PÓS-GRADUAÇÃO ALFA

FACULDADES ALFA ALVES ARIA

A MELHOR ESCOLA DE NEGÓCIOS DO CENTRO-OESTE

Diego Am érico Guedes

PROGRAMAÇ ÃO COM FRAMEWORKS E COMPONENTES

GRUPO JOSÉ ALVES

Apresentaç ão

- Bacharel em Ciências da Computação, UFG (2010)
- Mestrado em Ciências da Computação, UFG (2013)
- Campeão das regionais da Maratona de Programação em 2008 e 2009
- Trabalhei em projetos regionais, nacionais e mundiais de pesquisa na área de Ciências da Computação
- Fui professor substituto na UFG por 2 anos (2013-2014)
- Fui professor Adjunto na Faculdade Senac por 2 anos (2015-2016)
- P&D goGeo (2014)
- Analista de Sistemas na Saneago (Desde 2014)

Chamada Remota de Procedimento (RPC)

- Etapas
  - O cliente chama um procedimento local, a função *stub*.
  - A função *stub* do cliente empacota os argumentos e constrói a mensagem a ser enviada pela rede (*marshalling*)
  - A função *stub* faz uma chamada ao sistema (socket) para enviar a mensagem para o outro processo
  - A função *stub* do servidor recebe a mensagem e desempacota os parâmetros (*unmarshalling*)
  - A função *stub* do servidor chama o procedimento local
  - A função *stub* do servidor empacota o resultado e faz uma chamada ao sistema para enviar o resultado para o outro processo
  - A função *stub* do cliente recebe a mensagem do servidor, desempacota e retorna para o cliente.
  - O cliente continua sua execução

Objetos Distribuídos

- Ferramentas de programação para Sistemas Distribuídos
  - Sockets
  - RPC
  - Objetos Distribuídos

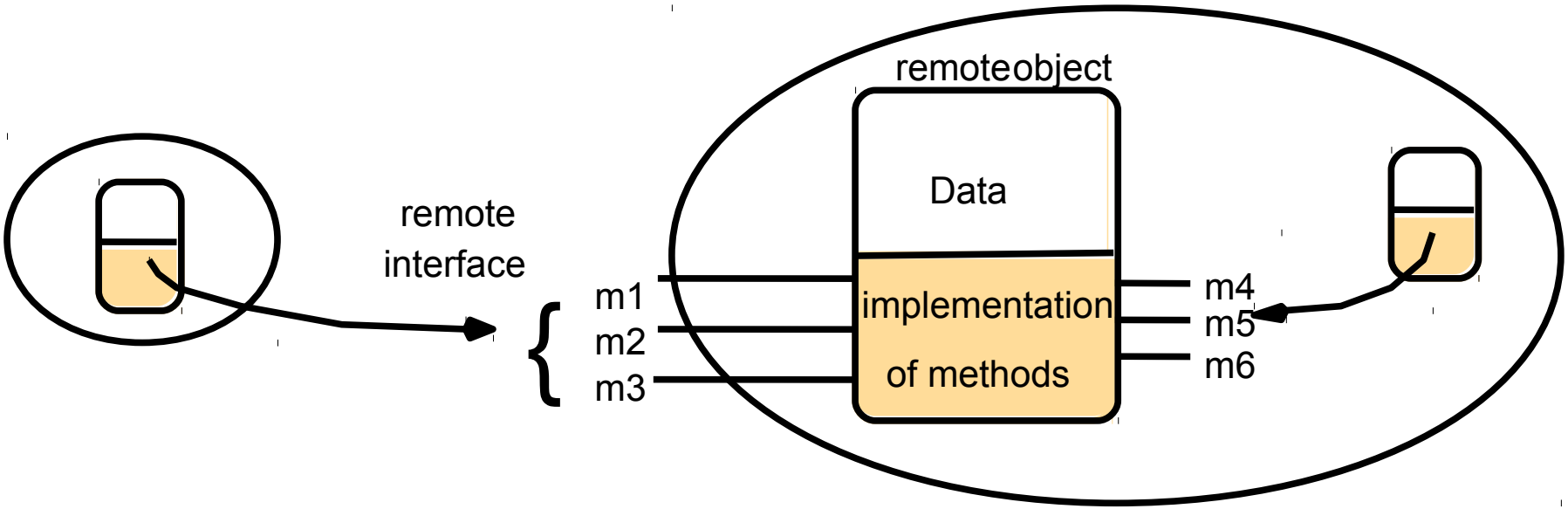
Objetos Distribuídos

- Evolução do conceito de RPC
- Principais características
  - Encapsulamento de implementação
  - Interfaces bem definidas
  - Baixo grau de acoplamento entre os componentes
  - Unidades computacionais que podem ser reutilizadas para compor várias aplicações
- Exemplos:
  - CORBA, JAVA RMI, DCOM

