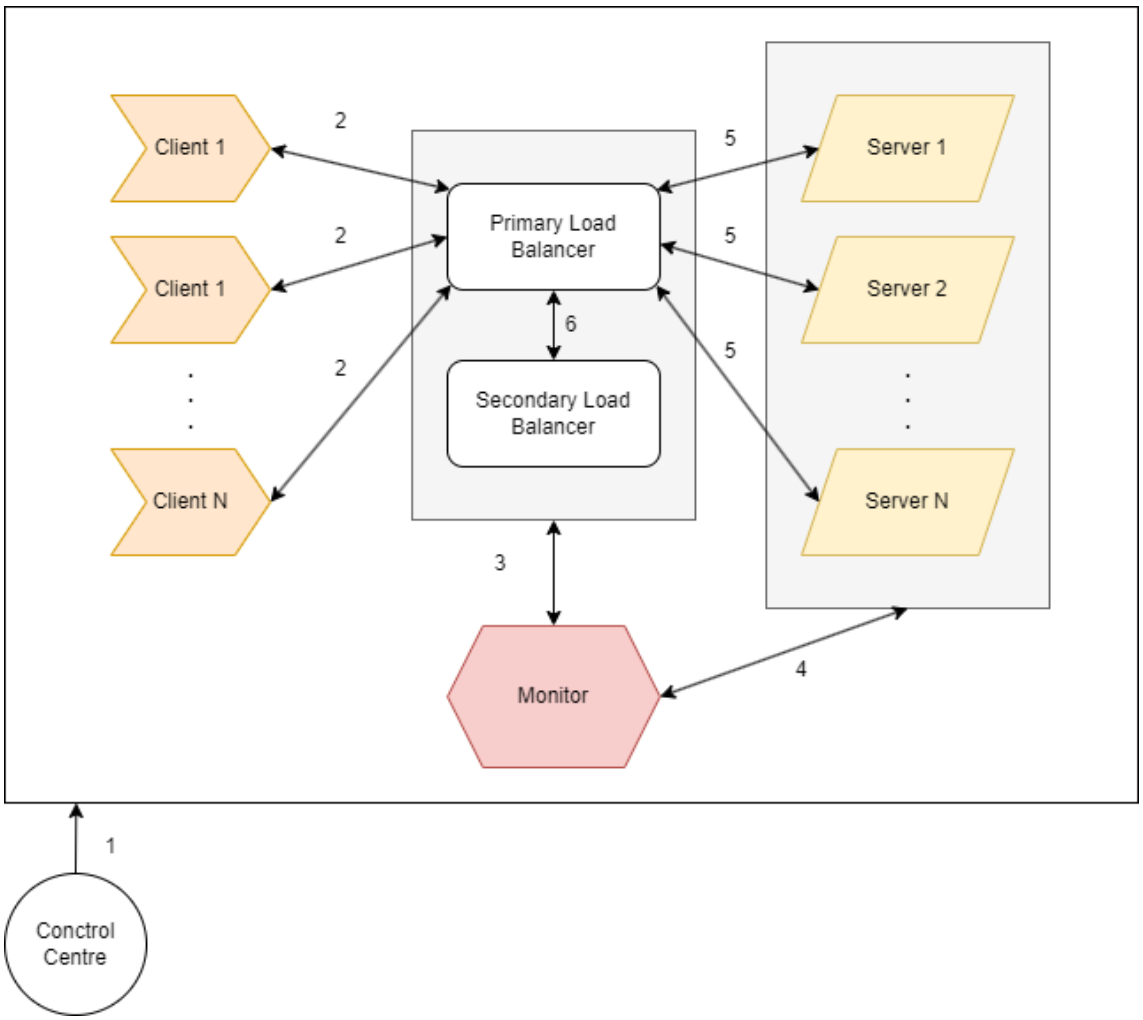
Report

Practical Assignment 3 (Quality Attributes)

Architectural Tactic



1 – Processes Management

2 – Client Communication

3 – Data Validation/Lookup

4 – Server Connection

5 – Service Communication

6 – Hearbeat Checkup

1 - Processes Management

- [Control Centre] Initialization of the intervening processes of the cluster: Client(s), Load Balancer(s), Server(s), Monitor.

```
public static Process executeProcess(String sourceDir, String classDir) throws IOException {
    //System.out.println("Working Directory = " + System.getProperty("user.dir"));

    Process process = Runtime.getRuntime()
        .exec(command:sourceDir);        // "javac -cp src src/LoadBalancer/Main.java -d build/classes/"

    process = Runtime.getRuntime()
        .exec(command:classDir);        // "java -cp build/classes/ LoadBalancer.Main"

    return process;
```

