

**Department of Computer Science and Software Engineering**

**Comp 6321:**

**Distributed System**

Instructor: Prof. R. Jayakumar

Teaching Assistant: Alexandre Hudon

**Project**

**Software Failure Tolerant CORBA Distributed Player Status System**

|  |  |
| --- | --- |
| **Name** | **Student ID** |
| **Soyoung Kim** | **7096941** |
| **Meet Anadkat** | **7122829** |
| **Chintan Sarvaiya** | **7148879** |

**TABLE OF CONTENTS**

1 INTRODUCTION 3

2 HIGH LEVEL SYSTEM DESIGN 4

2.1 Theories description 4

2.2 DPSS Use Case Model 7

2.3 System Architecture 8

2.3.1 Clients 9

2.3.2 Front End 9

2.3.3 Replica Manager 10

2.3.4 Replica Handler 10

2.3.5 Sequence Diagrams 11

3 ASSUMPTIONS 12

4 OVERVIEW OF IMPLEMENTATION 12

5 TEST CASES SCENRIOS 13

6 GROUP WORK ASSIGNMENT 13

1. **INTRODUCTION**

The Distributed Player Status System (DPSS) is a simple system that is used by two clients that are player client and an administrator client. The application consists of three server programs corresponding to North-America (NA), Europe (EU) and Asia (AS) server. The system for *player client* has four operations used by the users to create account, sign in, signs out, and transfer account provided by *game server*. For *admin client*, there are two operations under the valid credentials. One is *getPlayerStatus* that the server associated with this administrator determined by the *IP Address* attempts to concurrently count the number of online players and offline players in the other geo-locations using UDP/IP sockets and returns the result to the administrator. The other is *suspendAccount, which* deletes the account of the username specified by the administrator.

For this project, the CORBA implementation of the DPSS is enhanced to tolerate a single software failure by applying the active Replication model. The active model of Replication consists of number of Replicas that act as state machines. In this model, the same request of client is processed at every Replica and all of them should be in the same state with similar consistent data after each operation. In order to make all the Replicas to receive the same sequence of operations, an atomic multicast protocol is used.

The actively Replicated DPSS server system should have at least three Replicas each running a different implementation in different hosts on the network. Three significant subsystems should be added to the system: The Front End, the Replica Manager and the Request Handler (FIFO) which is implemented in the Leader Replica. The entire system is deployed and run over a Local Area Network (LAN) where the server Replicas, the Front End, the Request Handler, and the Replica Manager communicate among them using User Datagram Protocol.

In the following section, the high level system design will be discussed where it explains the system functionality, architecture and expected behaviour of each component. In section 2 and 3, the design of the system specifications is covered. Also, in section 4, test cases scenarios are shown, which are the success and failure results from each 6 operations, concurrency, and the failure detection case. In the last section, the individual task of all of the members is mentioned.

1. **HIGL LEVEL SYSTEM DESIGN**
   1. **Theories Description**

Theories for protocols and algorithms used in our DPSS system as below:

**CORBA:** The Common Object Request Broker Architecture (CORBA) is a standard framework allowing software objects to communicate with one another, no matter where they are located or who has designed them. Two major components of CORBA are Object Request Broker (ORB) and Interface Definition Language (IDL). The CORBA ORB essentially enables communication between clients and remote server objects. The IDL is a declarative language that describes the interfaces to server objects. CORBA objects can be written in any standard programming language and exist on any computing platform supported by a CORBA vendor.

