

**Distributed Flight Reservation System**

**Submitted to:**

**Instructor: Rajagopalan Jayakumar**

**Department of Computer Science & Software Engineering**

**Concordia University, Montreal, QC**

**In Fulfillment**

**of the Requirements**

**of COMP 6231**

**Design Documentation**

**Submitted by:**

**Varun Pattiah Sankaralingam - 27679009**

**Shreyas Vasudevareddy Movva - 40027704**

**Ravinder Pal Singh - 40021206**

**Master in Applied Computer Science**

**Concordia University**

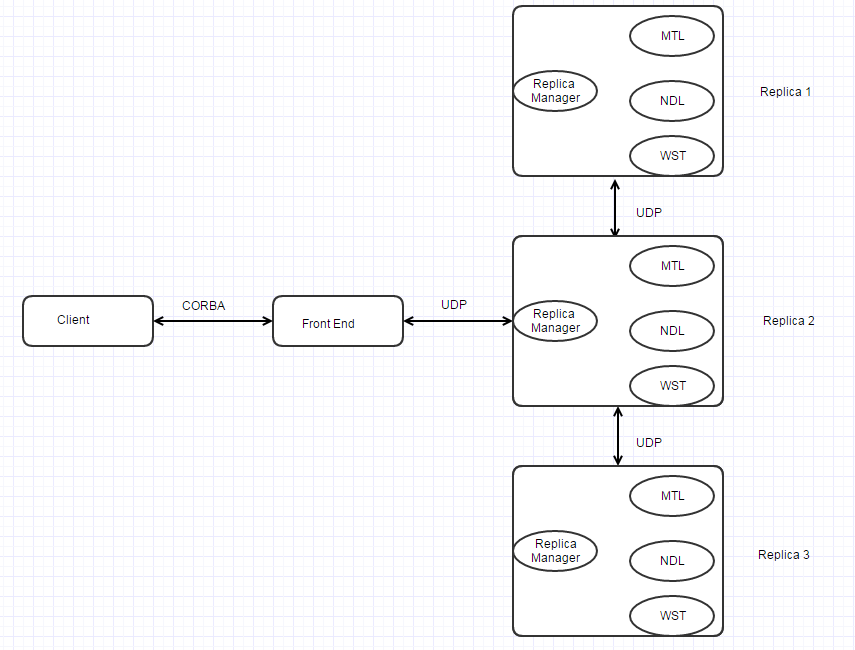
# Introduction

This document contains the detailed description how High availability is achieved in implementation of Distributed Flight Reservation System (DFRS) Using CORBA.

This document has information on design of highly available DFRS system with total order implemented through FIFO. It contains the detailed explanation about various Replica, Front End, Sequencer and Replica Manager to achieve high availability.

It also discusses about the data structure used for implementing Distributed Flight reservation system and the problem faced while implementing the whole system. Also it contains detailed description of logs maintained for both Passenger and Manager. At the same time Logs at server also maintained to keep track of each action happened at server.

# 2. System Architecture

****

**DFRS system architecture**

Highly Available CORBA Distributed Flight Reservation System is implemented over the CORBA project implemented as part of assignment2. We will implemented a Front End module which would accept the Client Request through CORBA. And Front End will send the request to Replica Manager after assigning a sequence number to each request to maintain the total ordering. We are implementing FIFO to achieve the total ordering in the system. The FE is responsible for broadcasting the request with using the User Datagram Protocol, and that request is broadcasted from the FE to the Leader. The Leader processes these requests iteratively by using FIFO mechanism

Front End interact with Replica Manager through UDP message. One of the Replica Manager is selected as Lead and the Request handling is implemented here in order to communicate the request to other replica Manager through UDP message. All Replica Manager invoke the replica call through CORBA.

In order to process each request, the Leader multicasts each request to the other two Replicas which process in their local servers and send the reply back to the Leader. After then, the Leader compares the results from all the Replicas and sends the correct result to the FE.

Since the replicated servers are usually on a local area network, they communicate using the unreliable UDP protocol. However, the communication among them should be reliable and FIFO. At a same time, a low-level protocol is also possible since the developers know the communication protocol and data representation strategy about all the details concerning the module implementation. Using a low-level protocol allows alienating from the overhead linked to the generalizations required for the higher-level protocols and to design an ad-hoc communication.

## Clients

The DFRS System include Passenger and Manager Request. Passenger can book a Flight and Manager could perform functionalities like getBookedFlightCount, EditFlightRecord and transfer Record from one city to other.

## Front End

The Front End (FE) is the connection between the clients and the Replicas. When a client needs a service from the DFRS, it sends a request over the CORBA to the FE. The FE then sends the request to the Lead Replica Manager using a UDP connection. The Request Handler implemented in the Leader Replica communicates with other Replicas and sends the request to them. Each Replica computes the result and sends it back to the Lead. The Request Handler receives three results including the result of the Leader Replica from each Replica and compares them. If the results are equal then it is sent to the FE. Assuming that only one Replica may produce a wrong result at a time, to tolerate this failure, the FE takes what it considers as the most returned result and sends it back to the client. The FE guarantees the transparency of clients by using the CORBA architecture. In addition, multithreading is used as a way for Replicas and replies to synchronize with each other.

As the FE is implemented as a CORBA object and managed by the CORBA engine, the server is automatically multithreaded to communicate with several clients in parallel. The FE may process several client requests in parallel and broadcast multiple requests simultaneously. In order to send multiple requests, a sequence number is generated by the FE. This number is then attached to the request so as to keep track of it and deal with concurrency issues.

## Request Handler (Lead Replica Manager)

The Request Handler is the component that translates and manages the request broadcasted from the FE. Request Handler module is implemented in the same host where the Leader is. The module receives the requests via UDP/IP from the FE and uses FIFO technique to process them iteratively. As several of these requests may have been sent in a random order from the FE, the Request Handler is responsible for arranging the received requests.

The received requests are processed and ordered based sequence number assigned to it by the FE before sending. The Leader will then multicast each request to the other two Replicas and also processes the request locally. After getting the results from other Replicas, the Leader will compare all three results and send the most returned result to the FE. This is done for each request received iteratively. Moreover, if any of the Replicas generates a wrong result, the Request Handler will inform the RM.

## FIFO Scheduler

To ensure a correct response (response for the correct method invoked) is sent back to client, scheduling request at the leader is important and this task is performed by scheduling module at the leader processes, which schedules client request in FIFO order for the requests and sends the replies to the clients in the same order.