UNIVERSIDADE
FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM **ENGENHARIA DE COMPUTAÇÃO**
SÉTIMO SEMESTRE

QXD0043 - SISTEMAS DISTRIBUÍDOS

Relatório do Projeto "Agenda de Contatos Distribuída"

QUIXADÁ - CE
2022

471047 — PEDRO HENRIQUE **MAGALHÃES BOTELHO**

475664 — DAVID MACHADO **COUTO BEZERRA**

493470 — PAULO ARAGÃO DE **AZEVEDO NETO**

Relatório do Projeto "Agenda de Contatos Distribuída"

Orientador: Prof. Me. Marcos Dantas Ortiz

Relatório escrito para a disciplina de Sistemas Distribuídos, no curso de graduação em Engenharia de Computação, pela Universidade Federal do Ceará (UFC), campus em Quixadá.

QUIXADÁ - CE
2022

Conteúdo

Lista de Figuras

1 Introdução

Esse é o relatório do projeto final da disciplina de **Sistemas Distribuídos**, ministrada pelo professor Marcos Dantas Ortiz. Nesse relatório é discutido sobre a implementação de uma agenda de contatos distribuída. O objetivo do projeto foi implementar funcionalidades de uma agenda e suas operações com os contatos de maneira distribuída, onde um servidor fornece os métodos e um banco de dados e o cliente pode os acessar por meio de um **proxy**. Os métodos implementados foram, adicionar, editar, procurar, remover, listar contatos e limpar agenda.

O projeto foi desenvolvido inteiramente em linguagem Java, utilizando a arquitetura cliente-servidor, com protocolo **UDP** e mensagens no padrão JSON, de forma que a comunicação é facilitada e padronizada.

O repositório do projeto, com o código, apresentação de slides e diagrama UML pode ser encontrado em [1].

2 Definição do Serviço Remoto

2.1 Definição de Parâmetros

Foram definidas três classes que vão compor a mensagem: **Contato**, **Endereco** e **Data**. Essas três classes foram otimizadas para o uso com JSON: o método **toString()** destas classes converte o objeto em uma **String JSON**, que pode ser utilizada para a criação de um **JSONObject**, muito usado no projeto para manusear Strings JSON.

1. **Contato**
    - primeiroNome: **String**;
    - ultimoNome: **String**;
    - telefone01: **String**;
    - telefone02: **String**;
    - diaAniversario **Data**;
    - email **String**;
    - enderecoCasa **Endereco**;
    - id: **String**;
2. **Endereco**
    - rua: **String**;
    - bairro: **String**;

- cidade: **String**;
- cep: **String**;

3. **Data**
    - dia: **int**;
    - mes: **int**;
    - ano: **int**;

2.2 Métodos Implementados

1. **adicionarContato:**
    - **In:** Contato;
    - **Out:** Boolean;
    - **Exceções:** Nome ou telefone vazios;
2. **listarContatos:**
    - **In:** Void;
    - **Out:** Lista de Contatos;
    - **Exceções:** Nenhuma;
3. **buscarContatos:**
    - **In:** Nome;
    - **Out:** Contato;
    - **Exceções:** Nome ou telefone vazios;
4. **editarContato:**
    - **In:** Id;
    - **Out:** Boolean;
    - **Exceções:** Nome ou telefone vazios;
5. **removerContato:**
    - **In:** Nome;
    - **Out:** Boolean;
    - **Exceções:** Nenhuma;
6. **limparAgenda:**
    - **In:** Void;
    - **Out:** Void;
    - **Exceções:** Nenhuma;

3 Diagrama de Classe

Com os métodos definidos foi possível montar o diagrama de classes UML.