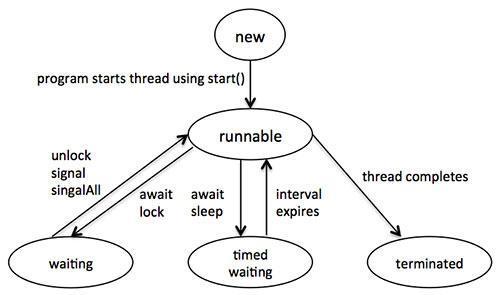
The interface definition specifies which member functions are available to a client without making any assumptions about the implementation of the object. To invoke member functions on a CORBA object, a client needs only the object’s IDL definition. The client does not need to know the language used to implement the object, the location of the object, or the operating system on which the object resides. CORBA objects use the ORB as an intermediary to facilitate network communication. A CORBA ORB delivers requests to objects and returns any response to clients making the requests. The key feature of ORB is the transparency of its facilitation of client/server communication. When a client invokes a member’s function on a CORBA object via the IDL interface, the ORB intercepts the function call and redirects the function call across the network to the target object. The ORB then collects the results and returns them to the client.

In this project, the CORBA architecture is used between the Client and the Front End. The client invoke mainly 6 remote methods (*createPlayerAccount, SignInPlayer, SignOutPlayer, getPlayerStatus, transferAccount, suspendAccount)* that reside on the FE, however the functionality of these methods would be defined on the replica servers. The purpose of the CORBA methods in the implementation class on the Frond End is to gather the details of the Player/Administrator, perform marshaling in order to convert these details into external data representation and send a UDP request message to the Leading server. All these methods on FE would be declared in an interface which in turn is defined in the IDL.

### Concurrency: Java provides built-in support for multithreaded programming. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution. A multithreading is a specialized form of multitasking. Multithreading requires less overhead than multitasking processing. Multithreading enables you to write very efficient programs that make maximum use of the CPU, because idle time can be kept to a minimum.

Life Cycle of a Thread:

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. Following diagram shows complete life cycle of a thread.



**Fig 1. Various stages of Life Cycle of a Thread**

In this project, it is being used to test how the Frond End would handle multiple client requests simultaneously. Multiple clients can login from Client program and they should be able to interact with the Server replicas without any sort of internal interference. In order to maximize concurrency, SYNCHRNOIZED block is being used on the server side. Also, on each server replica, 3 UDP servers, one for every geo-location, are listening on 3 different ports in separate threads.

**UDP**: UDP (User Datagram Protocol) is a communications protocol that offers a limited amount of service when messages are exchanged between computers in a network that uses the Internet Protocol (IP). UDP is an alternative to the Transmission Control Protocol ([TCP](http://searchnetworking.techtarget.com/definition/TCP)) and, together with IP, is sometimes referred to as UDP/IP. Like the Transmission Control Protocol, UDP uses the Internet Protocol to actually get a data unit (called a [datagram](http://searchnetworking.techtarget.com/definition/datagram)) from one computer to another. Unlike TCP, however, UDP does not provide the service of dividing a message into packets (datagrams) and reassembling it at the other end. Specifically, UDP doesn't provide sequencing of the packets that the data arrives in. This means that the application program that uses UDP must be able to make sure that the entire message has arrived and is in the right order. Network applications that want to save processing time because they have very small data units to exchange (and therefore very little message reassembling to do) may prefer UDP to TCP. The Trivial File Transfer Protocol (TFTP) uses UDP instead of TCP.

UDP provides two services not provided by the IP layer. It provides [port number](http://searchnetworking.techtarget.com/definition/port-number)s to help distinguish different user requests and, optionally, a [checksum](http://searchsecurity.techtarget.com/definition/checksum) capability to verify that the data arrived intact.

This connectionless UDP protocol is used by the Front End, the Replica Manager and the Replica servers. Every message exchanged between the Front End and the Leading server would be UDP messages. The inter-communication between all 3 replicas would also be using UDP protocol. As a single replica server would be running 3 different instances of North America, European and Asia server, their interaction would also take place with UDP messages. Replica Manager would also use UDP for its interaction with the Leading server.

* 1. **Use case Model**

### The system for *playerClient* has four operations: creating account, signing in or out, and transferring account. For *adminClient*, there are two operations: *getPlayerStatus* and *suspendAccount.* These operations are shown below in the following high-level use case model.

