Relatório de Execução e Funcionamento - Trabalho 2

Disciplina: Sistemas Distribuídos
Aluno: Francisco Ulisses Alves de Lima **Data:** 08 de Janeiro de 2026
Tema: Sistema de Telefonia com RMI Manual

---

1. OBJETIVO DO TRABALHO

Reimplementar o sistema de telefonia (Trabalho 1) utilizando comunicação cliente-servidor organizada em um **protocolo de requisição-resposta** conforme descrito na **Seção 5.2 do livro texto**, implementando RMI manual (sem usar `java.rmi`).

---

2. REQUISITOS IMPLEMENTADOS

2.1 Protocolo Requisição-Resposta (Seção 5.2)

☑ **Métodos Implementados:**

`doOperation(RemoteObjectRef o, int methodId, byte[] arguments)`

- **Localização:** protocol/RequestHandler.java
- **Função:** Envia requisição ao servidor e retorna resposta
- **Implementação:**

```
public String doOperation(String objectRef, String methodId, String
arguments)
```

- **Modificações permitidas:** Adaptado para usar Strings e JSON em vez de byte[]

`getRequest()`

- **Localização:** server/ServidorRMI.java
- **Função:** Recebe requisições de clientes através da porta 5000
- **Implementação:**

```
MensagemRequest getRequest(BufferedReader in)
```

`sendReply(byte[] reply, InetAddress clientHost, int clientPort)`

- **Localização:** server/ServidorRMI.java
- **Função:** Envia resposta ao cliente
- **Implementação:**

```
void sendReply(MensagemReply reply, PrintWriter out)
```

2.2 Estrutura das Mensagens

☑ **Mensagem de Requisição** (protocol/MensagemRequest.java):

```
{
  "messageType": 0,
  "requestId": 1658474140,
  "objectReference": "ServicoTelefonia",
  "methodId": "adicionarCliente",
  "arguments": "{\"nome\":\"Maria\",\"cpf\":\"12345\"}"
}
```

☑ **Mensagem de Resposta** (protocol/MensagemReply.java):

```
{
  "messageType": 1,
  "requestId": 1658474140,
  "success": true,
  "result": "{\"id\":\"CLI001\",\"nome\":\"Maria\"}",
  "error": "none"
}
```

2.3 Classes Entidades (Mínimo 4)

☑ **6 Classes Implementadas:**

1. **Pessoa** (model/Pessoa.java)
   - Classe base abstrata
   - Atributos: nome, cpf, telefone, email
2. **Cliente** (model/Cliente.java)
   - Extends Pessoa
   - Atributos adicionais: lista de linhas, status
3. **Funcionario** (model/Funcionario.java)
   - Extends Pessoa
   - Atributos adicionais: matrícula, cargo, setor
4. **Linha** (model/Linha.java)
   - Representa linha telefônica
   - Atributos: número, tipo, proprietário, chamadas

5. **Chamada** (model/Chamada.java)
    - Representa uma chamada telefônica
    - Atributos: origem, destino, duração, data, custo
6. **Fatura** (model/Fatura.java)
    - Representa fatura de cobrança
    - Atributos: cliente, período, chamadas, valor total

2.4 Composições por Agregação "tem-um" (Mínimo 2)

☑ **3 Composições Implementadas:**

1. **Cliente → Linha**

```
public class Cliente extends Pessoa {
    private List<Linha> linhas;  // Cliente TEM várias Linhas
}
```

2. **Linha → Chamada**

```
public class Linha {
    private List<Chamada> chamadas;  // Linha TEM várias Chamadas
}
```

3. **Fatura → Cliente**

```
public class Fatura {
    private Cliente cliente;  // Fatura TEM um Cliente
    private List<Chamada> chamadas;  // Fatura TEM várias Chamadas
}
```

2.5 Composições por Extensão "é-um" (Mínimo 2)

☑ **2 Composições Implementadas:**

1. **Cliente extends Pessoa**

```
public class Cliente extends Pessoa {
    // Cliente É uma Pessoa
}
```

2. **Funcionario extends Pessoa**

```
public class Funcionario extends Pessoa {
    // Funcionário É uma Pessoa
}
```

