

# MC823 – 1º Relatório

Marco Antônio Lasmar Almada – RA 092208

Patricia Fernanda Hongo – RA 103715

## 1 Introdução

O presente relatório tem como objetivo descrever a implementação e os resultados do terceiro projeto de MC823 no primeiro semestre de 2013, comparando-os com o resultados obtidos nos dois primeiros projetos. Tal como os Projetos 1 e 2, este projeto consistia na implementação de um sistema para consulta e controle de estoque em uma biblioteca, utilizando uma arquitetura cliente-servidor. Entretanto, a implementação o sistema desta vez utilizou a invocação remota de métodos da linguagem Java em vez de lidar diretamente com os *sockets*.

O servidor armazena as seguintes informações sobre livros:

- ISBN;
- Título;
- Descrição;
- Estoque;
- Autor;
- Editora;
- Ano;

O cliente pode efetuar três tipos de operações: listar todos os livros (o cliente pode escolher entre receber todos os dados de cada livro ou somente ISBN e título), obter dados de um livro cujo ISBN é conhecido (pode pedir a descrição, o número de livros em estoque ou todos os dados do livro) ou alterar o número de unidades em estoque de um livro cujo ISBN seja conhecido.

## 2 Implementação

A implementação do sistema de livreria neste projeto foi bastante semelhante à implementação dos projetos anteriores, porém dessa vez aproveitando o nível de abstração mais alto da linguagem Java. Os códigos os códigos estão divididos em código do cliente, do servidor e da interface para as diferentes operações com a biblioteca.

A interface remota `Biblio` define os métodos que poderão ser invocados remotamente pelo cliente e as exceções remotas que estes retornam caso necessário. Suas implementações são realizadas no código do servidor, e o cliente faz as chamadas de métodos.

O pré-processamento utilizado na base de dados da livraria foi bem semelhante ao das implementações em C: o arquivo `.txt` com os dados dos livros é lido pelo servidor, que armazena os dados de cada livro em um `ArrayList` para cada categoria das elencadas anteriormente. Dessa forma, não é necessária uma nova consulta ao banco de dados a cada operação requisitada pelo cliente.

Após o pré-processamento, o servidor fica no aguardo das chamadas remotas, escutando um *stub* criado a partir da declaração da interface remota. Quando o cliente faz chamadas de métodos, o servidor executa suas implementações dos métodos da interface, enviando resposta de tipo adequado. Ao final da execução de um método, o servidor imprime o tempo decorrido entre o recebimento por ele da chamada e o fim da execução do método.

O cliente, ao ser inicializado, fornece ao usuário um menu com as funcionalidades disponíveis ao programa, elencadas na Introdução e na especificação do projeto. O usuário seleciona uma opção e fornece os parâmetros requeridos pela operação. Com base na opção selecionada e em eventuais parâmetros, o cliente faz uma chamada a um método remoto, por meio de um *stub* pelo qual ele pode chamar os métodos da interface. Ao receber a resposta, o cliente imprime o tempo decorrido entre o envio da solicitação e o recebimento do retorno.

Para que a comunicação entre cliente e servidor funcione, é necessário executar o registro de objetos remotos do RMI, a partir do comando `rmiregistry`. É graças a ele que o cliente consegue obter a referência aos objetos remotos.

### 3 Vantagens e Desvantagens da Solução

O pré-processamento pareceu uma opção viável, uma vez que não seria necessário lidar com um número grande de livros. Assim, o consumo de memória é relativamente pequeno e há um ganho em desempenho. Ao final de sua execução, o servidor salva seus dados em disco, efetivamente atualizando a versão permanente do banco de dados. Como a chance de panes no servidor é relativamente baixa, os riscos associados a essa escolha são poucos, mas um sistema mais robusto salvaria os dados de uma maneira mais eficiente. O uso de um sistema gerenciador de banco de dados facilitaria o tratamento desse risco.

O emprego de constructos com maior nível de abstração simplificou bastante a implementação do código. Com o uso de RMI, é possível abstrair a rede e chamar um método implementado no servidor como se fosse um método local. A implementação com `sockets`, por outro lado, é mais baixo nível e exigiu a criação de protocolos específicos para a nossa aplicação para que cliente e servidor se comunicassem. Porém, isso se refletiu em um *overhead* maior, tanto no código do cliente e do servidor em comparação com suas versões em C quanto na comunicação em RMI em comparação com os *sockets*.