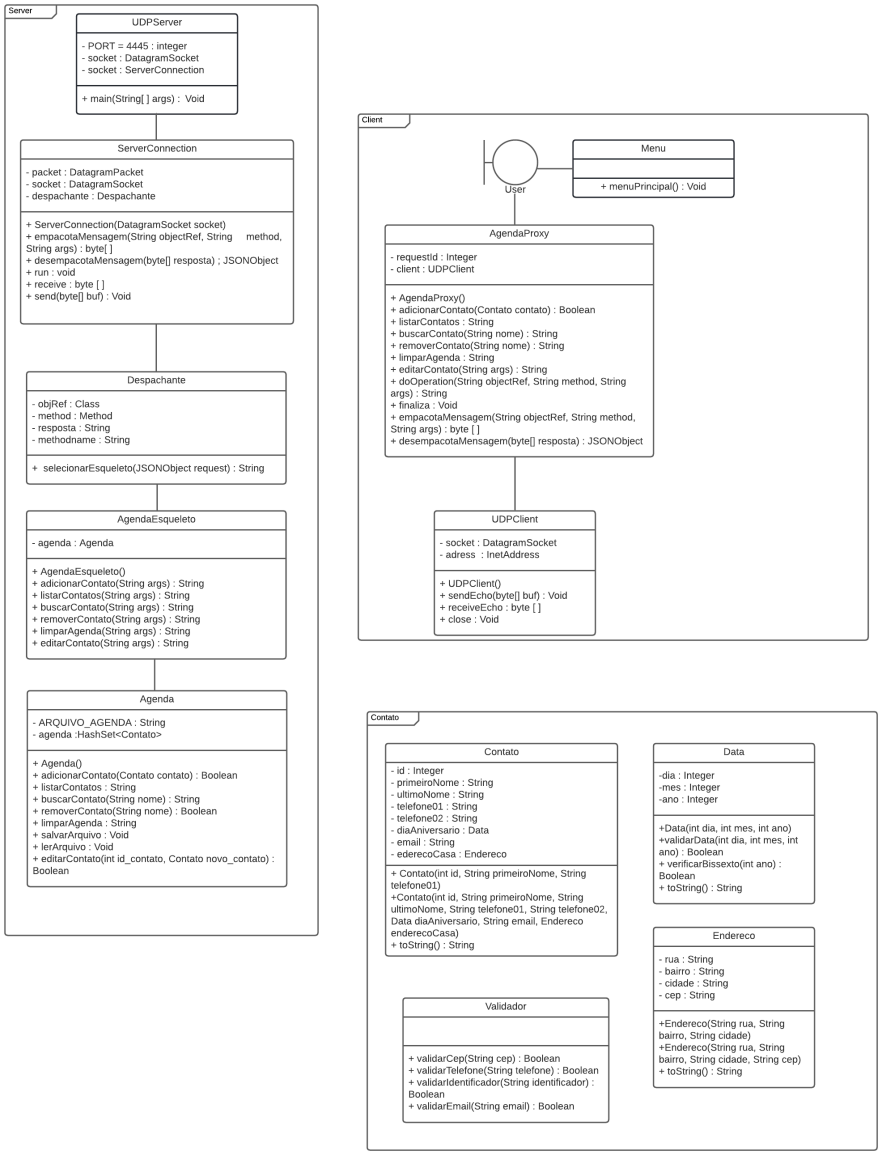


Figura 1: Diagrama de Classes

4 Descrição das Classes

4.1 Cliente

O lado cliente do sistema possui as seguintes classes:

- **User**;
- **agendaProxy**;
- **UDPClient**;

A classe **User** é o meio de interação do usuário com o sistema, uma instancia **agendaProxy** é criada na classe User para pre carregar os recursos do sistema.

A classe **agendaProxy** abstrai as chamadas para o servidor e possui também a função de empacotamento e desempacotamento das mensagens trocadas entre o cliente e o servidor via UDP.

Por fim a classe **UDPClient** efetua a comunicação com o servidor trocando os dados anteriormente enpacotados.