Objetos Distribuídos

- Conceitos Fundamentais
  - Objeto remoto : objeto que implementa um ou mais métodos que podem ser invocados remotamente, ou seja, invocados por outros processos
  - Interface remota : **Todo objeto remoto deve implementar uma interface remota.** A interface remota descreve os métodos do objeto que podem ser invocados remotamente
    - CORBA : interfaces remotas são definidas em IDL
    - RMI Java: interfaces remotas são interfaces Java que estendem a interface **Remote**

Objetos Distribuídos

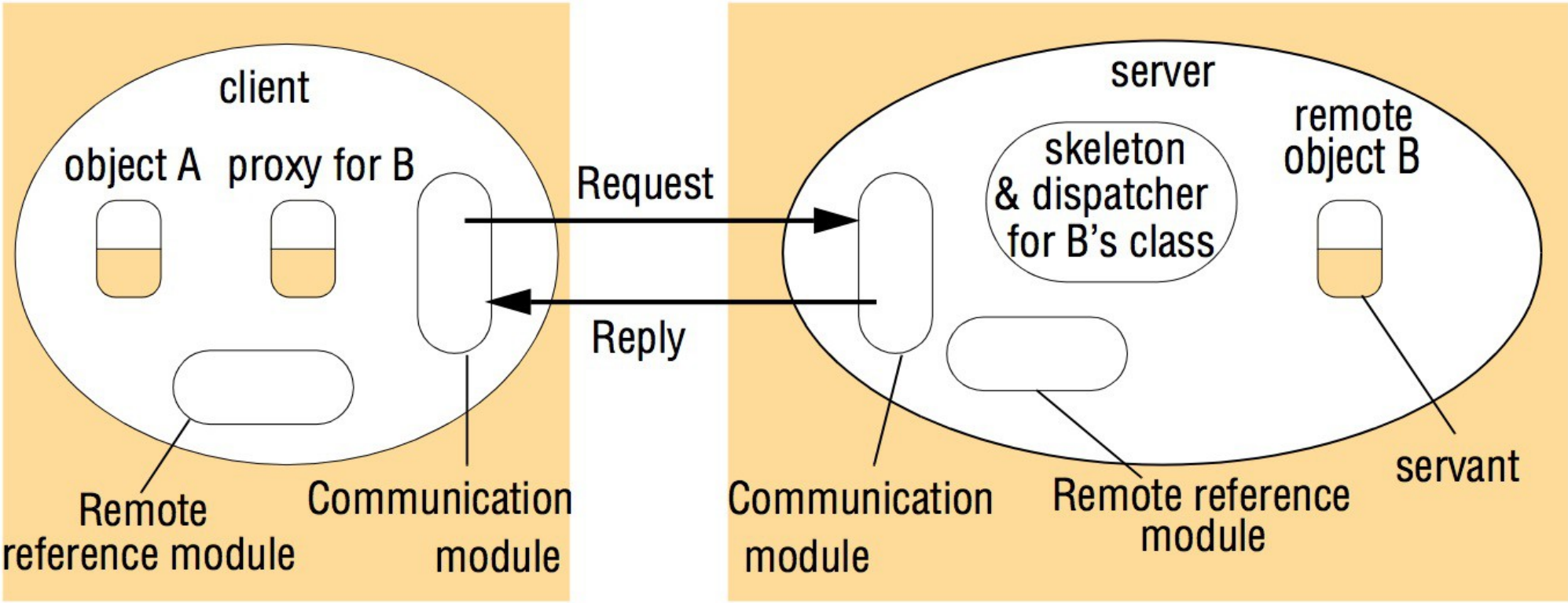


- m1, m2 e m3 podem ser invocados remotamente
- m4, m5 e m6 **não** podem ser invocados remotamente

Objetos Distribuídos

- Conceitos Fundamentais
  - Referência de objeto remoto : é um identificador que pode ser usado por todo o sistema distribuído para se referir a um objeto remoto único em particular.
  - Uma referência de objeto remoto se assemelha a uma referência a objeto local, no sentido que
    - Ela pode ser usada para invocar um método do objeto
    - Ela pode ser passada como argumento ou retorno de métodos