Outra vantagem em relação ao UDP é o fato de o RMI usar comunicação confiável (TCP/IP).

Não foram adotadas medidas de segurança para o acesso com senha à base de dados. Também no aspecto da segurança, decidiu-se não usar um `SecurityManager` no código do servidor, que serviria para a proteção dos recursos de sistema. Além de apresentar um risco de segurança, isso também resulta em uma restrição das funcionalidades disponíveis, devido às medidas de segurança adotadas pelo RMI.

### 4 Resultados

Os resultados estão presentes na tabelas anexas a este documento. Para todas as 6 operações, as medições foram obtidas a partir da realização de 50 requisições sequenciais originadas por um único cliente.

O tempo de consulta é o tempo decorrido entre imediatamente após o servidor receber a requisição e imediatamente antes de ele enviar a mensagem com a resposta. O tempo total é medido desde antes do

envio da mensagem pelo cliente até depois do recebimento da resposta por ele, e o tempo de comunicação é metade da diferença entre esses tempos.

Como nos projetos 1 e 2, o tempo de comunicação representou a maior parte do tempo total de atividade do cliente. Para a maioria das operações, o tempo médio de comunicação foi em torno de uma ordem de grandeza maior do que os verificados com a manipulação direta de *sockets*, exceto para a operação 5, em que o tipo de retorno definido pela interface é `void`; nesse caso, a implementação em RMI apresentou vantagem.

O tempo de consulta da implementação em Java foi superior em todos os casos, graças às diferenças entre o nível de abstração em relação ao código em C. Porém, a diferença não foi o bastante para compensar o impacto no tempo total da operação 5 da comunicação significativamente mais ágil proporcionada pelo RMI nesse caso.

## 5 Conclusão

Uma vez que os testes foram realizados na mesma rede e aproximadamente no mesmo horário, podemos concluir que a maior parte da diferença nos tempos de comunicação se deve às diferenças entre os protocolos, já que os níveis de uso da rede são aproximadamente os mesmos.

A diferença entre os tempos de comunicação com o UDP e com o TCP fica mais pronunciada com o aumento do tamanho da mensagem de retorno. Ainda assim, quando há efetivamente algo a se retornar, a comunicação por meio de *sockets* é significativamente mais rápida do que o uso de RMI.

Desta forma, o experimento permitiu ao grupo a visualização clara do *trade-off* entre abstração e desempenho: adicionar mais camadas de abstração facilita o trabalho de desenvolvimento e implementação dos programas, mas com um impacto na velocidade de funcionamento e conexão dos programas. Usar a invocação de métodos remotos simplifica a implementação de código para a comunicação entre clientes e servidores, mas resulta em desempenho menor que aquele que pode ser obtido ao lidar com código de mais baixo nível.

## 6 Referências

SUN. Trail: RMI. *The Java Tutorials*. Disponível em <<http://docs.oracle.com/javase/tutorial/rmi/index.html>>.

### Anexo 1: Código do Servidor

```
package example.biblio;

import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

import java.util.ArrayList;
import java.util.Arrays;

import java.io.File;
import java.io.FileNotFoundException;
```

```

import java.util.Scanner;

public class Server implements Biblio {

    public Server() {
        preparaDados();
    }

    //ArrayLists armazenarao banco de dados da biblioteca
    private ArrayList<String> ISBN = new ArrayList<String> ();
    private ArrayList<String> titulo = new ArrayList<String> ();
    private ArrayList<String> descricao = new ArrayList<String> ();
    private ArrayList<Integer> estoque = new ArrayList<Integer> ();
    private ArrayList<String> autor = new ArrayList<String> ();
    private ArrayList<String> editora = new ArrayList<String> ();
    private ArrayList<Integer> ano = new ArrayList<Integer> ();

    //variaveis para contagem de tempo
    long inicio = 0, fim = 0;
    double sec;

    /*Pre-processamento: Os dados do banco serao armazenados na
    nas 7 ArrayList declaradas acima*/
    private void preparaDados(){
        File file = new File("dados.txt");
        try{
            Scanner sc = new Scanner(file);
            while (sc.hasNextLine()) {
                ISBN.add(sc.nextLine());
                titulo.add(sc.nextLine());
                descricao.add(sc.nextLine());
                estoque.add(Integer.parseInt(sc.nextLine()));
                autor.add(sc.nextLine());
                editora.add(sc.nextLine());
                ano.add(Integer.parseInt(sc.nextLine()));
            }
            sc.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }

    //Operacao 1
    public String listaISBN() {
        //obtem tempo inicial
    }
}

