



INSTITUTO POLITÉCNICO NACIONAL ESCUELA SUPERIOR DE CÓMPUTO

Sistemas Distribuidos

Prof. Carlos Pineda Guerrero

Alejandro de Jesús Zepeda Flores

Tarea 6. Multiplicación de matrices utilizando objetos distribuidos

19 de noviembre de 2020

Requisitos

Desarrollar un sistema que calcule el producto de 2 matrices cuadradas utilizando Java RMI, tal como se explicó en clase. Se deberá ejecutar 2 casos:

- $N = 4$, desplegar las matrices A, B y C y el checksum de la C.
- $N = 500$, desplegar el checksum de la matriz C.

Los elementos de las matrices serán de tipo int y el checksum long

Se deberá inicializar las matrices A y B de la siguiente manera (notar que la inicialización es diferente a la que se realizó en la tarea 3):

- $A[i][j] = 2 * i - j$
- $B[i][j] = 2 * i + j$

El servidor RMI ejecutará en cuatro máquinas virtuales (nodo 0, nodo 1, nodo 2 y nodo 3) con **Ubuntu** (no se admitirá esta tarea si se usan maquinas virtuales con Windows). El programa rmiregistry ejecutará en cada nodo donde ejecute el servidor RMI.

El cliente RMI, el cual ejecutará en el nodo 0, inicializará las matrices A y B, obtendrá la transpuesta de la matriz B, invocará el método remoto multiplica_matrices(), calculará el checksum de la matriz C, y en su caso ($N=4$) desplegará las matrices A, B y C.

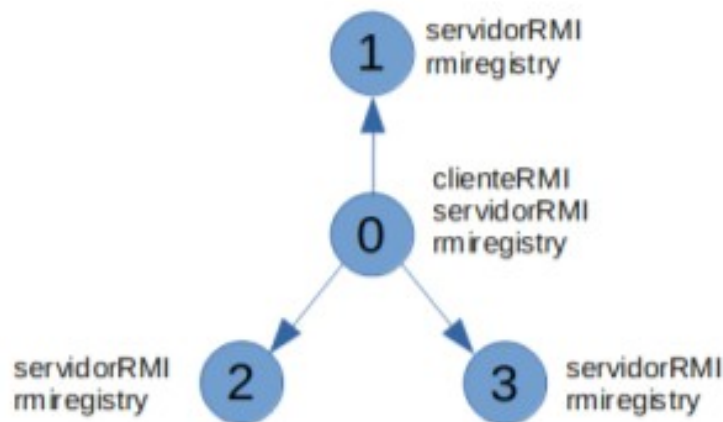


Figura 1. Nodos del programa.

Desarrollo

La interface hereda de la clase Remote y únicamente tiene la definición del método remoto, en este caso, multiplica_matrices; este arroja una excepción del tipo RemoteException.

Nota. Cuando se trabaja con Interfaces de RMI, es recomendable definir los métodos con ese tipo de excepción.

```
01. import java.rmi.*;
02.
03. public interface InterfaceRMI extends Remote{
04.     public int[][] multiplica_matrices(int[][] A,int[][] B) throws RemoteException;
05. }
```

Figura 2. InterfaceRMI.

La clase hereda de UnicastRemoteObject e implementa la interface de la figura 2. Aquí, se implementará el método que definimos en la interface. Además, se utiliza el constructor invoca al constructor de la súper clase.

```
01. import java.rmi.*;
02. import java.rmi.server.*;
03.
04. public class ClaseRMI extends UnicastRemoteObject implements InterfaceRMI{
05.     public int N = 4;
06.
07.     public ClaseRMI() throws RemoteException{
08.         super( );
09.     }
10.
11.     @Override
12.     public int[][] multiplica_matrices(int[][] A,int[][] B) throws RemoteException{
13.         int[][] C = new int[N/2][N/2];
14.         for(int i = 0; i < N/2; i++)
15.             for(int j = 0; j < N/2; j++)
16.                 for(int k = 0; k < N; k++)
17.                     C[i][j] += A[i][k] * B[j][k];
18.         return C;
19.     }
20. }
```

Figura 3. ClaseRMI.