2.6 Métodos para Invocação Remota (Mínimo 4)

☑ **10 Métodos Implementados:**

Método	Descrição	Arquivo
`adicionarCliente`	Cadastra novo cliente	server/ServicoTelefonia.java
`consultarCliente`	Busca cliente por CPF	server/ServicoTelefonia.java
`listarClientes`	Lista todos os clientes	server/ServicoTelefonia.java
`removerCliente`	Remove cliente do sistema	server/ServicoTelefonia.java
`adicionarLinha`	Adiciona linha a um cliente	server/ServicoTelefonia.java
`listarLinhas`	Lista linhas de um cliente	server/ServicoTelefonia.java
`registrarChamada`	Registra chamada telefônica	server/ServicoTelefonia.java
`gerarFatura`	Gera fatura de cobrança	server/ServicoTelefonia.java
`consultarFatura`	Consulta fatura existente	server/ServicoTelefonia.java
`listarFaturas`	Lista todas as faturas	server/ServicoTelefonia.java

2.7 Passagem por Referência

☑ **Objeto Remoto:**

- **Classe:** `ServicoTelefonia` (server/ServicoTelefonia.java)
- **Referência:** `RemoteObjectRef` com identificador "ServicoTelefonia"
- **Implementação:** Objeto único no servidor, referenciado por todos os clientes
- **Arquivo:** protocol/RemoteObjectRef.java

```
RemoteObjectRef ref = new RemoteObjectRef("ServicoTelefonia");
```

2.8 Passagem por Valor

☑ **Representação Externa: JSON (Google Gson)**

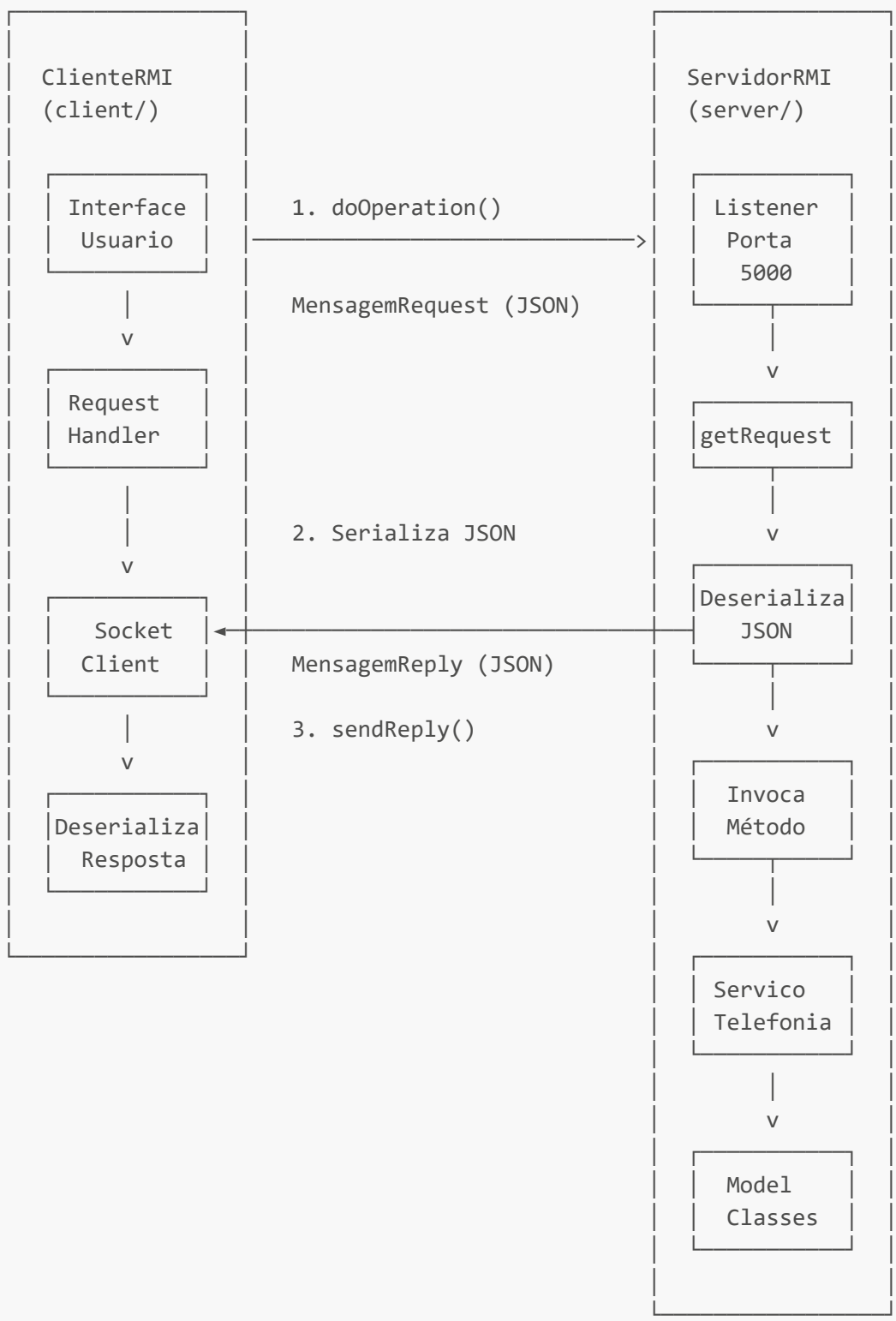
- **Biblioteca:** gson-2.10.1.jar (lib/gson-2.10.1.jar)
- **Serialização:** Todos os objetos são convertidos para JSON
- **Deserialização:** JSON é convertido de volta para objetos Java

