



Project:

Software Failure Tolerant and/or Highly Available Distributed Course Registration System

Name: Kishan M Bhimani

Student ID: 40081118

Name: Rajan Shah

Student ID: 40081117

Name: Parth Patel

Student ID: 40081116

Table of Contents

Introduction.....	3
Technology Used	4
CORBA.....	4
User Datagram Protocol (UDP)/ Internet Protocol (IP)	4
System Specifications	5
Important and Difficult Parts.....	6
Important Parts	6
Difficult Parts	6
Important Data Structures	7
Design Architecture & Data Flow	8
Front End (FE)	8
Sequencer	9
Replica	9
Replica Manager	9
Test Cases	10
Multi-Threading.....	10
Crash Result.....	11
Software bug	12
Sequencer Output	13

Introduction

The Distributed Course Registration System (DCRS) is a software using which students can enroll/ register for subjects of, either their own or other departments for different semester with some constraints. The constraint being, no student can enroll in more than 3 subjects (courses) per semester and not more than 2 subjects per semester from other department. The students can enroll and drop the subjects according to their wish, view their schedule for the entire year and also swap subjects within a semester. The advisor, on the other hand, can add, and remove, the courses for their own department and also get list of all the subjects for a specific semester from other departments to view to the students. He/ she can also enroll, and drop students from any course, for any semester, and from any department, i.e. the advisor can perform all the functions of the students. The students can enroll to a course only if the course is not full for that semester and if he/ she has not already enrolled for that subject in any other semester.

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 students and the advisors – the clients, can communicate and invoke their respective operations i.e. perform the RPC, respective to their departments – 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 department, for the clients of other department, for example, students of one department can only access data and invoke operations, respective to their own department, but for them to enroll in subject of department, other than their own, they need to send request to their own department and then their department can request, on their behalf to the other department to perform such operations. Such operations can be performed by one server (department) 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 Students and the Advisors, the clients, are identified by unique identifiers studentID and advisorID, which is constructed from the acronym of their department and a 4-digit number (e.g. COMPA1111 for an advisor and COMPS1111 for a student).

Departments: There are 3 departments, viz: Computer Science (COMP), Software Engineering (SOEN) and Information Security (INSE), which are implemented as 3 different servers in the system, each having their own pre-authorized set of advisors who can only perform the operations, and unauthorized advisors will be checked and refrained from performing the operations. Only the students of the department can access that department's server directly, whereas, the students of other departments need to make a request to their own department, which in turn requests the other department 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.

Students: One of the client of the system, the students, identified by their unique identifiers, can only access the data and invoke functions respective to their own department using JAVA RMI. The students can enroll in maximum of 3 courses per semester and a maximum of 2 subjects, included in the maximum 3, from other departments, i.e. it is compulsory for the students to enroll in at least 1 course from their own department, if he/she wishes to enroll for 3 courses in specific semester. They can drop the courses that they are already enrolled in for any semester at any time. They cannot enroll for the same course in any other semester, once they have enrolled in one semester. They can also view their entire schedule, i.e. the courses they are enrolled in for the entire year. They can swap the subjects within a semester given that the above mentioned constraints hold. They can swap the subjects based on their wish following the constraints.

Advisors: The other client, the advisors, can add, and remove, the courses for their own department and also get list of all the subjects for a specific semester from other departments to view to the students, and can also enroll, and drop students from any course, for any semester, and from any department, i.e. the advisor can perform all the functions of the students, given the constraints imposed on enrolling and dropping are satisfied. The advisor, therefore, can perform all the functions there are in total in the entire system.

Important and Difficult Parts

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 an advisor or a student and thus assigning the operations allowed to the client. Also, checking on the server side, the courseId, to know whether the client is trying to access the course of the department it belongs to or of some other department and thus proceeding with either executing the methods on the same server or starting the inter-server 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.