**Fig 2. Use-case model**

* 1. **System Architecture**

In order to design the Failure Tolerant Distributed Player Status System (FT-DPSS), the previous assignment, the CORBA IDL is utilized by modifying the original work keeping the design flexible, simple, and comprehensible. To access for each DPSS of all group members same services, all three DPSS should have the same interface. The three systems are ready to publish these services using the CORBA architecture. This functionality is also already tested as part of previous delivered works. These three systems are converted to the three Replicas of the DPSS required to build up the FT-DPSS. Their services would still be accessed using the CORBA architecture. Compared to the direct access between the system clients and the DPSS Replicas, in this FT-DPSS system, the client request would be managed by a set of components playing between the clients and the Replicas. These components coordinate the Replicas work in order to support fault tolerance.

The system clients communicate with the system Front End (FE) by using the CORBA architecture. 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. 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. During this process, the Leader also handles with the RM when any of the other two Replicas gives faulty results. When the number of the faulty results exceeds the maximum limit, the RM has to reinitialize the specific Replica.

The communication between the different modules such as the Replicas, the FE, and the RM is implemented over the UDP/IP protocol so as to optimize the process. 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.

**Fig 3. DPSS system architecture**

* + 1. **Clients**

The DPSS system consists of two clients which are Player and Admin. Player can perform four different operations: Create Account, Sign In, Sign Out, and Transfer Account. Compared to Player Clients, Admin has two functions: GetPlayer Status, Suspend Account. Clients send multiple requests, which are handled in parallel. From the client’s end, the whole system’s functionalities are encapsulated while the complexities are hidden from the end users.

* + 1. **Front End**

The Front End (FE) is the connection between the clients and the Replicas. When a client needs a service from the DPSS, it sends a request over the CORBA to the FE. The FE then sends the request to the Leader Replica 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 Leader. 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.