El proceso servidor deberá registrar la clase rmi invocando el método rebind(). Al método se le pasa como parámetros la URL correspondiente al objeto remoto y una instancia de la clase **C**. La URL tiene la siguiente forma: **rmi://ip:puerto/nombre**.

```
01. import java.rmi.*;
02.
03. public class ServidorRMI{
04.     public static void main(String args[])throws Exception{
05.         String url = "rmi://localhost/prueba";
06.         ClaseRMI obj = new ClaseRMI();
07.         Naming.rebind(url,obj);
08.     }
09. }
```

Figura 4. ServidorRMI.

```

01. import java.rmi.*;
02. import java.rmi.server.*;
03.
04. public class ClienteRMI{
05.     static int N = 4;
06.
07.     public static void main(String args[]) throws Exception{
08.         String url1 = "rmi://13.85.28.175:1099/prueba";
09.         String url2 = "rmi://13.85.27.169:1099/prueba";
10.         String url3 = "rmi://13.84.213.172:1099/prueba";
11.         String url4 = "rmi://13.84.215.85:1099/prueba";
12.
13.         InterfaceRMI r1 = (InterfaceRMI)Naming.lookup(url1);
14.         InterfaceRMI r2 = (InterfaceRMI)Naming.lookup(url2);
15.         InterfaceRMI r3 = (InterfaceRMI)Naming.lookup(url3);
16.         InterfaceRMI r4 = (InterfaceRMI)Naming.lookup(url4);
17.
18.         int A[][] = new int[N][N];
19.         int B[][] = new int[N][N];
20.         int C[][] = new int[N][N];
21.
22.         for(int i = 0; i<N; i++){
23.             for(int j = 0; j<N; j++){
24.                 A[i][j] = 2*i-j;
25.                 B[i][j] = 2*i+j;
26.             }
27.         }
28.
29.         B = transpuestaB(B);
30.
31.         int A1[][] = parte_matriz(A,0);
32.         int A2[][] = parte_matriz(A,N/2);
33.         int B1[][] = parte_matriz(B,0);
34.         int B2[][] = parte_matriz(B,N/2);
35.
36.         int C1[][] = r1.multiplica_matrices(A1,B1);
37.         int C2[][] = r2.multiplica_matrices(A1,B2);
38.         int C3[][] = r3.multiplica_matrices(A2,B1);
39.         int C4[][] = r4.multiplica_matrices(A2,B2);
40.
41.         C = acomoda_matriz(C,C1,0,0);
42.         C = acomoda_matriz(C,C2,0,N/2);
43.         C = acomoda_matriz(C,C3,N/2,0);
44.         C = acomoda_matriz(C,C4,N/2,N/2);
45.
46.         if(N==4){
47.             mostrarMatriz(A,"Matriz A");
48.             mostrarMatriz(B,"Matriz B");
49.             mostrarMatriz(C,"Matriz C");
50.         }
51.
52.         calcularChecksum(C);
53.     }
54.
55.     public static int[][] acomoda_matriz(int[][] C,int[][] A,int renglon,int columna) throws RemoteException{
56.         for(int i = 0; i<N/2; i++){
57.             for(int j = 0; j<N/2; j++){
58.                 C[i+renglon][j+columna] = A[i][j];
59.             }
60.         }
61.         return C;
62.     }
63.
64.     public static void mostrarMatriz(int matriz[][], String cadena){
65.         System.out.println(cadena+"\n");
66.         for(int i = 0; i<matriz.length; i++){
67.             for(int j = 0; j<matriz[i].length; j++){
68.                 System.out.print(matriz[i][j]+" ");
69.             }
70.             System.out.println("\n");
71.         }
72.     }
73.
74.     public static void calcularChecksum(int[][] M) throws RemoteException{
75.         long checksum = 0;
76.         for(int i = 0; i<N; i++){
77.             for(int j = 0; j<N; j++){
78.                 checksum = checksum + M[i][j];
79.             }
80.             System.out.println("\nResultado del checksum = "+checksum);
81.         }
82.     }
83.
84.     public static int[][] parte_matriz(int[][] A,int inicio) throws RemoteException{
85.         int M[][] = new int[N/2][N];
86.         for(int i = 0; i<N/2; i++){
87.             for(int j = 0; j<N; j++){
88.                 M[i][j] = A[i + inicio][j];
89.             }
90.         }
91.         return M;
92.     }
93.
94.     public static int [][] transpuestaB(int [][] B) throws RemoteException{
95.         for(int i = 0; i < N; i++){
96.             for(int j = 0; j < 1; j++){
97.                 int x = B[i][j];
98.                 B[i][j] = B[j][i];
99.                 B[j][i] = x;
100.             }
101.         }
102.         return B;
103.     }
104. }

