INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Comunicação, Administração e Turismo

Sistemas distribuídos

Tema: Sistema de previsão de tempo

Número: 45827

Nome: Carlos Wilson Lopes Furtado

Data de entrega: 19 / 02 /2023

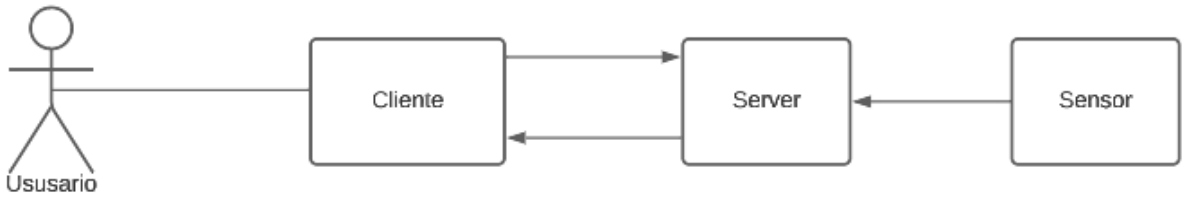
Introdução

No âmbito da disciplina de Sistemas Distribuídos do lecionada pelo professor Carlos Rompante, desafiei-me a implementar uma solução distribuída de um sistema Meteorológico que cumpra os seguintes requisitos:

- A componente servidor tem ligados um conjunto de sensores de temperatura e umidade espalhados pelo território, que são agrupados por distritos e concelhos.
- Cada sensor mede valores que envia para o servidor. A componente cliente obtém dados climáticos do servidor que lhe permite prever o estado meteorológico a cada instante (e.g. sol, chuva, etc.).
- O Sistema cliente deverá permitir obter qualquer dado meteorológico, que seja em tempo real, quer seja agregado por média diária ou semanal.
- A cadência com a qual se obtém dados dos sensores e servidor, deverá poder ser ajustada, pelo utilizador.

Estruturação do sistema .
É fundamental para a implementação de sistemas e softwares a compreensão e especificação dos componentes do sistema a ser desenvolvido e entender a fundo todos os dados, algoritmos e estruturas ali presentes.

Por isso, um dos primeiros passos logo após o levantamento de requisitos do sistema foi definir os componentes do sistema e a interação entre eles.



De acordo com a imagem acima o sistema possui 3 componentes principais:

- Cliente - Responsável por apresentar uma interface gráfica agradável ao utilizador e implementar os métodos de comunicação com o
- Servidor - Responsável por armazenar os dados recebidos do sensor e responder adequadamente a requisições do cliente.
- Sensor - Responsável por realizar a medição da Temperatura e Humidade e enviar esses dados ao servidor. Essa comunicação é unidirecional pois o sensor apenas envia o dado (temperatura, humidade, concelho distrito) ao servidor e não recebe nada dele.

Nota 1: O componente sensor num sistema mais robusto com certeza a comunicação entre o sensor e servidor seria bidirecional pois poderia ser necessário a realização da autenticação e registro dos sensores ativos mas como possivelmente passa do escopo desta disciplina e do próprio trabalho optei por não o implementar.

Protocolos
Os diferentes componentes do nosso sistema distribuído recorrem a alguns protocolos para entender com precisão o pedido ou a informação enviada.

No presente sistema se encontram os seguintes protocolos:
- <U> - Humidade
- <TP> - Temperatura
- <C> - Conselho
- <D> - Distrito
- <AC> - Lista de todos os concelhos de um determinado distrito
- <AD> - Lista de todos os distritos
- <EX> - Desligar do servidor

- <F> - Define a frequência ou taxa de atualização que os dados são enviados ao cliente
- <IT> - define o intervalo de tempo dos dados meteorológicos ex: média diária
- <L> - Define a localização
- <SA> - Identifica que os dados foram enviados pelo sensor

Servidor

O servidor de é composto por 3 classes principais:

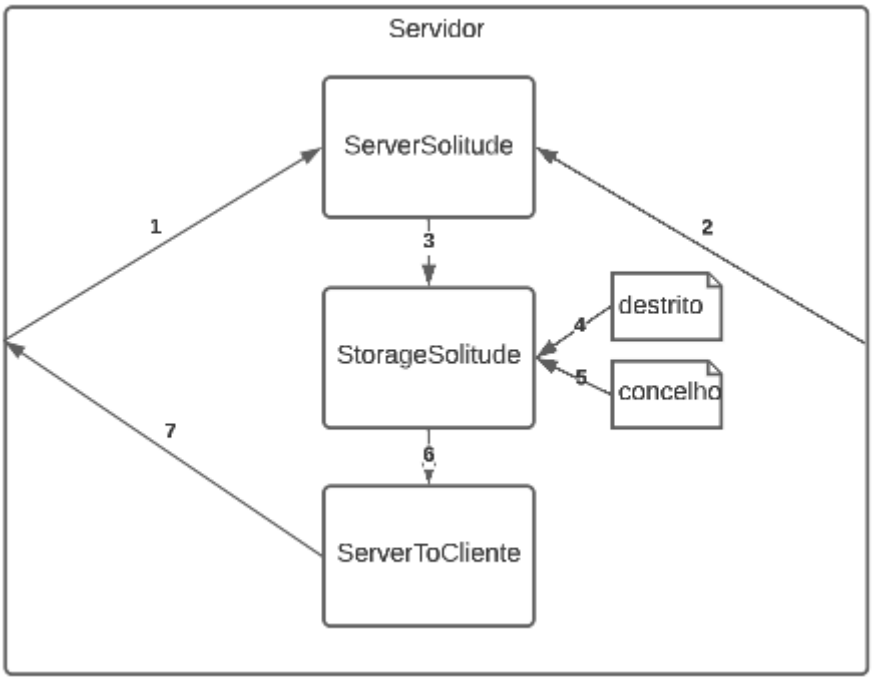
- ServidorSolitude - é responsável por aceitar as ligações do cliente e sensor além de tratar as requisições de acordo com os protocolos utilizados
- StorageSolitude - é responsável por armazenar os dados de temperatura e humidade recebidos pelo sensor além de responder os pedidos do servidor
- ServerToCliente - é responsável por enviar um fluxo constante de dados meteorológicos ao cliente

Nota 2:

O StorageSolitude é um falso banco de dados. Ele pode ser depois substituído por um banco de dados real sem muita mudança no código. Atualmente os dados que são retornados ao servidor sofrem algumas alterações nomeadamente no concelho e distrito para se adequar ao requisitado. Isso foi feito com a intenção de simular um cenário onde há sensores espalhados por todo o país.

Outro aspecto importante é que a lista de todos os concelhos existentes e distribuídos se encontram armazenados num arquivo.

A imagem abaixo retrata a interação básica entre as classes principais do servidor

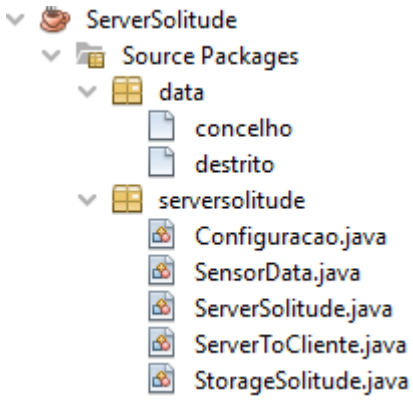


Legenda:

1. Requisição feita pelo cliente
2. Dados enviados constantemente pelo sensor
3. Dados de sensores adicionados pelo servidorSolitude
4. Leitura do arquivo de concelhos
5. Leitura do arquivo de distritos
6. Requisições de dados meteorológicos
7. Envio dos dados meteorológicos ao cliente

Nota: os dados enviados aos clientes dependem das confirmação definidas por elas.