- [Control Centre] Stop intervening processes that may be opened.

```
private void destroyProcess(ArrayList<Process> processes, int removeIndex, javax.swing.JList list, String type) {
    processes.get(index: removeIndex).destroy();
    processes.remove(index: removeIndex);

    list.setModel(new javax.swing.AbstractListModel<String>() {
        public int getSize() { return processes.size(); }
        public String getElementAt(int i) { return type + i; }
    });
}
```

2 - Client Communication

- [Client] Request client id (will also serve as the client service reply handler port).

```
public void requestClientId() {
    while (true) {
        // connect to the load balancer
        try (Socket serviceSocket = new Socket(host:this.hostname, port:this.loadBalancerPort)) {
            System.out.println("SERVICE CLIENT SOCKET=" + serviceSocket);

            ObjectOutputStream out = new ObjectOutputStream(out:serviceSocket.getOutputStream());
            ObjectInputStream in = new ObjectInputStream(in: serviceSocket.getInputStream());

            // send service request
            IMessage message = (IMessage)Message.getInstance(clientId:this.clientPort, requestId: -1, serverId:0,
                    messageCode:Consts.REQUESTCLIENT, iterations: 0, pi: 0, deadline:0);
            out.writeObject(obj:message);

            // receive service response
            message = (IMessage)in.readObject();
            this.clientPort = message.getClientId();
            System.out.println("Received client id " + this.clientPort);
            this.textArea.append("Received client id " + this.clientPort + "\n");

            break;
```

- [Client] Request pi.

```
public void requestPiService(int iterations, int deadline) {
    while (true) {
        // connect to the load balancer
        try (Socket serviceSocket = new Socket(host:this.hostname, port:this.loadBalancerPort)) {
            System.out.println("SERVICE CLIENT SOCKET=" + serviceSocket);

            ObjectOutputStream out = new ObjectOutputStream(out:serviceSocket.getOutputStream());
            ObjectInputStream in = new ObjectInputStream(in: serviceSocket.getInputStream());

            // send service request
            IMessage message = (IMessage)Message.getInstance(clientId:this.clientPort, requestId: -1,
                    serverId: 0, messageCode:Consts.REQUESTPI, iterations, pi: 0.0, deadline);
            out.writeObject(obj:message);
            System.out.println("RQST > " + message);
            this.textArea.append("RQST > " + message + "\n");

            break;
```

- [Load Balancer] Send client reply (client id or pi computation).

```
public void sendClientReply(IMessage message) {
    // connect to client
    try {
        Socket serviceSocket = new Socket(host:this.hostname, port:message.getClientId());
        System.out.println("CLIENT SOCKET=" + serviceSocket);

        ObjectOutputStream out = new ObjectOutputStream(out:serviceSocket.getOutputStream());
        //ObjectInputStream in = new ObjectInputStream(serviceSocket.getInputStream());
        this.textArea.append("RPLY > " + message.toString() + "\n");

        // send service reply
        out.writeObject(obj:message);

    } catch(UnknownHostException e) {
        System.out.println("Could not connect to client (UnknownHostException " + e.getMessage() + ")");
    } catch (IOException e) {
        System.out.println("Could not connect to client (IOException " + e.getMessage() + ")");
    }
}
```

3 - Data Validation/Lookup

- [Load Balancer] Establish monitor connection.

```
public void requestPiService(int iterations, int deadline) {
    while (true) {
        // connect to the load balancer
        try (Socket serviceSocket = new Socket(host:this.hostname, port:this.loadBalancerPort)) {
            System.out.println("SERVICE CLIENT SOCKET=" + serviceSocket);

            ObjectOutputStream out = new ObjectOutputStream(out:serviceSocket.getOutputStream());
            ObjectInputStream in = new ObjectInputStream(in: serviceSocket.getInputStream());

            // send service request
            IMessage message = (IMessage)Message.getInstance(clientId:this.clientPort, requestId: -1,
                    serverId: 0, messageCode:Consts.REQUESTPI, iterations, pi: 0.0, deadline);
            out.writeObject(obj:message);
            System.out.println("RQST > " + message);
            this.textArea.append("RQST > " + message + "\n");

            break;
```

