



Project:

Software Failure Tolerant and/or Highly Available Distributed Event Management System

Submitted by:

Name: Shubham Ranadive

Student ID: 40083991

Name: Krisha Patel

Student ID: 40084336

Name: Dhruvi Gadhiya

Student ID: 40084176

Table of Contents

Introduction	2
Technology Used	3
CORBA	3
User Datagram Protocol(UDP)/Internet Protocol(IP)	3
System Specifications	4
Difficult Parts	5
Design Architecture and Data Flow	6
Front End(FE)	6
Replica	7
Replica Manager	7
Test Cases	8
Book Event and Cancel Event	8
Crash Result	8
Software Bug	9
Multithreading	9
Sequencer Output	10

Introduction

Distributed Event Management System(DEMS) is a software which manage events (Conferences, Seminars or Trade Shows). There are Managers and Customers to access the System for booking and canceling of events. Customers can book only 3 events outside of their city. They can book and cancel events and also can view their schedule for all the events. Customers can swap events according to their need. Managers can add new events or remove existing event for their city along with can list all the events of specific event type with its availability. Manager can also perform all the tasks for Customers of their city.

Technology Used

CORBA

The Common Object Request Broker Architecture (CORBA) is a standard developed by the Object Management Group (OMG) to provide interoperability among distributed objects. CORBA is independent of hardware platforms, programming languages, and operating systems. It is essentially a design specification for an Object Request Broker (ORB), where an ORB provides the mechanism required for distributed objects to communicate with one another, whether locally or on remote devices, written in different languages, or at different locations on a network.

It is used to design the interfaces containing the methods, using which the customers and the managers – the clients, can communicate and invoke their respective operations i.e. perform the RPC, respective to their cities– the servers, which performs the invoked operations and returns the result to the clients.

User Datagram Protocol (UDP)/ Internet Protocol (IP)

UDP is used primarily for establishing low-latency and loss-tolerating connections between applications on the internet. It is not connection oriented protocol as opposed to its counter-part TCP and thus, it is faster and less reliable protocol.

UDP is used for inter-server communication to perform operations of some city, for the clients of other city, for example, customer of one city can only access data and invoke operations, respective to their own city, but for them to book in events of cities, other than their own, they need to send request to their own city and then their city can request, on their behalf to the other city to perform such operations. Such operations can be performed by one server (city) sending UDP message to other server including information that enables the receiving server to understand what operation is being invoked / requested by the sending server. For this purpose UDP is used for inter-server communication.

System Specifications

Identifiers: The Customers and the Managers, the clients, are identified by unique identifiers CustomerID and ManagerID, which is constructed from the acronym of their city and a 4-digit number (e.g. MTLM1234 for a Montreal Manager and MTLC1234 for a Montreal Customer).

Cities: There are 3 cities, viz: Montreal (MTL), Toronto (TOR) and Ottawa (OTW), which are implemented as 3 different servers in the system, each having their own pre-authorized set of Managers who can only perform the operations, and unauthorized Managers will be checked and refrained from performing the operations. Only the Customers of the city can access that city's server directly, whereas, the Customers of other cities need to make a request to their own city, which in turn requests the other city to perform the specific function requested which are performed in such a manner that the client is not aware of the inter-server communication and it appears to the client as if the operation requested is being performed on the server it sent the request to i.e. maintaining the location transparency and access transparency.

Customers: One of the clients of the system, the Customers, identified by their unique identifiers, can only access the data and invoke functions respective to their own city using JAVA RMI. The Customer has no limit to book events in their own city. However, customer cannot book more than 3 events outside their own city in a given month. They can cancel the event that they are already booked at any time. They can also view their entire schedule, i.e. events book by them. They can swap the events they have already booked with the new ones.

Managers: The other client, the Managers, can add, and remove, the events for their own city and also get list of all the events for a specific type from other cities to view to the Customers, and can also book, and cancel the events for the Customers from, for any type, i.e. the Manager can perform all the functions of the Customers, given the constraints imposed on booking and canceling are satisfied. The Manager, therefore, can perform all the functions there are in total in the entire system.

Important Parts

The important part was to make the system work correctly with every function performing the way it should as described in the system specifications and thus updating every data structure after every operation performed correctly and thus maintaining system consistency. Identifying from the ID, whether the client is a manager or can be customer and thus assigning the operations allowed to the client. All the methods of customer can be access by manager also but visa versa is not possible. Also, checking on the server side, the eventID, to know whether the client is trying to access the event of the same city or it belongs to some other city and thus proceeding with either executing the methods on the same server or starting the interserver communication. Also maintaining logs on both the server and the client side is important to later verify the flow of the system and operations performed.