Estrutura do projeto



Configuracao.java

Essa classe tem o objetivo de armazenar as configurações do cliente sendo elas frequencia (taxa de atualização ou a cada quantos milissegundos eu envio dados ao cliente referente a dados climáticos), distrito e concelho.

```
public class Configuracao {
    private int frequencia;
    private String destrito = "";
    private String concelho = "";

    public Configuracao(){}
    public Configuracao(String destrito, String concelho, int frequencia){...5 lines }

    public int getFrequencia(){...3 lines }

    public Configuracao setFrequencia(int milissegundos){...4 lines }

    public String getDestrito(){...3 lines }

    public Configuracao setDestrito( String destrito){...4 lines }

    public String getConcelho(){...3 lines }

    public Configuracao setConcelho(String concelho){...4 lines }
    public static String ParseToString(Configuracao config){...5 lines }
    public static Configuracao ParseToConfiguracao(Configuracao config,String msg){...2
}
```

SensorData.java

Armazena os dados do sensor como umidade, temperatura, distrito, concelho e datetime(data da criação da medida).

```
public class SensorData {
    private double umidade;
    private double temperatura;
    private LocalDateTime datetime;
    private String destrito = null;
    private String concelho = null;

    public SensorData(){}
    public SensorData(String destrito, String concelho, double temperatura, double umidade){...7 lines }
    public SensorData(String destrito, String concelho,double temperatura, double umidade, LocalDateTime datetime)
    public double getUmidade(){...3 lines }
    public void setUmidade(double umidade){...3 lines }
    public double getTemperatura(){...3 lines }
    public void setTemperatura(double temperatura){...3 lines }
    public LocalDateTime getDateTime(){...3 lines }
    public void setDateTime(LocalDateTime datetime){...3 lines }
    public String getDestrito(){...3 lines }
    public String getConcelho(){...3 lines }
    public SensorData setDestrito(String destrito){...4 lines }
    public SensorData setConcelho(String concelho){...4 lines }
    public static String ParseToString(SensorData sensorData){...8 lines }
    public static SensorData ParseToSensorData(SensorData sensorData, String msg){...18 lines }

}
```

ServerSolitude.java

```
public class ServerSolitude extends Thread{
    public ServerSolitude(){}

    public ServerSolitude(Socket ligacao){...4 lines }

    static StorageSolitude storageSolitude;
    Socket ligacao = null;
    Configuracao config = null;
    ServerToCliente ThreadSaidaDeDados;

    public static void main(String[] args) {...20 lines }

    @Override
    public void run(){...22 lines }

    // Trada as entradas dos clientes de forma a funcionar no metodos despostos no Servidor
    public void Action(String msg){...51 lines }

    public static void addFila(SensorData sensorData){...3 lines }
}
```

A classe ServerSolitude tem as seguintes funções principais:

- main - aceita as ligações dos cliente e dos sensores e cria um thread para parar cada ligação.
- run - cria um loop que fica recebendo as requisições do cliente e tratando ele devidamente. E cria um thread ServerToCliente que envia dados constantemente ao cliente
- Action - trata as mensagens recebidas pelo cliente e aciona on métodos adequados

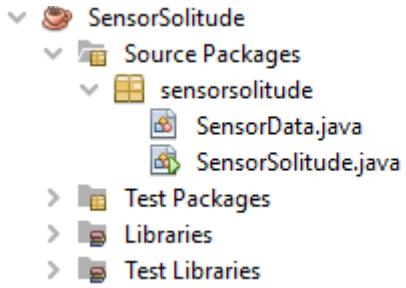
- addFila - adiciona os dados do sensor ao StorageSolitude

StorageSolitude.java
A classe StorageSolitude executa a função de um banco de dados, armazenando e manipulando os dados do sensor.

Como referido anteriormente o lastItem altera o “concelho” e “destrito” para ser igual ao requisitado não realizando realmente uma pesquisa e filtrando de acordo com a requisição.

```
public class StorageSolitude {

    public static int Diario = 1;
    public static int Semanal = 2;

    public StorageSolitude(){}
    public StorageSolitude(ArrayList<SensorData> data){...3 lines }
    private ArrayList<SensorData> data;
    public void add(SensorData item){...3 lines }
    public SensorData lastItem(){...19 lines }
    public static void add(StorageSolitude storageSolitude, SensorData item){...3 lines }
    public SensorData lastItem(String destrito, String concelho){...3 lines }
    public SensorData lastItem(int Periodo){...3 lines }
    public SensorData lastItem(String destrito, String concelho, int Periodo){...3 lines }
    public ArrayList<String> getDestrito(){...3 lines }
    public ArrayList<String> getConcelho(String destrito){...10 lines }
    private ArrayList<String> LerArquivoData(String nomeArquivo){...15 lines }

}
```

