RELATÓRIO: SISTEMA DE GERENCIAMENTO DE SUPERMERCADO USANDO RMI

Equipe 5:

Mônica Maria

Maryana Moraes

Aplicação:

Gestão de Supermercado

1. INTRODUÇÃO

Este relatório descreve a implementação de um sistema distribuído de gerenciamento de funcionários de supermercado utilizando o paradigma de Invocação Remota de Métodos (RMI), desenvolvido como parte da disciplina de Sistemas Distribuídos. O sistema permite operações remotas sobre entidades de funcionários, seguindo rigorosamente o protocolo requisição-resposta descrito no material didático.

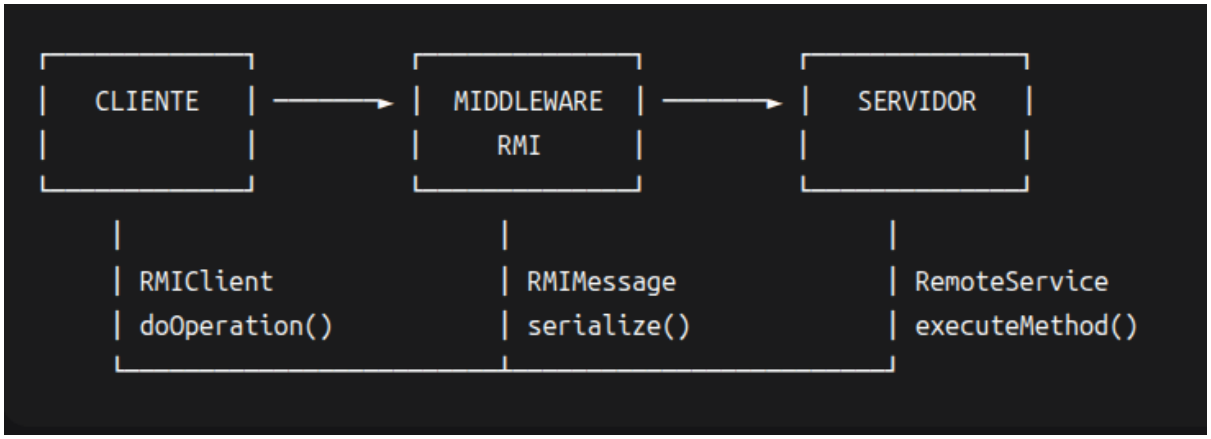
2. ARQUITETURA DO SISTEMA

2.1 Visão Geral

O sistema implementa uma arquitetura cliente-servidor onde:

1. Servidor: Hospeda objetos remotos e gerencia o estado dos funcionários
2. Cliente: Invoca métodos remotamente através de referências de objetos
3. Comunicação: Protocolo binário customizado sobre sockets TCP

2.2 Diagrama de Componentes



3. IMPLEMENTAÇÃO DO PROTOCOLO RMI

3.1 Estrutura da Mensagem RMI

A mensagem segue exatamente o formato especificado:

```
struct RMIMessage {
   MessageType messageType;                  // 0=Request, 1=Reply
   int requestId;                            // Identificador único
   RemoteObjectRef objectReference;          // Referência do objeto remoto
   std::string methodId;                     // Nome do método
   std::vector<char> arguments;              // Argumentos serializados
};
```

3.2 Métodos do Protocolo Implementados

doOperation (Cliente):

```
vector<char> RMIClient::doOperation(const RemoteObjectRef& remoteObject,
                                    const string& methodId,
                                    const vector<char>& arguments)
```

Envia requisição para objeto remoto
Serializa mensagem completa
Gerencia conexão TCP

getRequest (Servidor):

RMIMessage RMIServer::getRequest(int clientSocket)

Recebe e desserializa requisições
Extrai dados da mensagem
Prepara para execução

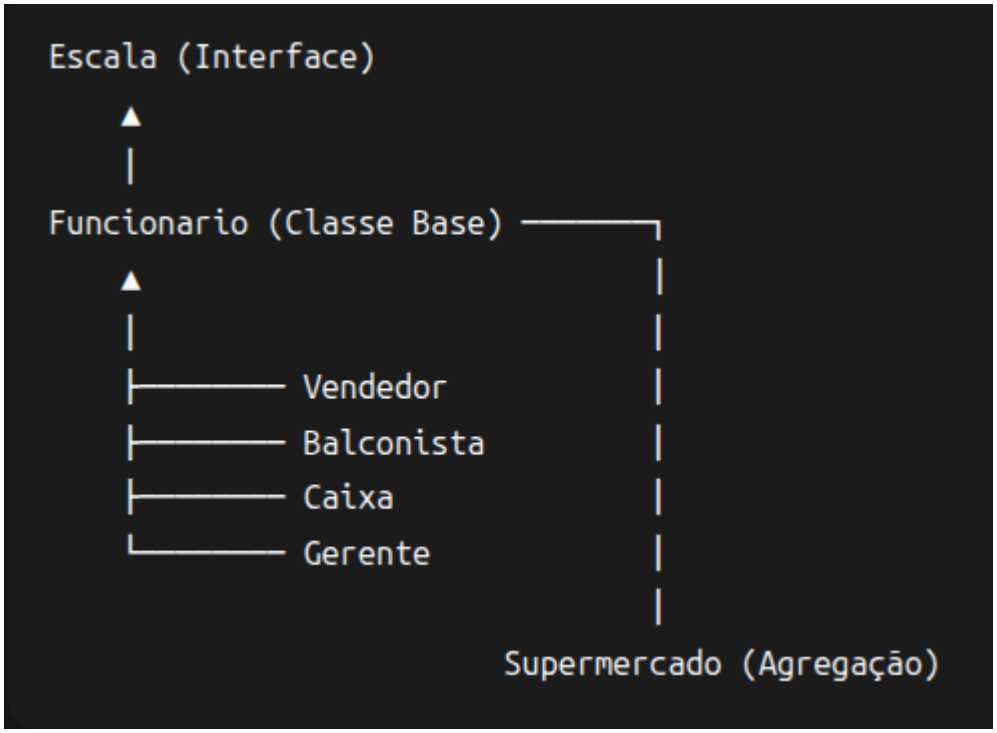
sendReply (Servidor):