* + 1. **Replica Manager**

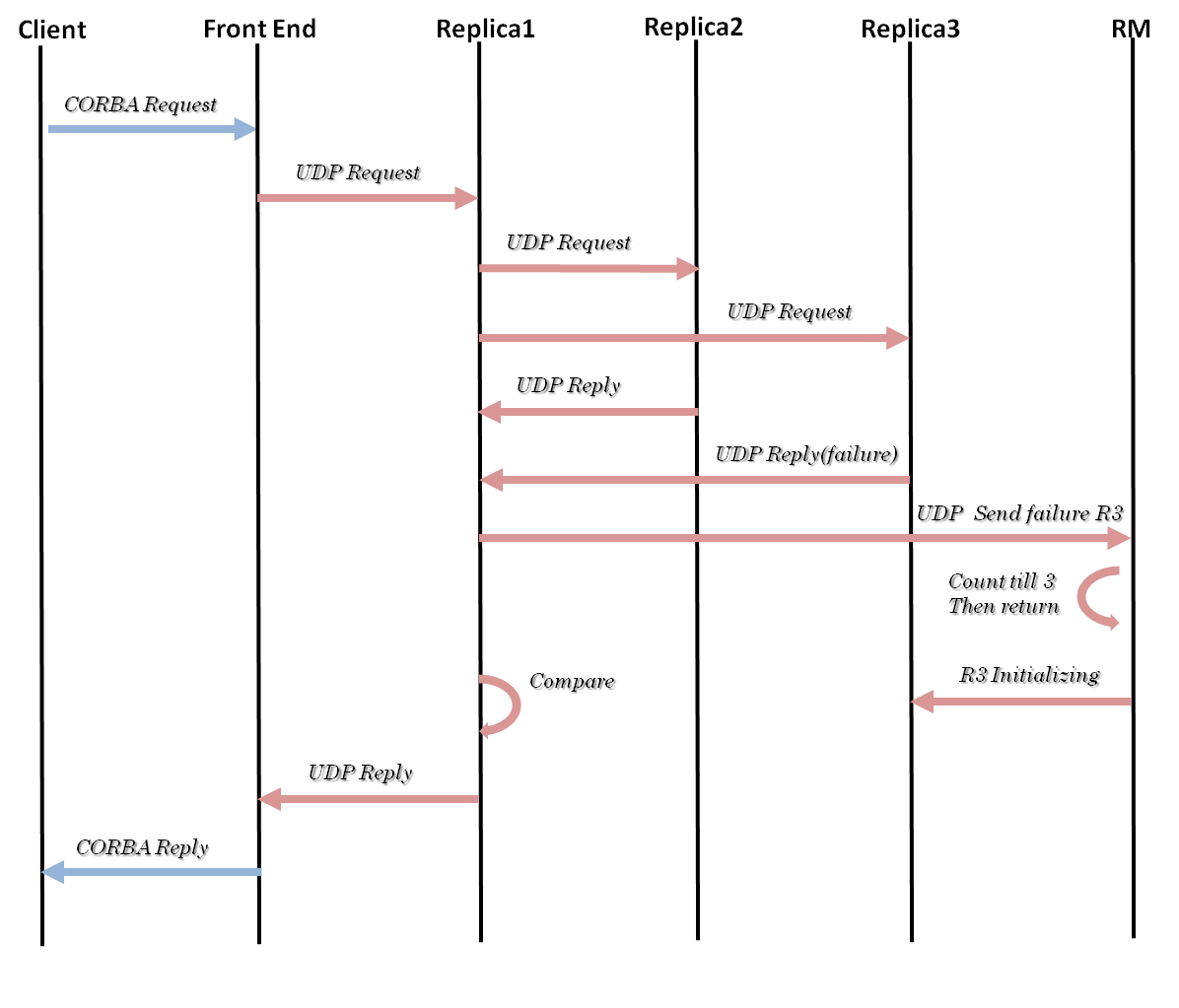
The Replica Manager (RM) is responsible for tracing the different active Replicas on the DPSS system. The main job of the RM is a communication with the Leader. For example, if any Replica generates a wrong result, then the Leader informs the RM the status of this Replica. The RM maintains a failure counter for each Replica. This counter is incremented for each wrong Replica result. If the counter reaches three failures, the Replica Manager will replace the corresponding replica with a new one.

* + 1. **Request Handler**

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.