- [Load Balancer] Send message to be processed by the monitor (get client id or server to which the pi request will be sent).

```
// message has no assigned server, communicate with monitor
if (message.getServerId() == 0) {

    try (Socket monitorSocket = new Socket(host: this.hostname, port: this.monitorPort)) {
        ObjectOutputStream outMonitor = new ObjectOutputStream(out: monitorSocket.getOutputStream());
        ObjectInputStream inMonitor = new ObjectInputStream(in: monitorSocket.getInputStream());

        // send request to monitor
        outMonitor.writeObject(obj: message);

        // get monitor response
        message = (IMessage)inMonitor.readObject();

        if (message.getMessageCode() == Consts.REPLYCLIENT) {
            outRequest.writeObject(obj: message);
        }
        else if (message.getMessageCode() == Consts.REQUESTPI) {
            // send service request to available server
            try {
                Socket socket = new Socket(host: this.hostname, port: message.getServerId());
                System.out.println("SERVICE LOAD BALANCER SOCKET=" + socket);

                ObjectOutputStream outServer = new ObjectOutputStream(out: socket.getOutputStream());
                ObjectInputStream inServer = new ObjectInputStream(in: socket.getInputStream());

                // send pi request
                outServer.writeObject(obj: message);
```

- [Load Balancer] Send pi request message to be processed by the server.

```
// message processed by a server
else {
    // send reply to client
    sendClientReply(message);

    // signal monitor that the service request was completed
    try (Socket monitorSocket = new Socket(host: this.hostname, port: this.monitorPort)) {
        ObjectOutputStream outMonitor = new ObjectOutputStream(out: monitorSocket.getOutputStream());
        ObjectInputStream inMonitor = new ObjectInputStream(in: monitorSocket.getInputStream());

        outMonitor.writeObject(obj: message);

    } catch(UnknownHostException e) {
        System.out.println("Could not connect to service (UnknownHostException " + e.getMessage() + ")");
    } catch (IOException e) {
        System.out.println("Could not connect to service (IOException " + e.getMessage() + ")");
    }
}
```

- [Monitor] Handle service management requests.

```
public void handleRequest(Socket monitorHandlerSocket) {
    int assignedServerPort = 0;
    try {
        // NOTE(L1fer): ObjectOuputStream needs to be created before ObjectInputStream
        ObjectOutputStream out = new ObjectOutputStream(out:monitorHandlerSocket.getOutputStream());
        ObjectInputStream in = new ObjectInputStream(in: monitorHandlerSocket.getInputStream());

        // request handling
        //while (true) {
        IMessage message;
        try {
            message = (IMessage)in.readObject();

            // server port request
            if (message.getMessageCode() == Consts.REQUESTPORT) {
                message = this.mManagerMonitor.getAvailablePort(message);
                assignedServerPort = message.getServerId();
                out.writeObject(obj:message);

                // keep connection with server alive in a blocking state
                in.readObject();
            }
            // client id request
            else if (message.getMessageCode() == Consts.REQUESTCLIENT) {
                this.mManagerMonitor.getClientId(message);
                out.writeObject(obj:message);
            }
```

4 - Server Connection

- [Server] Establish monitor connection (requests server port which will also serve as the id).

```
public void establishMonitorConnection() {
    while (true) {
        // connect to the monitor
        try {
            Socket monitorSocket = new Socket(host:this.hostname, port:this.monitorPort);
            System.out.println("MONITOR CLIENT SOCKET=" + monitorSocket);

            ObjectOutputStream out = new ObjectOutputStream(out:monitorSocket.getOutputStream());
            ObjectInputStream in = new ObjectInputStream(in: monitorSocket.getInputStream());

            // send port request
            out.writeObject(obj:Message.getInstance(clientId: 0, requestId: -1, serverId: 0, messageCode:Consts.REQUESTPORT, iterations:

            // get port response
            try {
                this.serverPort = ((IMessage)in.readObject()).getServerId();
            } catch (ClassNotFoundException e) {
                System.out.println("ClassNotFoundException " + e.getMessage());
            }

            // keep connection alive in a blocking state
            new Thread(() -> {
                try {
                    in.read();
                } catch (IOException e) {
                    System.out.println("Monitor server was shutdown (IOException " + e.getMessage() + ")");
                    System.exit(status: 0);
                }
            }).start();

            break;
```