Identifying and handling software bugs and crash failure and maintaining data consistency across all replicas.

Moreover, obtaining local and access transparency is as important as any other objective to make the system work properly using CORBA and UDP.

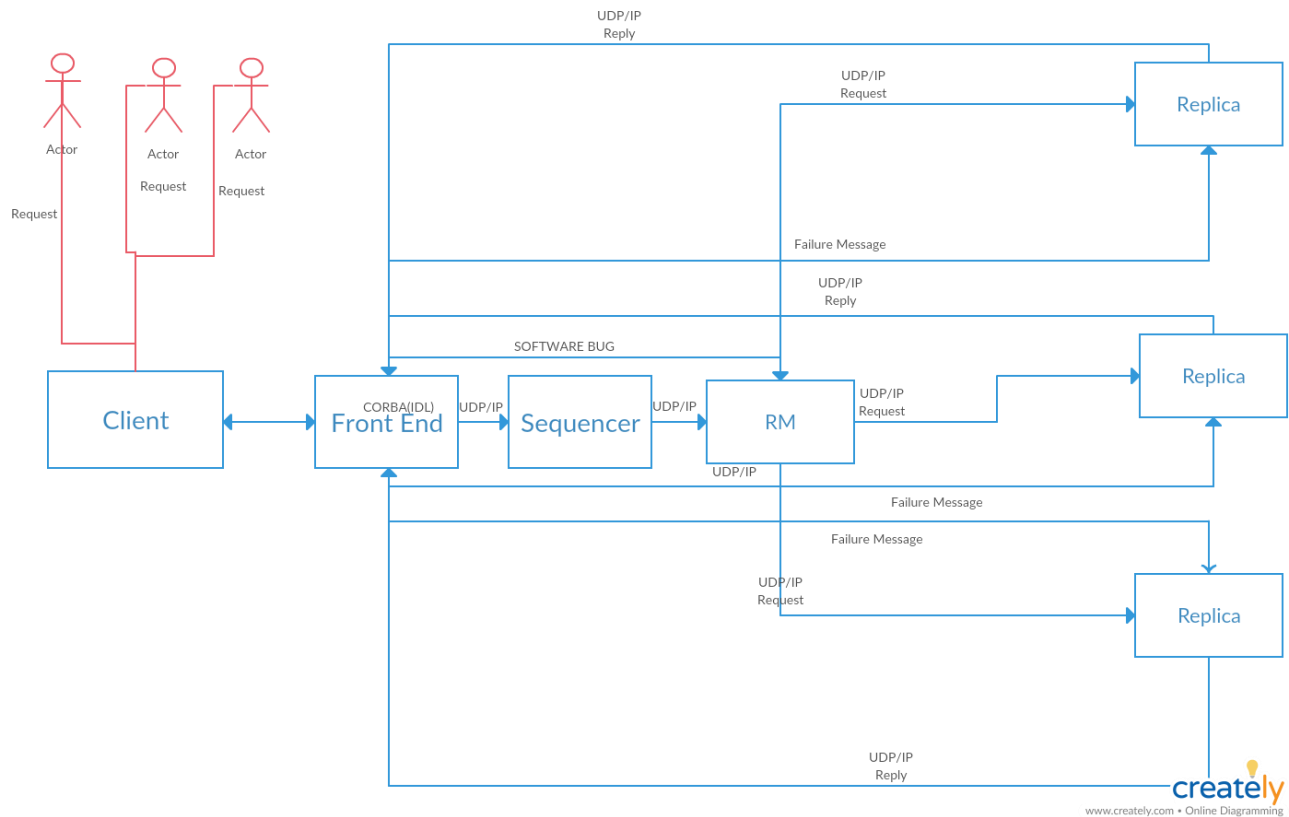
Difficult Parts

The most difficult part of the entire system is obtaining location and access transparency and atomic and concurrent access of shared data. All these obtained by using UDP and CORBA and multithreading synchronization technologies makes it easy.

Designing one server is the most important part, as, if it gets designed perfectly, the other servers can be easily designed based on how the first one is designed and thus making the system development easy.