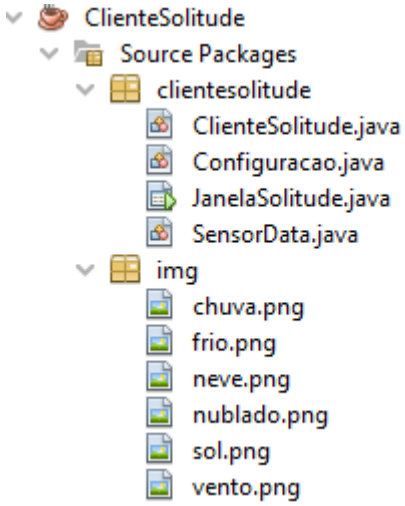
SensorSolitude



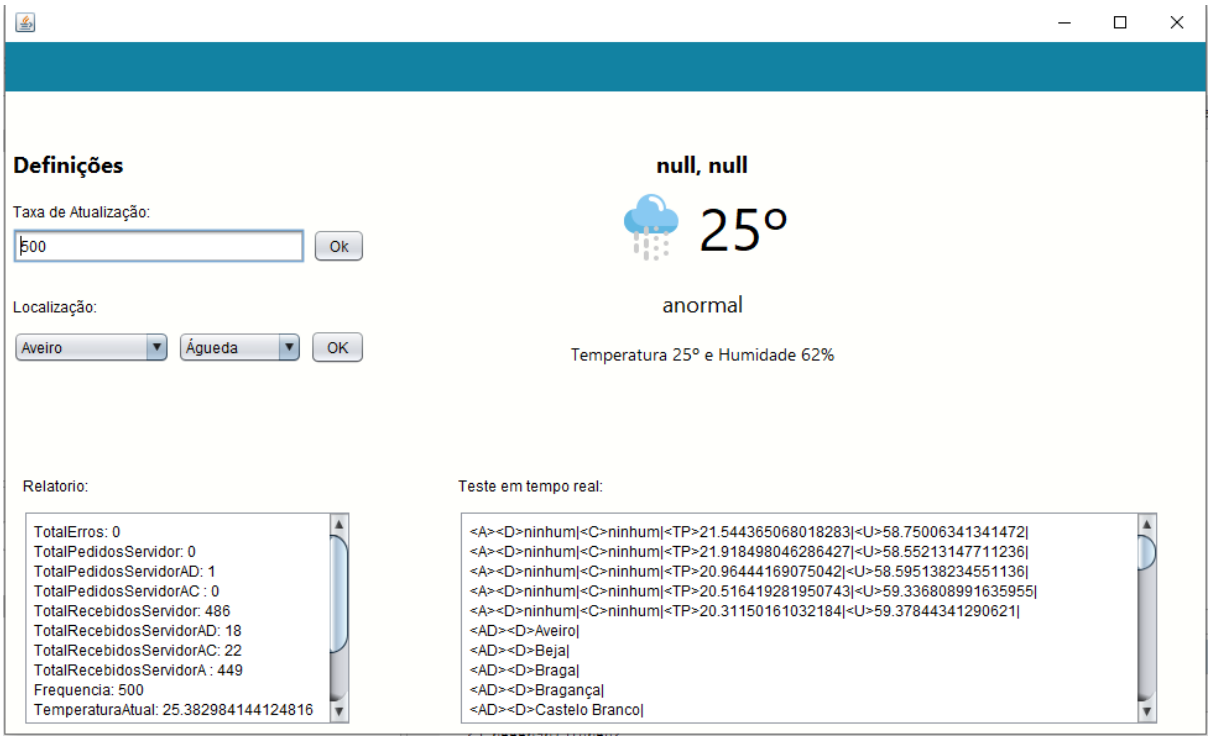
SensorSolitude.java
A classe sensor gera os dados de temperatura e umidade randomicamente e os envia ao servidor num fluxo constante de dados.

```
public class SensorSolitude extends Thread{

    public SensorSolitude(){}
    private static boolean Aleatorio = true;
    private static double Temperatura = 0.0;
    private static double Umidade = 0.0;

    public static void main(String[] args) {...35 lines }

    private static double getRandom(double temperaturaBase){...3 lines }
    @Override
    public void run(){...19 lines }
}
```