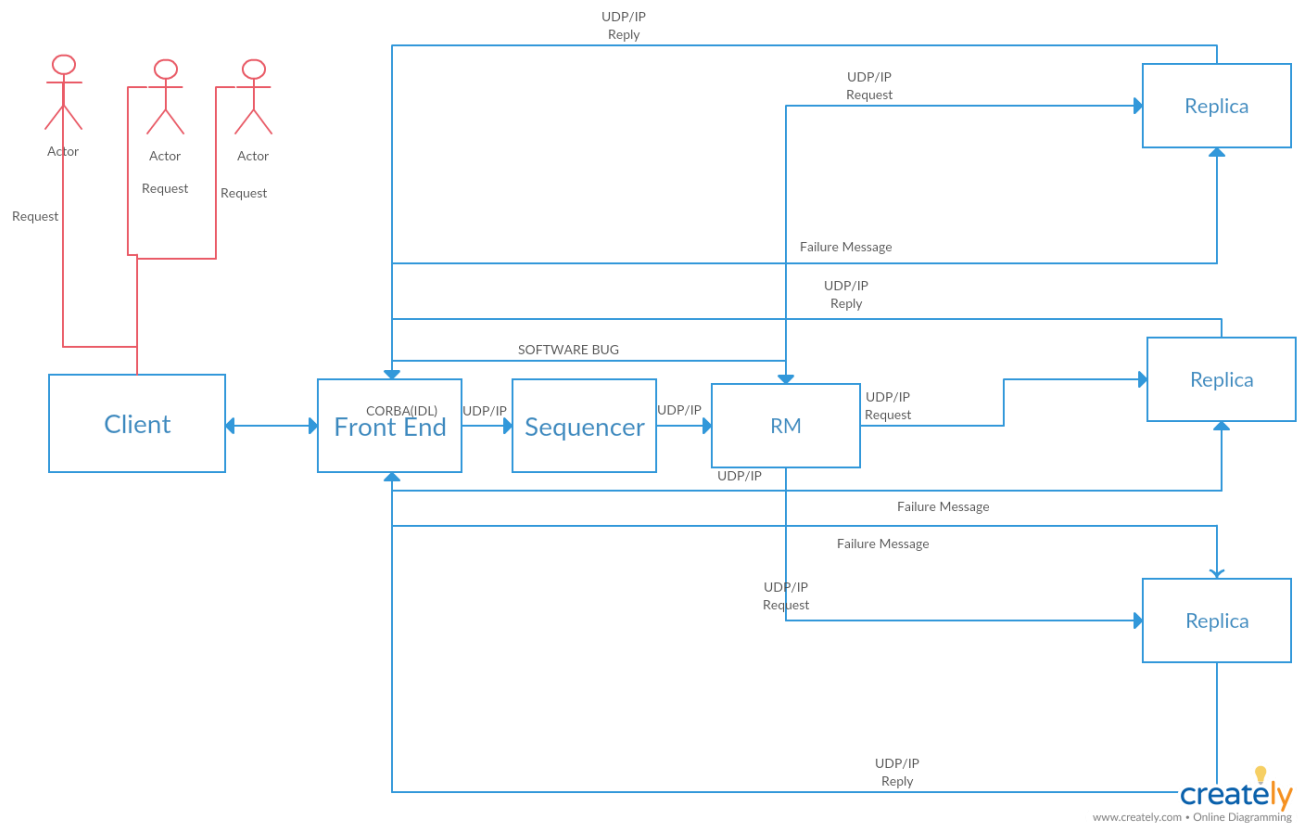
Important Data Structures

```
reply= new HashMap<String,String>();
reply.put("AC1", "Course Added Successfully");
reply.put("AC2", "Course Already Exists");
reply.put("AC3", "Unauthorized Advisor");
reply.put("RC1", "Course Removed Successfully");
reply.put("RC2", "Course Does not Exist");
reply.put("RC3", "Unauthorized Advisor");
reply.put("EC1", "Course Enrolled Successfully");
reply.put("EC2", "Course Not Available in this Semester");
reply.put("EC3", "You are Already Enrolled in Max Number of Subjects for this Semester");
reply.put("EC4", "Course is Full");
reply.put("EC5", "You are already Enrolled in this Course");
reply.put("EC7", "You are Already Enrolled in Max Number of Subjects from other Departments");
reply.put("DC1", "Course Dropped Successfully");
reply.put("DC2", "You are not Enrolled in this course for this Semester");
reply.put("SS1", "Swapping Successfully");
```

Message received from each server is mapped based on the keys and final result is sent to the client.

`reply` is a HashMap with **key as reply from the replica** and **value is corresponding message** for the client.

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

Multi-Threading

```
Enter ID:
COMPS6001
1.Enroll for a course
2.Drop a course
3.Swap Course
4.Class Schedule
5.Logout
Enter Choice
t1Course Enrolled Successfully
t3Course Enrolled Successfully
t4Course Removed Successfully
t2Course Not Available in this Semester
```

Front end ready and waiting ...

```
Inside enroll
22
30
37
Max Delay:40
+++++
Parth:-EC1
Rajan:-EC1
Kishan:-EC1
Inside enroll
12
25
35
Max Delay:35
+++++
Parth:-EC1
Rajan:-EC1
Kishan:-EC1
Inside Remove Course
6
13
13
Max Delay:23
+++++
Parth:-RC1
Rajan:-RC1
Kishan:-RC1
Inside enroll
12
12
17
Max Delay:27
+++++
Parth:-EC2
Rajan:-EC2
Kishan:-EC2
```