Main task of sending the requests between the servers based on the input by the clients and updating the data structures based on different scenarios arising after each step and returning correct result to the client after performing all the steps correctly can be obtained by properly synchronizing the communication between the servers i.e. keeping all the servers in listening mode as long as they are running and thus being able to receive messages sent by other servers and performing the asked operation and also sending back the results to the server using UDP and from that server sending back the result to the client using CORBA. Moreover, enabling multiple clients to perform different/same functions at the same time while maintaining the state of data using multithreading synchronization is a difficult aspect to implement.

Design Architecture & Data Flow



Front End (FE)

- FE is a CORBA implementation, which contains the interface definition of all the replicas.
- Client will send request to Front end, which will be a CORBA implementation, there will be new front end instantiated for each client request.
- All the FE's thus instantiated will send the request to single sequencer through UDP message.
- FE will wait for the reply from all the replicas and based on the reply it will send the correct result to the client in case it will receive reply from all the replicas.
- If FE doesn't receive reply from any of the replica within reasonable time, it will notify the other replicas about the crash failure of the particular replica.
- If FE receives three consecutive incorrect result from the same replica it will send a message about a software error at a particular replica to all the remaining replicas, so that they can take actions to restart the faulty replica and maintain the data consistency.

Sequencer

- It functions as a mediator in between Replicas and the front end.
- It will receive the request from the front end and forwards it to the replica manager using UDP unicast.
- It will assign unique sequence id to each request before forwarding it to replica manager, thus attaining total order.

Replica

- Each Replica is a Java implementation, which contains the implementation of the operations that client wants to perform, it will receive UDP multicast messages from the Replica Manager and send the reply containing the results back to the FE.
- Each Replica will have its own copy of data and a structure, which will handle it.
- Each replica will maintain a message queue of the operations performed, so that, when any of the replica crashes it can be used to maintain data consistency among all replicas.

Replica Manager

- Receives request containing the sequence number from the Sequencer and multicasts it to the replicas.
- It also receives Software Bug message from FE and maintains count of number of consecutive software bugs reported and when the count reaches 3, it will send a message to the faulty replica to inform it about the bug.

Test Cases

Book Event and Cancel Event:

```
Client [Java Application] C:\Program Files\Java\jre\bin\javaw.exe (Aug 7, 2019, 3:37:39 PM)
Enter ID:
MTLC1234
MTL C
1.Book Event
2.Get Booking Schedule
3.Cancel Event
4.Swap Event
5.Logout
Enter your Choice=
1
Enter Event ID:MTLA010119
Enter Event Type:SEM
BOOKING IS SUCCESSFUL.
1.Book Event
2.Get Booking Schedule
3.Cancel Event
4.Swap Event
5.Logout
Enter your Choice=
1
Enter Event ID:MTLM020119
Enter Event Type:CON
BOOKING IS SUCCESSFUL.
1.Book Event
2.Get Booking Schedule
3.Cancel Event
4.Swap Event
5.Logout
Enter your Choice=
3
Enter Event ID to drop:
MTLA010119
Enter Event Type:SEM
EVENT HAS BEEN CANCELED.
1.Book Event
2.Get Booking Schedule
3.Cancel Event
4.Swap Event
5.Logout
Enter your Choice=
```