* + 1. **Sequence Diagrams**

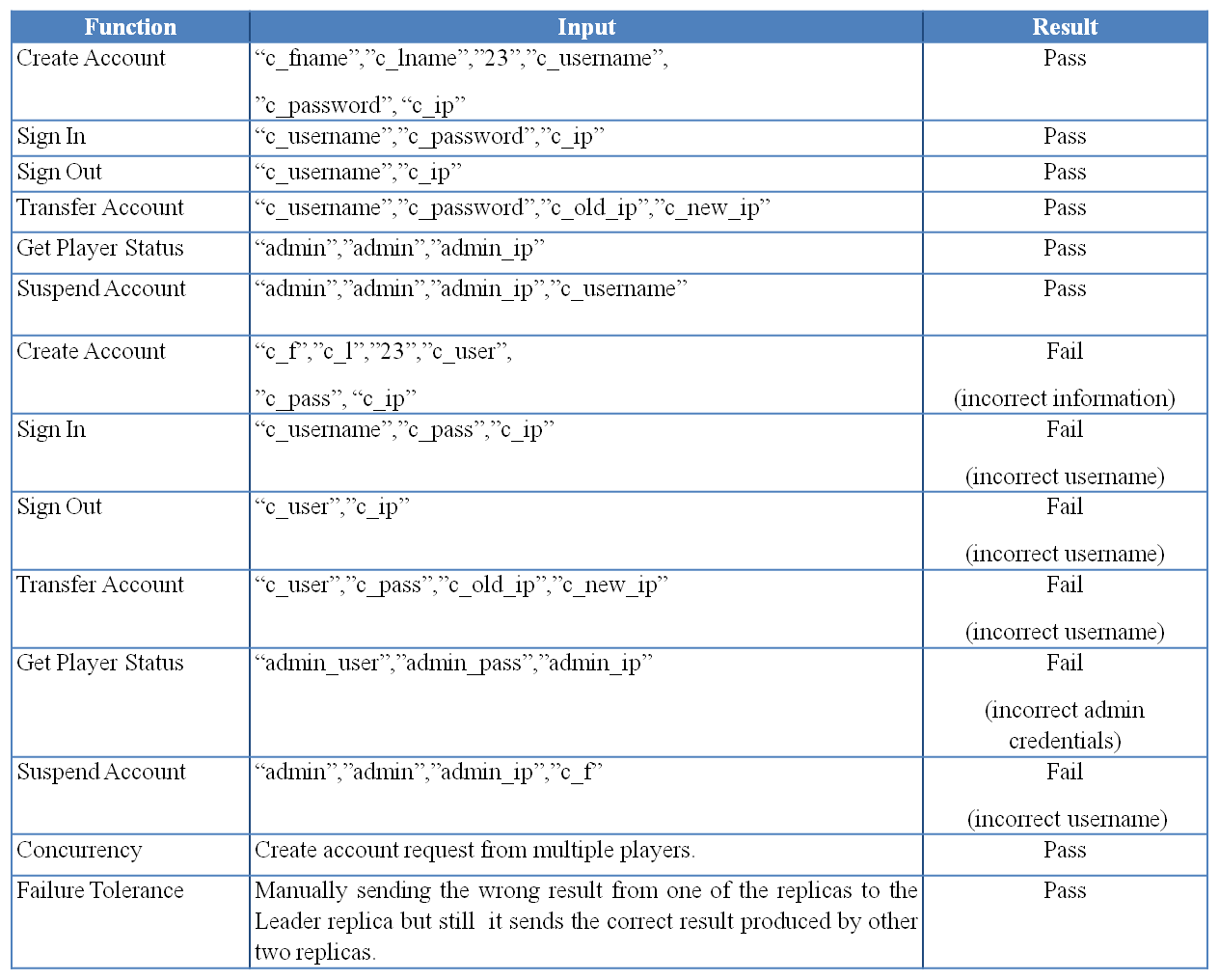
1. Typical success scenario
2. Typical Failure scenario



1. **ASSUMPTIONS**
2. The Replicas in this server subsystem are free from crash failure (software bug excluded).
3. Only one of 3 Replicas (except Leader) can produce an incorrect result.

(The number of Replica is 3=2N+1; N is the number of failure)

1. No failures could occur during the recovery of a faulty Replica.
2. The Front End, the Leader and the Replica Manager are all free from any failure.
3. **OVERVIEW OF IMPLEMENTATION**
4. Implementing Leader :
   1. Manage FIFO for all request coming from FE
   2. Multicast requests to replicas
   3. Process locally and compare with the replies of other two replicas
   4. Reply the maximum of two result to the Front End
   5. If any replica gives faulty result, send notification to Replica Manager.
5. Implementing Replica Manager
   1. Initiate Replicas except the Leader
   2. Listen to the Leader if any replica gives fault result
   3. I fault count reaches 3, then restart that particular replica (with or without data restore)
6. Implementing Front End
   1. Generate corba remote object
   2. Request any particular operation from client side via corba object
   3. For each client request, create a new thread & forward the request to the Leader by attaching the sequence number
   4. Accept the replies from the Leader.
7. **TEST CASE SCENARIOS**

Expected results are as follow

1. **Group Work Assignment**

|  |  |
| --- | --- |
| Component | Student |
| Front End | Soyoung Kim |
| Replica Manager | Meet Anadkat |
| FIFO | Chintan Sarvaiya |