Crash Result

Crashed replica output

+++++

Parth:-EC2

Rajan:-EC2

Kishan:-EC2

Inside enroll

3

4

Timed Out

1

Parth:-rajan:-

Parth:-EC1rajan:-EC1

Inside if: Parth:-EC1rajan:-EC1

5,COMPS6001,Enrol,COMP7002,Fall

Server Crashed

Receiving Replica Message and sending response to FE
Message From FE:Crash

Request sent to SOEN

Request sent to INSE

Data after Crash

_____Fall_____

Course: COMP7003 Capacity: 2

Course: COMP7002 Capacity: 2

_____Winter_____

Course: COMP7001 Capacity: 3

Course: COMP7005 Capacity: 3

Course: COMP7004 Capacity: 3

_____Summer_____

Course: COMP7002 Capacity: 3

Course: COMP7007 Capacity: 3

Course: COMP7006 Capacity: 3

COMP Fall

COMPS6001-->[COMP7003, COMP7002]

COMP Winter

COMP Summer

Receiving crash Message from FE

Software bug

```
+++++
Parth:-Course Semester
[COMP7003, COMP7002] Fall
[] Winter
[] Summer
Rajan:-Bug
Kishan:-Course Semester
[COMP7003, COMP7002] Fall
[] Winter
[] Summer
Inside classSchedule
3
3
3
Max Delay:13
+++++
Parth:-Course Semester
[COMP7003, COMP7002] Fall
[] Winter
[] Summer
Rajan:-Bug
Kishan:-Course Semester
[COMP7003, COMP7002] Fall
[] Winter
[] Summer
Inside classSchedule
3
3
4
Max Delay:14
+++++
Parth:-Course Semester
[COMP7003, COMP7002] Fall
[] Winter
[] Summer
Rajan:-Course Semester
[COMP7003, COMP7002] Fall
[] Winter
[] Summer
Kishan:-Course Semester
[COMP7003, COMP7002] Fall
[] Winter
[] Summer
```

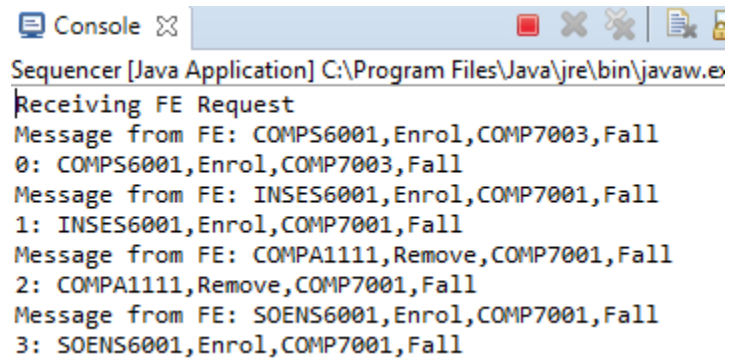
```
MSG from Replica Manager: 6,COMPS6001,Get
unique id:6 ID:COMPS6001 Method:Get
Inside GET METHOD
0
Get Schedule
Sending Bug Message
Receiving Replica Message and sending response to FE
```

```
MSG from Replica Manager: 7,COMPS6001,Get
unique id:7 ID:COMPS6001 Method:Get
Inside GET METHOD
0
Get Schedule
Sending Bug Message
Receiving Replica Message and sending response to FE
```

```
MSG from Replica Manager: 8,COMPS6001,Get
unique id:8 ID:COMPS6001 Method:Get
Inside GET METHOD
0
Get Schedule
Sending Bug Message
Receiving Replica Message and sending response to FE
Messsage from RM: Bug
Receiving SW bug Message from RM
```

```
MSG from Replica Manager: 9,COMPS6001,Get
unique id:9 ID:COMPS6001 Method:Get
Inside GET METHOD
Get Schedule
Course Semester
[COMP7003, COMP7002] Fall
[] Winter
[] Summer
Receiving Replica Message and sending response to FE
```

Sequencer Output



```
Sequencer [Java Application] C:\Program Files\Java\jre\bin\javaw.exe
Receiving FE Request
Message from FE: COMPS6001,Enrol,COMP7003,Fall
0: COMPS6001,Enrol,COMP7003,Fall
Message from FE: INSES6001,Enrol,COMP7001,Fall
1: INSES6001,Enrol,COMP7001,Fall
Message from FE: COMPA1111,Remove,COMP7001,Fall
2: COMPA1111,Remove,COMP7001,Fall
Message from FE: SOENS6001,Enrol,COMP7001,Fall
3: SOENS6001,Enrol,COMP7001,Fall
```