Crash Server

```
{C={MTLM020119=9, MTLE020119=6, MTLA020119=4}, S={MTLM010119=5, MTLA010119=7, MTLE010119=8}, T={MTLM030119=3, MTLE030119=7, MTLA030119=6}}
{MTLC1234={}}
EVENT HAS BEEN CANCELED.
Receiving Replica Message and sending response to FE

3,MTLC1234,BOOK,MTLE030119,TS
Server Crashed

Receiving Replica Message and sending response to FE
Message From FE:Crash
Queue : [0,MTLC1234,BOOK,MTLA010119,SEM, 1,MTLC1234,BOOK,MTLM020119,CON, 2,MTLC1234,CANCEL,MTLA010119,SEM, 3,MTLC1234,BOOK,MTLE030119,TS]
{C={MTLM020119=10, MTLE020119=6, MTLA020119=4}, S={MTLM010119=5, MTLA010119=6, MTLE010119=8}, T={MTLM030119=3, MTLE030119=7, MTLA030119=6}}
{MTLC1234={MTLA010119}}
{C={MTLM020119=9, MTLE020119=6, MTLA020119=4}, S={MTLM010119=5, MTLA010119=6, MTLE010119=8}, T={MTLM030119=3, MTLE030119=7, MTLA030119=6}}
{MTLC1234={MTLM020119}}
{C={MTLM020119=9, MTLE020119=6, MTLA020119=4}, S={MTLM010119=5, MTLA010119=7, MTLE010119=8}, T={MTLM030119=3, MTLE030119=7, MTLA030119=6}}
{MTLC1234={}}
{C={MTLM020119=9, MTLE020119=6, MTLA020119=4}, S={MTLM010119=5, MTLA010119=7, MTLE010119=8}, T={MTLM030119=3, MTLE030119=6, MTLA030119=6}}
{MTLC1234={MTLE030119}}
Data after Crash

Conferences
Event: MTLM020119 Capacity: 9
Event: MTLE020119 Capacity: 6
Event: MTLA020119 Capacity: 4

Seminars
Event: MTLM010119 Capacity: 5
Event: MTLA010119 Capacity: 7
Event: MTLE010119 Capacity: 8

TradeShows
Event: MTLM030119 Capacity: 3
Event: MTLE030119 Capacity: 6
Event: MTLA030119 Capacity: 6

MTL Sem
MTLC1234-->[]

MTL Con
MTLC1234-->[MTLM020119]

MTL TS
MTLC1234-->[MTLE030119]
```

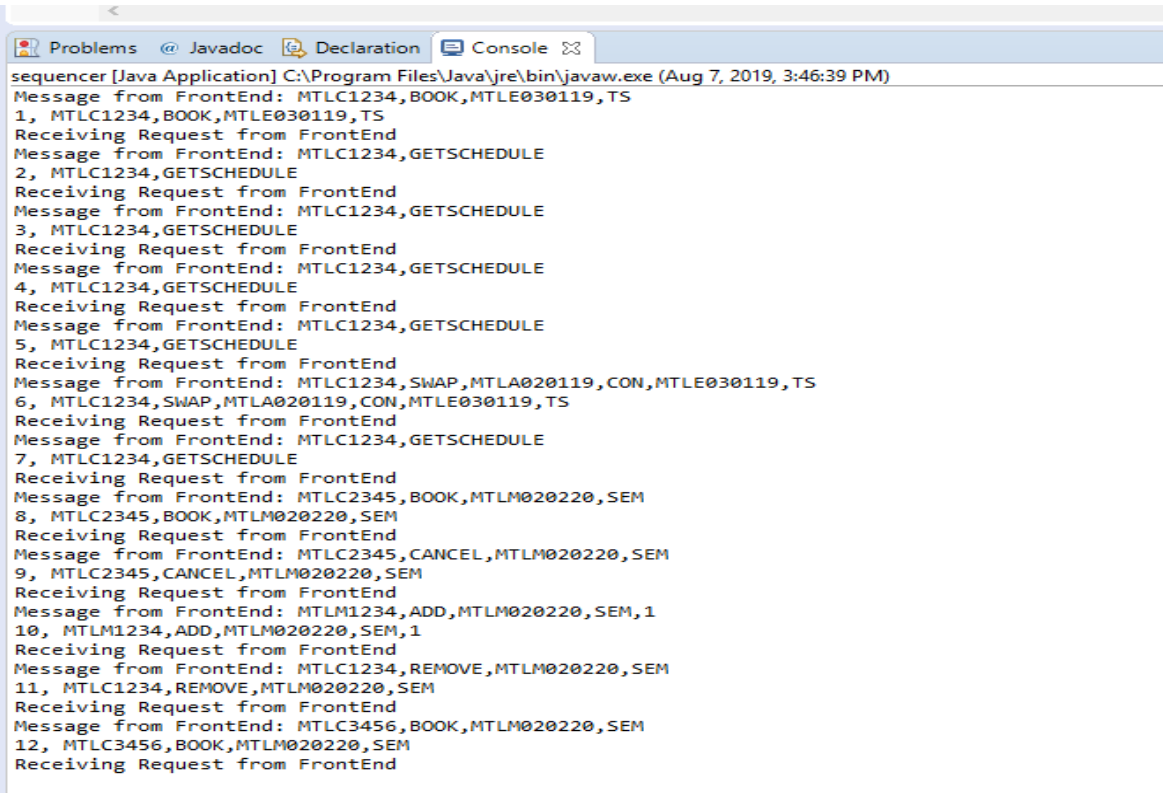
Bug Detection

```
Inside GetBookingSchedule
binding started
Binding close
3
3
3
Max Delay:13
+++++
1
EVENTS BOOKED :
Conferences: []
Seminars: [MTLA010119]
TradeShows: [MTLE030119]
EVENTS BOOKED :
Conferences: []
Seminars: [MTLA010119]
TradeShows: [MTLE030119]
Software bug in krisha
Inside GetBookingSchedule
binding started
Binding close
5
5
5
Max Delay:15
+++++
EVENTS BOOKED :
Conferences: []
Seminars: [MTLA010119]
TradeShows: [MTLE030119]
EVENTS BOOKED :
Conferences: []
Seminars: [MTLA010119]
TradeShows: [MTLE030119]
EVENTS BOOKED :
Conferences: []
Seminars: [MTLA010119]
TradeShows: [MTLE030119]
```