Implementação de Objetos Distribuídos

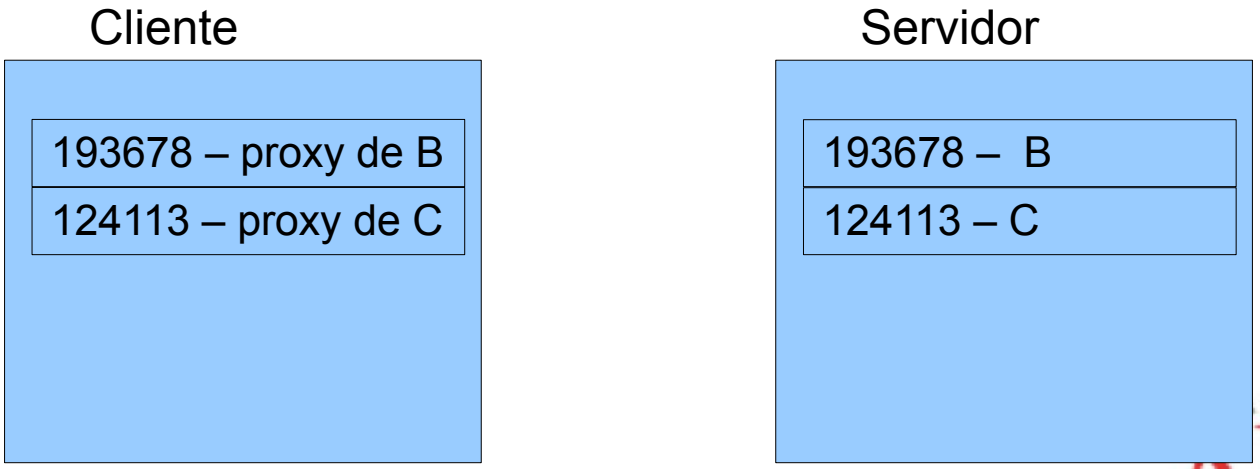


Implementação de Objetos Distribuídos

- Módulo de Comunicação:
  - Cooperam para executar o protocolo requisição-resposta entre o cliente e o servidor.
  - São responsáveis por fornecer uma semântica de invocação específica (talvez, pelo menos uma vez ou no máximo uma vez)
  - No Servidor, seleciona o despachante para a classe do objeto a ser invocado

Implementação de Objetos Distribuídos

- Módulo de referência remota
  - Responsável pela criação das referências de objetos remotas
  - Responsável também pela conversão de referencias de objetos remotas em referencias de objetos locais



Implementação de Objetos Distribuídos

- Servent
  - Objeto de uma classe que implementa uma interface remota
  - Trata as requisições remotas repassadas pelo esqueleto
- Proxy
  - Comporta-se como um objeto local para o invocador
  - Em vez de executar uma invocação local, encaminha uma mensagem para o objeto remoto, que efetivamente implementa o método invocado
  - Responsável pelo marshalling de parâmetros e unmarshalling do resultado
  - Existe um proxy para cada objeto remoto

Implementação de Objetos Distribuídos

- Despachante
  - Recebe a mensagem de requisição do módulo de comunicação e seleciona o método apropriado no esqueleto, repassando a mensagem a esse
- Esqueleto
  - Módulo que simula o objeto remoto no lado do servidor.
  - Implementa o unmarshalling dos argumentos da chamada remota e invoca o método correspondente no Servent.
  - executa o marshalling do resultado em uma mensagem de resposta.
  - Existe um Despachante e um Esqueleto para cada classe que implementa uma interface remota

Geração de proxy, despachante e esqueleto

- CORBA
  - Programador escreve a IDL, o programa cliente e o programa servidor
  - O compilador IDL gera as classes para o proxy, despachante e esqueleto em uma linguagem específica (C++, Java, Lua)
- RMI Java
  - Compilador RMI Java gera as classes de proxy, despachante e esqueleto a partir da classe do objeto remoto

Java RMI

RMI - Invocação de Objetos Remotos

- Java RMI (Remote Method Invocation), é um mecanismo que permite ao usuário, criar aplicações distribuídas utilizando Java.

