times of the same message, one time is sent by FE or Sequencer, the other three come from the remaining three RMs, which guarantees the reliability of the multicast.==

```
0-------
UDP receive: 0:132.205.46.181,5000:addEvent,MTLM0001,MTLM190324,1,2
RM 3 send message to other RMs: 0:132.205.46.181,5000:addEvent,MTLM0001,MTLM190324,1,2
manager MTLM0001 add event MTLM190324 in conference
Replica reply: 0:3:F:
0-------
UDP receive: 0:132.205.46.181,5000:addEvent,MTLM0001,MTLM190324,1,2
0-------
UDP receive: 0:132.205.46.181,5000:addEvent,MTLM0001,MTLM190324,1,2
0-------
UDP receive: 0:132.205.46.181,5000:addEvent,MTLM0001,MTLM190324,1,2
Failure-------
UDP receive: Failure:0:3
Replica 3 has failed 1 times
RM 3 send failure message to other RMs: Failure:0:3
Failure-------
UDP receive: Failure:0:3
Failure-------
UDP receive: Failure:0:3
Failure-------
UDP receive: Failure:0:3
1-------
UDP receive: 1:132.205.46.181,5000:addEvent,MTLM0001,MTLA190233,2,2
RM 3 send message to other RMs: 1:132.205.46.181,5000:addEvent,MTLM0001,MTLA190233,2,2
manager MTLM0001 add event MTLA190233 in seminar
Replica reply: 1:3:F:
1-------
UDP receive: 1:132.205.46.181,5000:addEvent,MTLM0001,MTLA190233,2,2
1-------
UDP receive: 1:132.205.46.181,5000:addEvent,MTLM0001,MTLA190233,2,2
1-------
UDP receive: 1:132.205.46.181,5000:addEvent,MTLM0001,MTLA190233,2,2
Crash-------
UDP receive: Crash:1:2
send crash message to other RM:IfCrash
Failure-------
UDP receive: Failure:1:3
Replica 3 has failed 2 times
RM 3 send failure message to other RMs: Failure:1:3
Failure-------
UDP receive: Failure:1:3
Failure-------
UDP receive: Failure:1:3
Failure-------
UDP receive: Failure:1:3
```

```
2-------
UDP receive: 2:132.205.46.181,5000:addEvent,MTLM0001,MTLA111122,3,2
RM 3 send message to other RMs: 2:132.205.46.181,5000:addEvent,MTLM0001,MTLA111122,3,2
manager MTLM0001 add event MTLA111122 in tradeshow
Replica reply: 2:3:F:
2-------
UDP receive: 2:132.205.46.181,5000:addEvent,MTLM0001,MTLA111122,3,2
2-------
UDP receive: 2:132.205.46.181,5000:addEvent,MTLM0001,MTLA111122,3,2
2-------
UDP receive: 2:132.205.46.181,5000:addEvent,MTLM0001,MTLA111122,3,2
Failure-------
UDP receive: Failure:2:3
Replica 3 has failed 3 times
Replica 3 has been fixed.
RM 3 send failure message to other RMs: Failure:2:3
Failure-------
UDP receive: Failure:2:3
Failure-------
UDP receive: Failure:2:3
Failure-------
UDP receive: Failure:2:3
```

FE:

```
FrontEndServer is Started.....
Request message sent from the client is : 132.205.46.181,5000:addEvent,MTLM0001,MTLM190324,1,2
Receive msg form replica2 : 0:2:T:Add event Conferences MTLM190324 successfully!
Receive msg form replica1 : 0:1:T:Add event Conferences MTLM190324 successfully!
Receive msg form replica4 : 0:4:T:successfully
Receive msg form replica3 : 0:3:F:
Wrong type: Failure|Wrong Replica:3|Wrong Snum:0
Current FETimeOut updated: 1000
----------------------------Add event Conferences MTLM190324 successfully!
2
Crash demand has been sent to : 132.205.46.179,6666
Request message sent from the client is : 132.205.46.181,5000:addEvent,MTLM0001,MTLA190233,2,2
Receive msg form replica1 : 1:1:T:Add event Seminars MTLA190233 successfully!
Receive msg form replica4 : 1:4:T:successfully
Receive msg form replica3 : 1:3:F:
Wrong type: Crash|Wrong Replica:2|Wrong Snum:1
Wrong type: Failure|Wrong Replica:3|Wrong Snum:1
Current FETimeOut updated: 1000
----------------------------Add event Seminars MTLA190233 successfully!
Request message sent from the client is : 132.205.46.181,5000:addEvent,MTLM0001,MTLA111122,3,2
Receive msg form replica1 : 2:1:T:Add event Trade Shows MTLA111122 successfully!
Receive msg form replica2 : 2:2:T:Add event Trade Shows MTLA111122 successfully!
Receive msg form replica4 : 2:4:T:successfully
Receive msg form replica3 : 2:3:F:
Wrong type: Failure|Wrong Replica:3|Wrong Snum:2
Current FETimeOut updated: 1000
----------------------------Add event Trade Shows MTLA111122 successfully!
Request message sent from the client is : 132.205.46.181,5000:listEventAvailability,MTLM0001,1
Receive msg form replica4 : 3:4:MTLM190324-2,
Receive msg form replica1 : 3:1:MTLM190324-2,
Receive msg form replica2 : 3:2:MTLM190324-2,
Receive msg form replica3 : 3:3:MTLM190324-2,
Current FETimeOut updated: 1000
----------------------------[MTLM190324-2]
Request message sent from the client is : 132.205.46.181,5000:listEventAvailability,MTLM0001,2
Receive msg form replica4 : 4:4:MTLA190233-2,
Receive msg form replica2 : 4:2:MTLA190233-2,
Receive msg form replica1 : 4:1:MTLA190233-2,
Receive msg form replica3 : 4:3:MTLA190233-2,
Current FETimeOut updated: 1000
```