4.2 Servidor

O lado servidor do sistema possui as seguintes classes:

- **UDPServer**;
- **ServerConnection**;
- **Despachante**;
- **AgendaEsqueleto**;
- **Agenda**;

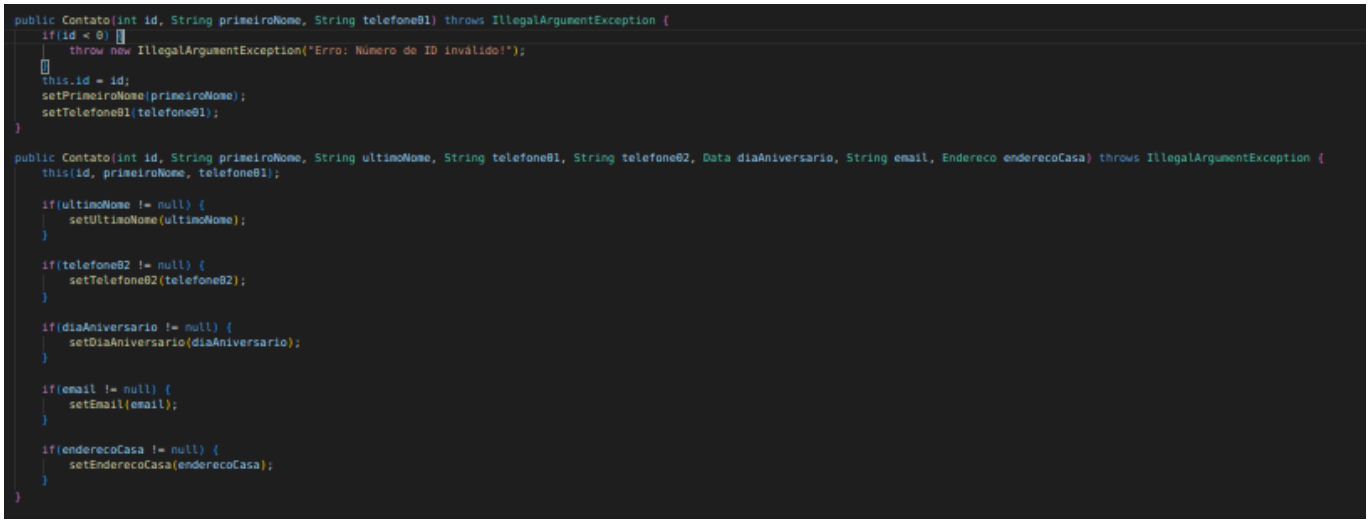
A classe **UDPServer** é responsável por atender as requisições enquanto a classe **ServerConnection** e responsável por desempacotar e de-serializar as mensagens assim como empacotar e serializar as respostas e depois enviar de volta a resposta da requisição.

A classe **Despachante** identifica qual método está sendo requisitado enquanto o método anteriormente identificado é executado pela classe **AgendaEsqueleto** que retorna a saída dos métodos. Todos os métodos são implementados na classe **Agenda**.

5 Transmissão de Dados

Foi utilizado o **JSON** como forma de transmissão de dados via Socket UDP.

Os dados referentes a agenda são definidos na classes**Contato**, sua implementação pode ser visualizada na **figura 3**.



```
public Contato(int id, String primeiroNome, String telefone01) throws IllegalArgumentException {
    if(id < 0) {
        throw new IllegalArgumentException("Erro: Número de ID inválido!");
    }
    this.id = id;
    setPrimeiroNome(primeiroNome);
    setTelefone01(telefone01);
}

public Contato(int id, String primeiroNome, String ultimoNome, String telefone01, String telefone02, Data diaAniversario, String email, Endereco enderecoCasa) throws IllegalArgumentException {
    this(id, primeiroNome, telefone01);

    if(ultimoNome != null) {
        setUltimoNome(ultimoNome);
    }

    if(telefone02 != null) {
        setTelefone02(telefone02);
    }

    if(diaAniversario != null) {
        setDiaAniversario(diaAniversario);
    }

    if(email != null) {
        setEmail(email);
    }

    if(enderecoCasa != null) {
        setEnderecoCasa(enderecoCasa);
    }
}
```

Figura 2: Construtores da Classe Contato

O método **empacotaMensagem** empacota a requisição a ser transmitida, portanto dentro dela está contido toda a informação necessária para a requisição.