```

```

    inicio = System.nanoTime();

    //processa requisicao
    String response = Arrays.toString(ISBN.toArray());

    //obtem tempo final e imprime
    fim = System.nanoTime();
    sec = fim-inicio;
    sec /= 1000000000;
    System.out.print("Tempo da operacao: ");
    System.out.printf("%.9f\n", sec);

    return response;
}

//Operacao 2
public String retornaDescricao(String isbn) {
    //obtem tempo inicial
    inicio = System.nanoTime();

    //processa requisicao
    String response = new String();
    for (int i = 0; i < ISBN.size(); i++) {
        String str = ISBN.get(i);
        if(str.compareTo(isbn) == 0){
            response = descricao.get(i);
        }
    }

    //obtem tempo final e imprime
    fim = System.nanoTime();
    sec = fim-inicio;
    sec /= 1000000000;
    System.out.print("Tempo da operacao: ");
    System.out.printf("%.9f\n", sec);

    return response;
}

//Operacao 3
public String retornaInfo(String isbn) {
    //obtem tempo inicial
    inicio = System.nanoTime();

    //processa requisicao

```

```

String response = new String();
for (int i = 0; i < ISBN.size(); i++) {
    String str = ISBN.get(i);
    if(str.compareTo(isbn) == 0){
        response = isbn + "\n";
        response += titulo.get(i) + "\n";
        response += descricao.get(i) + "\n";
        response += estoque.get(i) + "\n";
        response += autor.get(i) + "\n";
        response += editora.get(i) + "\n";
        response += ano.get(i);
    }
}

//obtem tempo final e imprime
fim = System.nanoTime();
sec = fim-inicio;
sec /= 1000000000;
System.out.print("Tempo da operacao: ");
System.out.printf("%.9f\n", sec);

return response;
}

//operacao 4
public String retornaTudo() {
    //obtem tempo inicial
    inicio = System.nanoTime();

    //processa requisicao
    String response = new String();
    for (int i = 0; i < ISBN.size(); i++) {
        response += ISBN.get(i) + "\n";
        response += titulo.get(i) + "\n";
        response += descricao.get(i) + "\n";
        response += estoque.get(i) + "\n";
        response += autor.get(i) + "\n";
        response += editora.get(i) + "\n";
        response += ano.get(i) + "\n";
    }

    //obtem tempo final e imprime
    fim = System.nanoTime();
    sec = fim-inicio;
    sec /= 1000000000;

```

```

        System.out.print("Tempo da operacao: ");
        System.out.printf("%.9f\n", sec);

        return response;
    }

    //operacao 5
    public void alteraEstoque(String isbn, int valor) {
        //obtem tempo inicial
        inicio = System.nanoTime();

        //processa requisicao
        String response = new String();
        for (int i = 0; i < ISBN.size(); i++) {
            String str = ISBN.get(i);
            if(str.compareTo(isbn) == 0){
                estoque.set(i, valor);
            }
        }

        //obtem tempo final e imprime
        fim = System.nanoTime();
        sec = fim-inicio;
        sec /= 1000000000;
        System.out.print("Tempo da operacao: ");
        System.out.printf("%.9f\n", sec);
    }

    //operacao 6
    public int retornaEstoque(String isbn) {
        int estoqueVal = 0;
        //obtem tempo inicial
        inicio = System.nanoTime();

        //processa requisicao
        String response = new String();
        for (int i = 0; i < ISBN.size(); i++) {
            String str = ISBN.get(i);
            if(str.compareTo(isbn) == 0){
                estoqueVal = estoque.get(i);
            }
        }

        //obtem tempo final e imprime

```

```

        fim = System.nanoTime();
        sec = fim-inicio;
        sec /= 1000000000;
        System.out.print("Tempo da operacao: ");
        System.out.printf("%.9f\n", sec);

        return estoqueVal;
    }

    public static void main(String args[]) {

try {
    Server obj = new Server();
    Biblio stub = (Biblio) UnicastRemoteObject.exportObject(obj, 0);

    // Bind the remote object's stub in the registry
    Registry registry = LocateRegistry.getRegistry();
    registry.bind("Biblio", stub);
    System.err.println("Server ready");
} catch (Exception e) {
    System.err.println("Server exception: " + e.toString());
    e.printStackTrace();
}

    }
}