7.2 Swap

(1) Swap old local event with new nonlocal event (when there are three nonlocal events and one local event in booking list)

FE:

```
---------------------------[1-MTLM190324, 2-MTLM130324, 2-OTWA110324, 3-MTLM150324]
Request message sent from the client is : 132.205.46.181,5000:swapEvent,OTWC1001,TORA220324,1,OTWA110324,2
Receive msg form replica1 : 19:1:F:Customer OTWC1001 have already booked 3 events from other cities in 03 month! Swap failed!
Receive msg form replica2 : 19:2:F:Customer OTWC1001 have already booked 3 events from other cities in 03 month! Swap failed!
Receive msg form replica3 : 19:3:F:
Receive msg form replica4 : 19:4:F:booked 3
Current FETimeOut updated: 1000
---------------------------Customer OTWC1001 have already booked 3 events from other cities in 03 month! Swap failed!
```

CLIENT:

```
--------------------------------------------------------------------------------
Please choose the option as follow(input the character in brackets to continue):
1. (B)ook an event.
2. (L)ist all your current events.
3. (C)ancel your current event.
4. (S)wap your current event.
5. (E)xit.
S
Please input the old eventType:
SEM
Please input the old eventID:
OTWA110324
Please input the new eventType:
CON
Please input the new eventID:
TORA220324
You can't book more than three events outside your city!
--------------------------------------------------------------------------------
```

(2) New event has been booked

FE:

```
Request message sent from the client is : 132.205.46.181,5000:swapEvent,OTWC1001,OTWA110324,2,MTLM190324,1
Receive msg form replica1 : 21:1:F:Customer OTWC1001 have already booked the eventSeminars OTWA110324. Swap failed!
Receive msg form replica2 : 21:2:F:Customer OTWC1001 have already booked the eventSeminars OTWA110324. Swap failed!
Receive msg form replica4 : 21:4:F:otherRs
Receive msg form replica3 : 21:3:F:
Current FETimeOut updated: 1000
---------------------------Customer OTWC1001 have already booked the eventSeminars OTWA110324. Swap failed!
```

CLIENT:

```
--------------------------------------------------------------------------------
Please choose the option as follow(input the character in brackets to continue):
1. (B)ook an event.
2. (L)ist all your current events.
3. (C)ancel your current event.
4. (S)wap your current event.
5. (E)xit.
S
Please input the old eventType:
CON
Please input the old eventID:
MTLM190324
Please input the new eventType:
SEM
Please input the new eventID:
OTWA110324
Sorry, the event you swap to has already been booked by yourself
--------------------------------------------------------------------------------
```

(3) New event was not exist

FE:

```
Request message sent from the client is : 132.205.46.181,5000:swapEvent,OTWC1001,OTWA110324,1,OTWA110324,2
Receive msg form replica1 : 20:1:F:The event Conferences OTWA110324 is not exit. You can't book it. Please try again! OTWC1001 Swap failed!
Receive msg form replica2 : 20:2:F:The event Conferences OTWA110324 is not exit. You can't book it. Please try again! OTWC1001 Swap failed!
Receive msg form replica4 : 20:4:F:otherRs
Receive msg form replica3 : 20:3:F:
Current FETimeOut updated: 1000
----------------------------The event Conferences OTWA110324 is not exit. You can't book it. Please try again! OTWC1001 Swap failed!
```

CLIENT:

```
--------------------------------------------------------------------------------
Please choose the option as follow(input the character in brackets to continue):
1. (B)ook an event.
2. (L)ist all your current events.
3. (C)ancel your current event.
4. (S)wap your current event.
5. (E)xit.
S
Please input the old eventType:
SEM
Please input the old eventID:
OTWA110324
Please input the new eventType:
CON
Please input the new eventID:
OTWA110324
Sorry, the event you swap to doesn't exist!
--------------------------------------------------------------------------------
```

8. Members Responsibility

Everyone is involved in the modification of their own replica. the specific work for everyone is going to be shown below:

- Tian Wang (40079289): Replica manager, sequencer, test cases
- Yilin Li (40083064): Replica manager, sequencer, test cases
- Minxue Sun (40084491): Replica manager, sequencer, test cases
- Rui Li (40091912): Design and implement the front end (FE) and client. Be responsible for the uniforms of the responses from all the replica managers.

Reference:

[1] Lecture Note : Group Communication p.8.