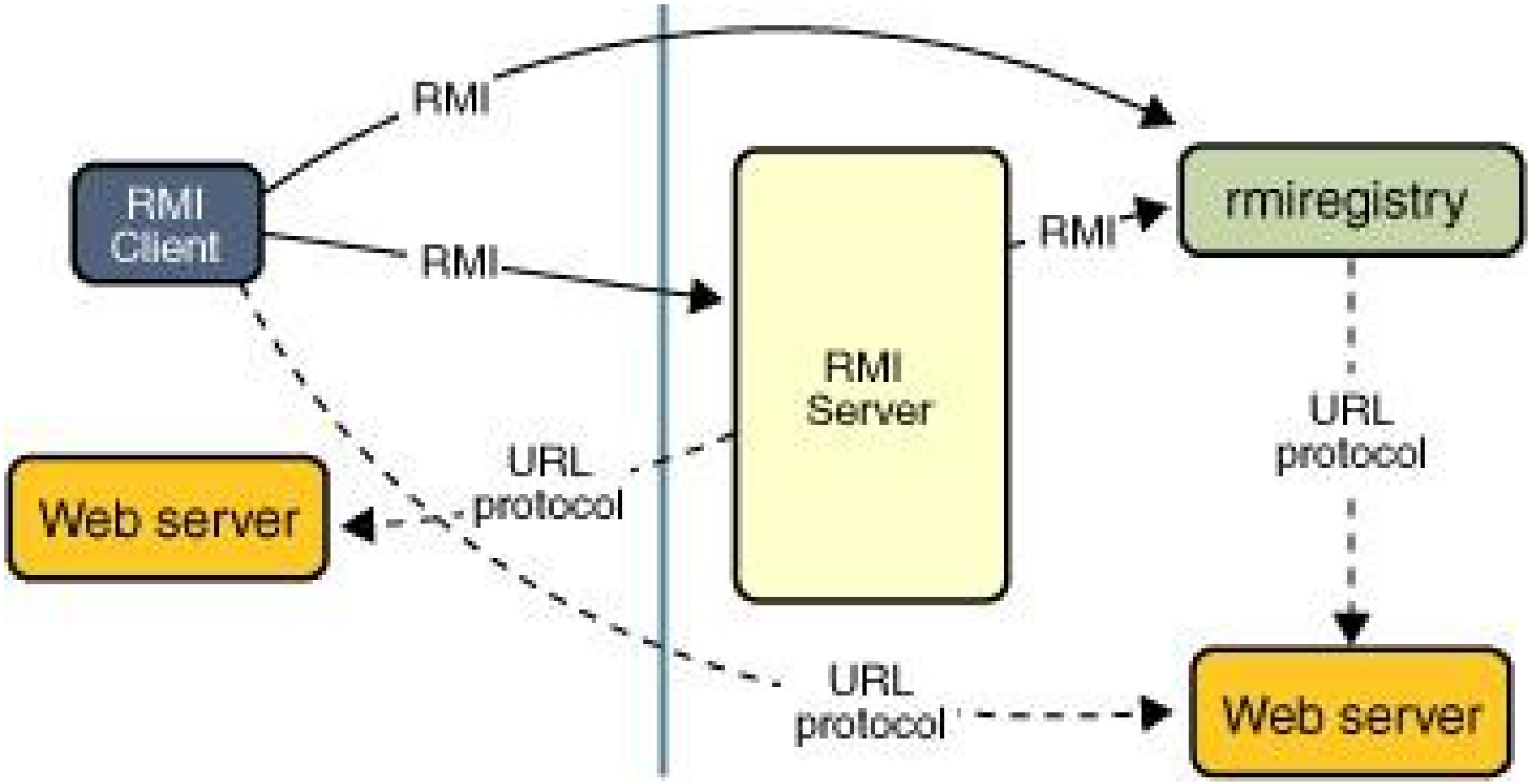
RMI - Invocação de Objetos Remotos

- As aplicações do RMI compreendem frequentemente dois programas separados, um servidor e um cliente.
- Um programa servidor cria alguns objetos remotos, faz referência a esses objetos, e aguarda os “clients” invocarem os métodos desses objetos.
- Enquanto que um programa cliente obtém uma referência remota ao objetos criados pelo servidor e invoca os métodos desses objetos.

Funcionamento do RMI

- O servidor chama o “registry” para associar (bind) um nome com um objeto remoto.
- O cliente olha o objeto remoto por seu nome no “registry” do servidor, e invoca então um método nele.

Funcionamento do RMI



Criando aplicações distribuídas usando o RMI

Utilizar RMI para desenvolver uma aplicação distribuída envolve estas etapas gerais:

1. Projetando e implementando os componentes de sua aplicação distribuída.
2. Compilando o código fonte.
3. Fazendo com que as classes sejam acessíveis via rede.
4. Iniciando a aplicação.