```

## Anexo 2: Código do Cliente

```

package example.biblio;

import example.biblio.Biblio;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Scanner;
import java.util.InputMismatchException;

public class Client {

    private Client() {}

    public static void main(String[] args) {

String host = (args.length < 1) ? null : args[0];
try {

```



```

Registry registry = LocateRegistry.getRegistry(host);
Biblio stub = (Biblio) registry.lookup("Biblio");

//loop de leitura do teclado e processamento das requisicoes
while(true){
    System.out.println("1- listar todos os ISBN e seus respectivos titulos");
    System.out.println("2- dado o ISBN de um livro, retornar descricao");
    System.out.println("3- dado o ISBN de um livro, retornar todas as informacoes do");
    System.out.println("4- listar todas as informacoes de todos os livros");
    System.out.println("5- alterar numero de exemplares em estoque");
    System.out.println("6- dado o ISBN de um livro, retornar numero de exemplares em");
    System.out.println("7- fechar cliente");

    Scanner sc = new Scanner(System.in);
    int opt = 0;
    try{
        System.out.println("Digite o numero da operacao");
        opt = sc.nextInt();
    } catch (InputMismatchException e) {
        System.out.println("Digite o numero da operacao");
    }

    String response = new String(); //resposta do servidor

    //contagem de tempo
    long inicio = 0, fim = 0;
    double sec;

    //operacao 1
    if(opt == 1){
        //contagem de tempo e chamada
        inicio = System.nanoTime();
        response = stub.listaISBN();
        fim = System.nanoTime();
    }

    //operacao2
    if(opt == 2){
        System.out.println("Digite o ISBN do livro");
        String isbn = sc.next();
        //contagem de tempo e chamada
        inicio = System.nanoTime();
        response = stub.retornaDescricao(isbn);
        fim = System.nanoTime();
    }
}

```

```

        //operacao 3
if(opt == 3){
    System.out.println("Digite o ISBN do livro");
    String isbn = sc.next();
    //contagem de tempo e chamada
    inicio = System.nanoTime();
    response = stub.retornaInfo(isbn);
    fim = System.nanoTime();
}

//operacao 4
if(opt == 4){
    //contagem de tempo e chamada
    inicio = System.nanoTime();
    response = stub.retornaTudo();
    fim = System.nanoTime();
}

//operacao 5
if(opt == 5){
    System.out.println("Digite a senha");
    String senha = sc.next(); // verifica se cliente tem autorizacao
    if(senha.equals("senhalivraria")){
        System.out.println("Digite o ISBN do livro");
        String isbn = sc.next();
        System.out.println("Informe o novo valor do estoque");
        int valor = sc.nextInt();
        //contagem de tempo e chamada
        inicio = System.nanoTime();
        stub.alteraEstoque(isbn, valor);
        fim = System.nanoTime();
    }
    else {
        System.out.println("Senha invalida.");
    }
}

//operacao 6
if(opt == 6){
    System.out.println("Digite o ISBN do livro");
    String isbn = sc.next();
    //contagem de tempo e chamada
    inicio = System.nanoTime();
    response += stub.retornaEstoque(isbn);
}

```

```

        fim = System.nanoTime();
    }

    //opcao para cliente sair da aplicacao
    if(opt == 7){
        System.exit(0);
    }

    //imprime tempo total
    System.out.println(response);
    sec = fim-inicio;
    sec /= 1000000000;
    System.out.print("Tempo da operacao: ");
    System.out.printf("%.9f\n", sec);
}

} catch (Exception e) {
    System.err.println("Client exception: " + e.toString());
    e.printStackTrace();
}

}

}

```

### Anexo 3: Código da Interface

```

package example.biblio;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Biblio extends Remote {
    String listaISBN() throws RemoteException;
    String retornaDescricao(String isbn) throws RemoteException;
    String retornaInfo(String isbn) throws RemoteException;
    String retornaTudo() throws RemoteException;
    void alteraEstoque(String isbn, int valor) throws RemoteException;
    int retornaEstoque(String isbn) throws RemoteException;
}

```