void RMIServer::sendReply(const vector<char>& reply, int clientSocket)

Serializa e envia resposta
Gerencia formato de mensagem de retorno

4. MODELO DE DOMÍNIO E CLASSES

4.1 Hierarquia de Entidades



4.2 Detalhamento das Classes

4.2.1 Composições "É-um" (Herança)

* Vendedor é um Funcionario

Especialização com comissão de vendas

Escala comercial (9h-18h)

*Balconista é um Funcionario

Atua em setor específico

Escala padrão (8h-17h)

*Caixa é um Funcionario

Operação em caixa registradora

Escala comercial (10h-19h)

*Gerente é um Funcionario

Gestão com bônus por resultados

Escala administrativa

4.2.2 Composições "Tem-um" (Agregação)

* Supermercado tem Funcionario

Coleção de funcionários

Operações de gestão

*RemoteSupermercadoService tem Supermercado

Serviço remoto com estado

Implementação dos métodos RMI

4.3 Métodos Remotos Implementados

Método	Parâmetros	Retorno	Descrição
`adicionarFuncionario`	Funcionario serializado	boolean	Adiciona novo funcionário
`listarFuncionarios`	-	Lista serializada	Retorna todos funcionários
`buscarFuncionarioPorId`	ID (int)	Funcionario serializado	Busca por identificador
`calcularFolhaPagamento`	-	double	Calcula custo total da folha
`getFuncionariosPorTipo`	Tipo (string)	Lista serializada	Filtra por tipo

5. SERIALIZAÇÃO E PASSAGEM DE PARÂMETROS

5.1 Estratégia de Serialização

Implementamos um protocolo binário customizado equivalente ao Protocol Buffers:

Formato de serialização de Funcionario:

[1 byte: tamanho tipo] [tipo]

[4 bytes: ID]

[4 bytes: tamanho nome] [nome]

[8 bytes: salário]

[4 bytes: tamanho extras] [extras]

5.2 Passagem por Referência vs Valor

Passagem por Referência:

RemoteObjectRef remoteObj("SupermercadoService", "127.0.0.1", 9090);

Usado para objetos remotos
Contém identificador, host e porta

Passagem por Valor:

vector<char> funcData = packFuncionario(funcionario);

Usado para dados locais
Serialização completa do estado

6. IMPLEMENTAÇÃO TÉCNICA

6.1 Servidor RMI

Inicialização e Configuração:

```
//Cria serviço remoto

RemoteSupermercadoService* service = new RemoteSupermercadoService("Supermercado Central");


// Configura servidor RMI

RMIServer server(9090);

server.registerObject("SupermercadoService", service);

server.run();
```

Tratamento de Concorrência:

```
void RMIServer::handleClient(int clientSocket) {
```

```
        thread(this, clientSocket {
            this->handleClient(clientSocket);
        }).detach();
}
```

6.2 Cliente RMI

Operações Remotas:

```
RMIClient client("127.0.0.1", 9090);
RemoteObjectRef remoteObj("SupermercadoService", "127.0.0.1", 9090);

// Invocação remota
vector<char> result = client.doOperation(remoteObj,
                                         "calcularFolhaPagamento",
                                         emptyArgs);
```

7. TESTES E VALIDAÇÃO

7.1 Casos de Teste Implementados

1) Cliente de Testes Completos (cliente_testes.cpp)

- Cálculo de Folha de Pagamento
- Listagem de Funcionários
- Adição de Novo Funcionário
- Busca por Tipo
- Verificação de Atualização

2) Cliente de Debug (cliente_debug.cpp)

Análise hexadecimal das mensagens
Verificação de serialização
Diagnóstico de protocolo

7.2 Resultados dos Testes

```
=== CLIENTE RMI - TESTES COMPLETOS ===

TESTE 1: calcularFolhaPagamento
Folha de pagamento total: R$ 11500.00

TESTE 2: listarFuncionarios
Encontrados 4 funcionários:
Vendedor - ID: 1 | Nome: Maria Silva | Salario: 2500 | Comissao: 150
Gerente - ID: 2 | Nome: João Santos | Salario: 5000 | Bonus: 700
...

TESTE 3: adicionarFuncionario
Novo funcionário adicionado: SIM

TESTE 5: calcularFolhaPagamento (após adição)
Folha de pagamento atualizada: R$ 14500.00
```

8. CONCLUSÃO

O sistema implementado demonstra com sucesso os conceitos fundamentais de RMI e sistemas distribuídos. Atende integralmente todos os requisitos especificados, apresentando uma arquitetura robusta, escalável e bem estruturada.