Iremos criar uma classe servidora que irá conter um método que será invocado remotamente através da utilização de RMI, para isso iremos precisar de alguns arquivos:

- Ola.java (interface que será implementada tanto pelo server quando pelo client)
- OlaImpl.java (implementação do servidor)
- Cliente.java (client que fará uso de métodos do OlaImpl - server)

Ola.java

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


public interface Ola extends Remote {
                String showMsg(String msg) throws
RemoteException;
}
```

OlaImpl.java

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class OlaImpl extends UnicastRemoteObject implements Ola {
    public OlaImpl() throws RemoteException {  super();  }
    public String showMsg(String msg) {
        System.out.println("msg: " + msg);
        return("msg enviada"); }
    public static void main(String args[]) {
     try { OlaImpl obj = new OlaImpl();
            Naming.rebind("OlaServidor", obj);
            System.out.println("Servidor carregado no registry");
        } catch (Exception e) {
            System.out.println("OlaImpl erro: " + e.getMessage());
        }}}
```

Client.java

```
import java.rmi.Naming;
public class Client {
    public static void main(String[] args) {
        Ola obj = null;
        String msg = "minha mensagem";
        String retorno = null;
        try {
            obj = (Ola)Naming.lookup("//127.0.0.1/OlaServidor");
            retorno = obj.showMsg(msg);
            System.out.println(retorno);
        }
        catch (Exception e) {
            System.out.println("Client exception: " + e.getMessage());
        } } }
```

Compilando o código fonte.

Compilar os códigos fonte, para isso basta digitar:

javac Ola.java

javac OlaImpl.java

javac Client.java

Fazendo com que as classes sejam acessíveis via rede

Para que as classes possam ser acessadas via rede, precisamos inicializar o servidor de rmi através do comando:

- start rmiregistry – no windows
- rmiregistry – no linux e MacOS

```
diego:rmi diego$ rmiregistry
```

Iniciando a aplicação

- Em um terminal iniciamos o servidor:

java OlaImpl

```
diego:rmi diego$ java OlaImpl
Servidor carregado no registry
msg: minha mensagem
```

- Em outro terminal iniciamos o cliente

java Client

```
diego:rmi diego$ java Client
msg enviada
diego:rmi diego$
```

Registry no código

- Pode-se colocar o registry no código.
  - Por padrão, a porta do RMI é 1099
    - **No Cliente**

```
Registry registry = LocateRegistry.createRegistry(2001);
registry.rebind("OlaServidor", obj);
```

    - **No Cliente**

```
Registry registry = LocateRegistry.getRegistry("127.0.0.1", 2001);
obj = (Ola) registry.lookup("OlaServidor");
```

Parâmetro Objeto

- Para passar objetos, tem alguma mudança?

Parâmetro Objeto

- Para passar objetos, tem alguma mudança?
  - Sim. No RMI, como os dados serão transmitidos pela rede, o objeto passado como parâmetro deve ter a capacidade de ser serializado
  - Assim, deve implementar (implements) a interface Serializable.
  - É uma boa prática (as vezes não funciona) se a classe não ter um serialVersionUID

Observações sobre o RMI

O RMI permite que vários “Objetos Clientes” se conectem a um “Objeto Servidor”

- O mesmo “Objeto Servidor” irá atender a todos os clientes;
- Não é multi-thread;
- Os clientes compartilham o mesmo servidor.

Observações sobre o RMI

Não é possível registrar em um determinado registry um objeto que esteja em outra máquina

- Is it possible to run rmiregistry on a different machine than the remote object?
- http://www.jguru.com/faq/view.jsp?EID=417817

Observações sobre o RMI

Quando houver problemas de conexão em relação ao endereço publicado pelo servidor (Connection refused to host: 127.0.1.1) deve-se passar a propriedade java.rmi.server.hostname para a máquina virtual Java:

```
java -Djava.rmi.server.hostname=192.168.0.116 LigadorImpl

java -Djava.rmi.server.hostname=192.168.0.117 ServidorTempoImpl 192.168.0.116
```

- http://stackoverflow.com/questions/2624752/starting-rmi-server-on-ubuntu-laptop
- Se essa propriedade não for especificada no Ubuntu, será publicado o endereço 127.0.1.1 (ver arquivo /etc/hosts).
- Já no Windows, o IP da máquina para acesso remoto foi publicado corretamente nos testes que fiz, não precisando definir a propriedade java.rmi.server.hostname.

Observações sobre o RMI

Se for usado algum SecurityManager é preciso criar um arquivo que determina a política de segurança e dizer isso para o RMI através da propriedade java.security.policy:

```
java -Djava.rmi.server.hostname=192.168.0.116
-Djava.security.policy=security.policy LigadorImpl
```

Observações sobre o RMI

Um exemplo de arquivo security.policy que garante todas as permissões é dado abaixo:

```
grant {
  permission java.security.AllPermission "", "";
};
```

Entretanto, se for para dar todas as permissões, é melhor não usar nenhum SecurityManager.