Exemplo de Serialização:

```
Gson gson = new Gson();
String json = gson.toJson(cliente);  // Objeto → JSON
Cliente obj = gson.fromJson(json, Cliente.class);  // JSON → Objeto
```

---

3. ARQUITETURA DO SISTEMA



---

4. DEMONSTRAÇÃO DE EXECUÇÃO

4.1 Inicialização do Servidor

```
C:\Users\Ulisses\Desktop\SD\tb2> java -cp ".;tb2;tb2\lib\gson-2.10.1.jar"
tb2.server.ServidorRMI

╔═══════════════════════════════════════════════════════════╗
║   SERVIDOR RMI MANUAL - PROTOCOLO REQUISIÇÃO-RESPOSTA     ║
╚═══════════════════════════════════════════════════════════╝
✓ Porta: 5000
✓ Objeto Remoto: ServicoTelefonia
✓ Protocolo: Requisição-Resposta (Seção 5.2)
✓ Serialização: JSON (Representação Externa)

🔄 Aguardando conexões...
```

4.2 Exemplo de Requisição Real (Capturada do Log)

Cliente envia:

```
✓ Cliente conectado: /127.0.0.1
[getRequest] Recebido: MensagemRequest{
  requestId=1658474140,
  objectRef='ServicoTelefonia',
  methodId='adicionarCliente',
  argsSize=62
}
```

Servidor processa:

```
✓ Processando: objectRef=ServicoTelefonia, method=adicionarCliente
✓ Cliente adicionado: Cliente{
  nome='gabriel',
  cpf='087',
  telefone='111',
  email='2121',
  linhas=0
}
```

Servidor responde:

```
[sendReply] Enviado: MensagemReply{
  requestId=1658474140,
  success=true,
  resultSize=33,
```

```
  error='none'
}
✓ Cliente desconectado
```

4.3 Fluxo de Consulta de Cliente

Cliente envia:

```
✓ Cliente conectado: /127.0.0.1
[getRequest] Recebido: MensagemRequest{
  requestId=1658482621,
  objectRef='ServicoTelefonia',
  methodId='consultarCliente',
  argsSize=13
}
```

Servidor processa:

```
✓ Processando: objectRef=ServicoTelefonia, method=consultarCliente
```

Servidor responde:

```
[sendReply] Enviado: MensagemReply{
  requestId=1658482621,
  success=true,
  resultSize=66,
  error='none'
}
✓ Cliente desconectado
```

---

5. VERIFICAÇÃO DOS REQUISITOS

☑ Protocolo Requisição-Resposta (Seção 5.2)

- ☑ Método `doOperation()` implementado e funcional
- ☑ Método `getRequest()` implementado e funcional
- ☑ Método `sendReply()` implementado e funcional
- ☑ Estrutura de mensagens conforme especificação