ClienteSolitude



JanelaSolitude.java



```
public class JanelaSolitude extends javax.swing.JFrame {

    /**
     * Creates new form JanelaSolitude
     */

    private ClienteSolitude cliente = null;
    private ArrayList<String> destritos = new ArrayList<>();
    private ArrayList<String> concelhos = null;

    public void resetDestritos(){...3 lines }
    public void addDestrito(String destrito){...3 lines }
    public void resetConcelho(){...3 lines }
    public String getSelectedDestrito(){...3 lines }
    public String getSelectedConcelho(){...3 lines }
    public void addConcelho(String destrito){...3 lines }
    public void setImagemSol(){...5 lines }
    public void setImagemChuva(){...5 lines }
    public void setImagemNublado(){...5 lines }
    public void setImagemNeve(){...5 lines }
    public void setImagemFrio(){...5 lines }
    public void setLocalizacao(String destrito, String concelho){...3 lines }
    public void setFrequencia(String taxaDeAtualizacao){...3 lines }
    public void setRelatorio(){...3 lines }
    public JanelaSolitude() {...5 lines }
    public void enviarMsgTempoReal(String msg){...3 lines }
    public void atualizarTemperaturaHumidade(double temperatura, double humidade){...13 lines }
    public void atualizarLocalizacao(String Destrito, String Concelho){...3 lines }
    public void atualizarPrevisaoTempo(double temperatura, double humidade){...3 lines }
```

ClienteSolitude.java

A classe ClienteSolitude ‘e inicializada pela Janela Solitude. o Cliente Solitude ‘e responsavel por estabelecer a comunicação com o SeverSolitude, enviar pedidos ao servidor de acordo com os protocolos especificados além de tratar as respostas do servidor e atualizar a tela de acordo com as informações recebidas.

O método prevê o estado do clima de acordo com a umidade e temperatura basicamente percorre um conjunto de ifs e retorna a resposta mais adequada. De acorda com o estado de clima essa mesma função altera a imagem na janelaSolitude ao que mais se adequa ao estado de clima.

```java
public class ClienteSolitude extends Thread {

    private static PrintStream saidaDeDados = null;
    private static BufferedReader entradaDeDados = null;

    static Configuracao config = new Configuracao();
    static boolean run = true;
    public static JanelaSolitude janela = null;

    // variaveis para relatorio
    private static int TotalErros = 0;
    private static int TotalPedidosServidor = 0;
    private static int TotalPedidosServidorAD = 0;
    private static int TotalPedidosServidorAC = 0;
    private static int TotalRecebidosServidor = 0;
    private static int TotalRecebidosServidorAD = 0;
    private static int TotalRecebidosServidorAC = 0;
    private static int TotalRecebidosServidorA = 0;
    private static int Frequencia = 0;
    private static double TemperaturaAtual = 0.0;
    private static double UmidadeAtual = 0.0;
    private static String ConcelhoAtual = "";
    private static String DestritoAtual = "";
    public ClienteSolitude(JanelaSolitude janela){...3 lines }
    public ClienteSolitude(Socket ligacao){...8 lines }
    public boolean ligarServidor(){...28 lines }
    @Override
    public void run(){...18 lines }
    public String GerarRelatorio(){...15 lines }

    // executa uma açao especifica de acordo com o protocolo recebido
    public void Action(String msg){...32 lines }
    public void setFrequencia(int milissegundos){...6 lines }
    public void setLocalizacao(String destrito, String concelho ){...7 lines }
    public void setIntevaloTempo(int intervaloTempo){...4 lines }
    public void getAllDestrito(){...6 lines }
    public void getAllConcelho(String destrito){...6 lines }
    public void exit(){...4 lines }
    public String preverClima(double temperatura, double humidade){...28 lines }

}
```