A implementação pode ser visualizada na **figura 2**. No servidor utiliza-se tipo 1, e no cliente tipo 0, para distinguir as mensagens.

```
public byte[] empacotaMensagem(JSONObject mensagem, String args) {
    JSONObject resposta = new JSONObject();
    resposta.put(key: "type", value: 1);
    resposta.put(key: "id", (int) mensagem.get(key: "id"));
    resposta.put(key: "objReference", (String) mensagem.get(key: "objReference"));
    resposta.put(key: "methodId", (String) mensagem.get(key: "methodId"));
    resposta.put(key: "arguments", args);

    return resposta.toString().getBytes();
}
```

Figura 3: Método de Empacotar Mensagens

Abaixo está o método para desempacotar as mensagens que foram recebidas, já empacotadas. Esse método, então, desempacota a requisição recebida.

```
public static JSONObject desempacotaMensagem(byte[] resposta) {
        String resposta_string = new String(resposta, StandardCharsets.UTF_8);
        return new JSONObject(resposta_string);
}
```

Figura 4: Método de Desempacotar Mensagens

6 Tratamento de Falhas

Em um sistema distribuído inúmeras falhas podem ocorrer, dentre elas a perda de mensagens de requisição ou resposta. Com o objetivo de tratar isso foi definido um timeout de 1s na classe **UDPClient**, caso a mensagem não seja respondida ela será retransmitida.

O tratamento pode ser observado na figura abaixo:

```
//construtor
public UDPClient() throws SocketException, UnknownHostException {
    socket = new DatagramSocket();
    address = InetAddress.getByName("localhost");
    socket.setSoTimeout(1000);
}
```

Figura 5: *Timeout* do Cliente

Ao completar uma requisição ela será armazenada em um **HashMap**, caso seja uma requisição processada anteriormente ela enviara a resposta sem ter a necessidade de percorrer todo o código novamente.

A HashMap pode ser vista nas imagens abaixo:

```
public class ServerConnection extends Thread {
    private DatagramPacket packet;
    private DatagramSocket socket;
    private Despachante despachante;
    private HashMap<Integer, byte[]> historicoMensagens;
```

Figura 6: HashMap usada para manter o histórico de mensagens

O servidor opera como mostrado abaixo, no método **run**: recebe a requisição, desempacota em uma mensagem, verifica se a mensagem de resposta já tinha sido criada e envia se sim. Se não, obtém o resultado referente ao objeto, método e argumentos requeridos, empacota a mensagem e envia ao cliente. É feito um teste com espera de 2 segundos, para testar a retransmissão.

```java
public void run() {
    while(true) {
        byte[] m = receive();

        // ============== Mensagem Recebida ===================
        System.out.println("RECEBIDA:" + (new String(m)));

        JSONObject mensagem = desempacotaMensagem(m);

        if(mensagem.get(key: "arguments").toString().equals("Finaliza")) {
            this.historicoMensagens = new HashMap<>();
            continue;
        }

        int id = (int) mensagem.get(key: "id");

        if(this.historicoMensagens.containsKey(id)) {
            System.out.println("REENVIANDO ID " + id);
            send(this.historicoMensagens.get(id));
            continue;
        }

        String resultado = despachante.selecionaEsqueleto(mensagem);

        byte[] buf = empacotaMensagem(mensagem, resultado);

        this.historicoMensagens.put(id, buf);

        // ============== Mensagem Enviada ==================
        System.out.println("ENVIADA: " + (new String(buf)));

        // TESTAR RETRANSMISSAO
        try {
            sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        send(buf);
    }
}
```

Figura 7: Operação do Servidor no método **run** da Thread

Também foi utilizado um **TreeMap** para o mapeamento dos contatos.

```java
public class Agenda {
    private static int current_id = 0;
    private static final String ARQUIVO_AGENDA = "database/agenda.txt";
    private TreeMap<Integer, Contato> agenda;

    public Agenda() {
        this.agenda = new TreeMap<>();
    }
```

Figura 8: Mapeamento dos Contatos via TreeMap

7 Conclusão

O trabalho requeriu o uso de varios topicos abordados em sala na disciplina de Sistemas Distribuidos, como por exemplo a implementação da arquitetura Cliente-Servidor,protocolo no formato requisição-resposta, métodos reflexivos, serialização e desserialização, uso do JSON, etc.

Referências

[1] Repositório do projeto:. https://github.com/pedrobotelho15/Agenda_Distribuida. Acessado em 2022-07-11.