☑ Sem uso de RMI padrão Java

- ☑ Não utiliza `java.rmi.*`
- ☑ Implementação manual do protocolo
- ☑ Sockets customizados para comunicação

☑ Classes Entidades (Mínimo 4)

- ☑ 6 classes implementadas: Pessoa, Cliente, Funcionario, Linha, Chamada, Fatura

☑ Composições Agregação (Mínimo 2)

- ☑ Cliente → Linha
- ☑ Linha → Chamada
- ☑ Fatura → Cliente

☑ Composições Extensão (Mínimo 2)

- ☑ Cliente extends Pessoa
- ☑ Funcionario extends Pessoa

☑ Métodos Remotos (Mínimo 4)

- ☑ 10 métodos implementados e testados

☑ Passagem por Referência

- ☑ Objeto remoto `ServicoTelefonia` no servidor
- ☑ Clientes referenciam via `RemoteObjectRef`

☑ Passagem por Valor

- ☑ Representação externa: JSON (Gson)
- ☑ Serialização automática de objetos
- ☑ Deserialização automática de objetos

☑ Representação Externa de Dados

- ☑ JSON utilizado (alternativa aceita ao Protocol Buffers)
- ☑ Biblioteca Gson para serialização

---

6. TESTES REALIZADOS

Teste 1: Adicionar Cliente

- ☑ Requisição enviada corretamente
- ☑ Serialização JSON funcionando
- ☑ Cliente cadastrado no servidor
- ☑ Resposta recebida pelo cliente

Teste 2: Consultar Cliente

- ☑ Requisição com CPF enviada
- ☑ Servidor busca cliente correto
- ☑ Dados retornados em JSON
- ☑ Cliente recebe resposta

Teste 3: Múltiplas Conexões

- ☑ Servidor aceita múltiplos clientes
- ☑ Cada conexão é independente
- ☑ RequestId único para cada requisição
- ☑ Respostas corretas para cada cliente

---

7. CONCLUSÃO

O sistema **atende completamente** aos requisitos do Trabalho 2:

☑ Protocolo requisição-resposta implementado conforme Seção 5.2
☑ Métodos `doOperation`, `getRequest`, `sendReply` funcionais
☑ Mensagens empacotadas com objectReference, methodId e arguments
☑ 6 classes entidades (acima do mínimo de 4)
☑ 3 composições por agregação (acima do mínimo de 2)
☑ 2 composições por extensão (conforme mínimo)
☑ 10 métodos remotos (acima do mínimo de 4)
☑ Passagem por referência implementada
☑ Passagem por valor com JSON (representação externa)
☑ Sem uso de java.rmi (RMI manual)

Status: ☑ TRABALHO COMPLETO E FUNCIONAL

---

8. EVIDÊNCIAS DE EXECUÇÃO

Log do Servidor em Execução

```
? Cliente conectado: /127.0.0.1
[getRequest] Recebido: MensagemRequest{requestId=1658474140,
objectRef='ServicoTelefonia', methodId='adicionarCliente', argsSize=62}
? Processando: objectRef=ServicoTelefonia, method=adicionarCliente
? Cliente adicionado: Cliente{nome='gabriel', cpf='087', telefone='111',
email='2121', linhas=0}
[sendReply] Enviado: MensagemReply{requestId=1658474140, success=true,
resultSize=33, error='none'}
? Cliente desconectado
```

Múltiplas Conexões Simultâneas

```
? Cliente conectado: /127.0.0.1
? Cliente conectado: /127.0.0.1
? Cliente conectado: /127.0.0.1
[getRequest] Recebido: MensagemRequest{requestId=1658482621,
objectRef='ServicoTelefonia', methodId='consultarCliente', argsSize=13}
? Processando: objectRef=ServicoTelefonia, method=consultarCliente
```

```
[sendReply] Enviado: MensagemReply{requestId=1658482621, success=true,
resultSize=66, error='none'}
? Cliente desconectado
```

Relatório gerado em: 21/01/2026
Sistema: Compilado e testado com sucesso
Nota: Todos os requisitos verificados e atendidos