- [Monitor] Send port assigned to the server.

```
// server port request
if (message.getMessageCode() == Consts.REQUESTPORT) {
    message = this.mManagerMonitor.getAvailablePort(message);
    assignedServerPort = message.getServerId();
    out.writeObject(obj:message);

    // keep connection with server alive in a blocking state
    in.readObject();
}
```

5 - Service Communication

- [Load Balancer] Send pi request to the assigned server.

```
else if (message.getMessageCode() == Consts.REQUESTPI) {
    // send service request to available server
    try {
        Socket socket = new Socket(host:this.hostname, port:message.getServerId());
        System.out.println("SERVICE LOAD BALANCER SOCKET=" + socket);

        ObjectOutputStream outServer = new ObjectOutputStream(out:socket.getOutputStream());
        ObjectInputStream inServer = new ObjectInputStream(in: socket.getInputStream());

        // send pi request
        outServer.writeObject(obj:message);
        // receive server reply
        //message = (IMessage)inServer.readObject();
    } catch (IOException e) {
        System.out.println("No servers available (IOException " + e.getMessage() + ")");
        this.textArea.append(str:"No servers available\n");
        message.setMessageCode(messageCode:Consts.REPLYREJECT);
        sendClientReply(message);
    }
}
```

- [Server] Send pi reply to the load balancer.

```
public void sendServiceReply(IMessage message) {
    // connect to client
    try {
        Socket serviceSocket = new Socket(host:this.hostname, port:this.loadBalancerPort);
        System.out.println("LOAD BALANCER SOCKET=" + serviceSocket);

        ObjectOutputStream out = new ObjectOutputStream(out:serviceSocket.getOutputStream());
        //ObjectInputStream in = new ObjectInputStream(serviceSocket.getInputStream());

        // send service reply
        out.writeObject(obj:message);

        this.processedRequestsTextArea.append("RPLY > " + message + "\n");

    } catch(UnknownHostException e) {
        System.out.println("Could not connect to load balancer (UnknownHostException " + e.getMessage() + ")");
    } catch (IOException e) {
        System.out.println("Could not connect to load balancer (IOException " + e.getMessage() + ")");
    }
}
```

6 – Hearbeat Checkup

- [Primary Load Balancer] Handle hearbeats.

```
public void handleHeartBeat(Socket heartbeatSocket) {
    try {
        DataInputStream in = new DataInputStream(in: heartbeatSocket.getInputStream());
        DataOutputStream out = new DataOutputStream(out:heartbeatSocket.getOutputStream());

        // heartbeat handling
        while (true) {
            byte data = in.readByte();
            out.writeByte(v: 0xAA);
        }
    } catch (IOException e) {
        System.out.println("IOException " + e.getMessage());
    }
```

- [Secondary Load Balancer] Check hearbeats.

```
public void checkHeartBeat() {
    // try to establish a socket connection to the primary load balancer
    try {
        Socket heartbeatSocket = new Socket(host: this.hostname, port: this.heartbeatPort);
        System.out.println("HEARTBEAT CLIENT SOCKET=" + heartbeatSocket);

        DataInputStream in = new DataInputStream(in: heartbeatSocket.getInputStream());
        DataOutputStream out = new DataOutputStream(out:heartbeatSocket.getOutputStream());

        // keep checking hearbeats while the primary server is up
        while (true) {
            // send heartbeat request
            out.write(b: 0xAA);
            // receive heartbeat response
            byte data = in.readByte();

            System.out.println(x: "Received heart beat");
            this.textArea.append(str:"Received heart beat\n");

            Thread.sleep(millis: this.heartbeatDelay);
        }
    } catch(UnknownHostException e) {
        System.out.println("No primary load balancer avaialbe (UnknownHostException " + e.getMessage() + ")");
    } catch (IOException e) {
        System.out.println("No primary load balancer avaialbe (IOException " + e.getMessage() + ")");
    } catch (InterruptedException e) {
        System.out.println("No primary load balancer avaialbe (InterruptedException " + e.getMessage() + ")");
    }
}
```

Incorrect implementations

-

Students contribution

1. João Bernardo Coelho Leite (nº 115041) [contribution: 45%]
    - Processes Management
    - Client Communication
    - Data Validation/Lookup
    - Server Connection
    - Service Communication

2. Luís Miguel Gomes Batista (nº 115279) [contribution: 55%]
    - Processes Management
    - Client Communication
    - Server Connection
    - Service Communication
    - Hearbeat Checkup