Multithreading

```
Enter your Choice=
5
Enter ID:
MTLM1234
MTL M
1.Add Event
2.Remove Event
3.List Event Availability
4.Book Event
5.Get Booking Schedule
6.Cancel Event
7.Swap Event
8.MultiThreading task
9.Logout
Enter Choice:
8
1.Add Event
2.Remove Event
3.List Event Availability
4.Book Event
5.Get Booking Schedule
6.Cancel Event
7.Swap Event
8.MultiThreading task
9.Logout
Enter Choice:
TH3: ENTER THE VALID EVENTID.
TH4: CUSTOMER NOT REGISTERED FOR THIS EVENT.
TH1: EVENT ADDED SUCCESSFULLY.
TH5: EVENT HAS BEEN REMOVED.
TH2: ENTER THE VALID EVENTID.
```

Sequencer Output:



The screenshot shows a Java IDE window with the 'Console' tab selected. The title bar indicates the application is 'sequencer [Java Application]' running at 'C:\Program Files\Java\jre\bin\javaw.exe (Aug 7, 2019, 3:46:39 PM)'. The console output displays a series of messages and requests from a 'FrontEnd' to the application, numbered 1 through 12. The messages include commands like 'GETSCHEDULE', 'SWAP', 'CANCEL', 'ADD', and 'REMOVE' along with various identifiers and parameters.

```
sequencer [Java Application] C:\Program Files\Java\jre\bin\javaw.exe (Aug 7, 2019, 3:46:39 PM)
Message from FrontEnd: MTLC1234,BOOK,MTLE030119,TS
1, MTLC1234,BOOK,MTLE030119,TS
Receiving Request from FrontEnd
Message from FrontEnd: MTLC1234,GETSCHEDULE
2, MTLC1234,GETSCHEDULE
Receiving Request from FrontEnd
Message from FrontEnd: MTLC1234,GETSCHEDULE
3, MTLC1234,GETSCHEDULE
Receiving Request from FrontEnd
Message from FrontEnd: MTLC1234,GETSCHEDULE
4, MTLC1234,GETSCHEDULE
Receiving Request from FrontEnd
Message from FrontEnd: MTLC1234,GETSCHEDULE
5, MTLC1234,GETSCHEDULE
Receiving Request from FrontEnd
Message from FrontEnd: MTLC1234,SWAP,MTLA020119,CON,MTLE030119,TS
6, MTLC1234,SWAP,MTLA020119,CON,MTLE030119,TS
Receiving Request from FrontEnd
Message from FrontEnd: MTLC1234,GETSCHEDULE
7, MTLC1234,GETSCHEDULE
Receiving Request from FrontEnd
Message from FrontEnd: MTLC2345,BOOK,MTLM020220,SEM
8, MTLC2345,BOOK,MTLM020220,SEM
Receiving Request from FrontEnd
Message from FrontEnd: MTLC2345,CANCEL,MTLM020220,SEM
9, MTLC2345,CANCEL,MTLM020220,SEM
Receiving Request from FrontEnd
Message from FrontEnd: MTLM1234,ADD,MTLM020220,SEM,1
10, MTLM1234,ADD,MTLM020220,SEM,1
Receiving Request from FrontEnd
Message from FrontEnd: MTLC1234,REMOVE,MTLM020220,SEM
11, MTLC1234,REMOVE,MTLM020220,SEM
Receiving Request from FrontEnd
Message from FrontEnd: MTLC3456,BOOK,MTLM020220,SEM
12, MTLC3456,BOOK,MTLM020220,SEM
Receiving Request from FrontEnd
```