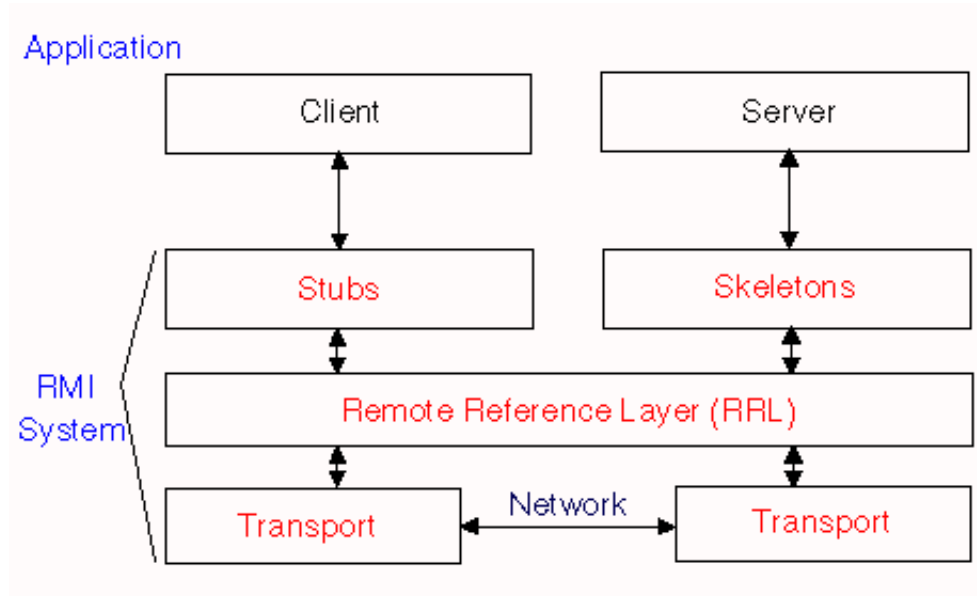


*Licenciatura em Tecnologias da Informação*  
**Tecnologias de Integração**  
Folha de trabalho - RMI

O Java Remote Method Invocation (RMI) é uma tecnologia constituída pelas seguintes camadas:



Fonte: <http://www.edm2.com/0601/rmi1.html>

- **Stub e Skeleton**  
Interfaces entre as aplicações e o sistema RMI
- **Remote Reference Layer**  
Disponibiliza os meios para permitir a invocação remota de objetos
- **Transport Layer**  
Gere e monitoriza as ligações entre as aplicações cliente e servidor.

## Implementação em Java

A módulo `java.rmi` contém as packages e classes necessárias para a utilização da tecnologia. Duas das classes são:

- `java.rmi.registry` - Package que contém classes e interfaces para o registo remoto de objetos
- `java.rmi.server` - Package onde se encontram as classes e interfaces que permitem a implementação dos stubs e skeletons e as que dão suporte à comunicação

## Serviço Remoto

### Processo para criar o serviço remoto

1. Criar a interface remota
  - Define os métodos que o cliente pode invocar remotamente.
  - Deve estar acessível para o servidor e para o cliente.
  - O stub e o skeleton implementam esta interface.
2. Criar a implementação remota
  - Implementa o(s) método(s) a ser(em) invocado(s) remotamente
3. Criar o servidor
  - Regista o serviço no rmiregistry.

#### Criar interface remota

1. Estender a interface java.rmi.Remote

```
1 public interface MyRemote extends Remote
```

2. Declarar em todos os seus métodos que enviam uma RemoteException

```
1 import java.rmi.*;
   public interface MyRemote extends Remote {
3     public String sayHello() throws RemoteException;
   }
```

3. Assegurar que os argumentos de entrada e valores retornados são primitivos ou serializáveis

#### Criar implementação remota

1. Implementar a interface remota

```
   public class MyRemoteImpl implements MyRemote {
2     public String sayHello() throws RemoteException {
        return "Server says: 'Hello'";
4     }
   }
```

2. Estender UnicastRemoteObject

```
1 public class MyRemoteImpl extends UnicastRemoteObject
   implements MyRemote
```

3. Implementar um construtor sem argumentos e que declare uma RemoteException

```
public MyRemoteImpl() throws RemoteException {}
```

### Criar servidor

1. Inicializar o rmiregistry

```
1 try {  
    LocateRegistry.createRegistry(1099);  
3 } catch (Exception e) {...}
```

2. Registrar o serviço no rmiregistry

```
1 try {  
    MyRemote service = new MyRemoteImpl();  
3    Naming.rebind("RemoteHello", service);  
} catch (Exception e) {...}
```

### Cliente

1. Procurar o objeto stub

```
MyRemote service = (MyRemote)  
2    Naming.lookup("rmi://127.0.0.1/RemoteHello");
```

2. O registo RMI retorna o objeto stub

3. Invocar o método no stub

```
String s = service.sayHello();
```

### Exercícios

1. Pretende-se que crie duas aplicações. Uma aplicação cliente que deverá enviar uma mensagem com o nome do utilizador. A segunda, o servidor, deverá responder com uma saudação que inclua o nome do utilizador.
2. Pretende-se que crie duas aplicações. A aplicação cliente deve solicitar a data e hora indicando o formato com que o pretende receber. A aplicação servidor deverá receber o pedido, com o formato pretendido e responder com a data e hora atual no formato pretendido.
3. Pretende-se que crie duas aplicações. A aplicação servidor deverá realizar alguns cálculos matemáticos mais complexos, por exemplo: calcular o factorial de um número; calcular o maior número primo entre dois valores; calcular o menor número primo entre dois valores; calcular a soma de um número indefinido de parcelas; .... A aplicação cliente deverá solicitar o cálculo pretendido.
4. Pretende-se que crie duas aplicações. A aplicação servidor deve realizar o cálculo de diferentes índices de saúde, por exemplo: índice de massa corporal; .... A aplicação cliente deverá solicitar o cálculo do índice pretendido.

5. Pretende-se que crie duas aplicações. A aplicação servidor deverá recorrer a uma base de dados com a informação sobre alunos, disciplinas e cursos, e disponibilizar um conjunto de funcionalidades que permitam a gestão de alunos de uma instituição de ensino. A aplicação cliente deverá interagir com a aplicação servidor para consultar e/ou atualizar dados.