```

Figura 5. ClienteRMI.

El cliente RMI, posiblemente sea la clase más importante, ya que tendrá la implementación de los métodos para mostrar, acomodar, partir y transponer la matriz, así como el cálculo del checksum.

El proceso cliente deber invocar el método lookup() de la clase java.rmi.Naming para obtener una referencia al objeto remoto. El método lookup() regresa una instancia de la clase Remote, la cual se debe convertir al tipo de la interface **I** mediante casting. Utilizando la referencia, el proceso cliente invocará los métodos remotos de la clase **C**.

Resultados

Para comenzar, mostramos la tabla de las direcciones IP's.

| Nodo | Usuario | IP |
|------|---------|---------------|
| 0 | nodo0 | 13.84.28.175 |
| 1 | nodo1 | 13.84.27.169 |
| 2 | nodo2 | 13.85.213.172 |
| 3 | nodo3 | 13.85.215.85 |

Tabla 1. Direcciones IP.

```

nodo0@nodo0: ~
# login as: nodo0
# nodo0@13.84.28.175's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1031-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Fri Nov 20 00:33:06 UTC 2020
System load: 0.0          Processes: 108
Usage of /:  4.5% of 28.9GB Users logged in: 0
Memory usage: 20%        IP address for eth0: 10.0.0.4
Swap usage:  0%

0 packages can be updated.
0 updates are security updates.

New release '20.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Fri Nov 20 00:15:29 2020 from 189.241.86.54
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

nodo0@nodo0:~$

nodo1@nodo1: ~
# login as: nodo1
# nodo1@13.84.27.169's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1031-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Fri Nov 20 00:34:00 UTC 2020
System load: 0.08         Processes: 109
Usage of /:  4.5% of 28.9GB Users logged in: 0
Memory usage: 20%        IP address for eth0: 10.0.0.5
Swap usage:  0%

0 packages can be updated.
0 updates are security updates.

New release '20.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

nodo1@nodo1:~$

nodo2@nodo2: ~
# login as: nodo2
# nodo2@13.84.213.172's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1031-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Fri Nov 20 00:35:04 UTC 2020
System load: 0.0          Processes: 111
Usage of /:  4.5% of 28.9GB Users logged in: 0
Memory usage: 20%        IP address for eth0: 10.0.0.6
Swap usage:  0%

0 packages can be updated.
0 updates are security updates.

New release '20.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

nodo2@nodo2:~$

nodo3@nodo3: ~
# login as: nodo3
# nodo3@13.84.215.85's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1031-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Fri Nov 20 00:35:55 UTC 2020
System load: 0.24         Processes: 110
Usage of /:  4.5% of 28.9GB Users logged in: 0
Memory usage: 20%        IP address for eth0: 10.0.0.7
Swap usage:  0%

0 packages can be updated.
0 updates are security updates.

New release '20.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

nodo3@nodo3:~$

```

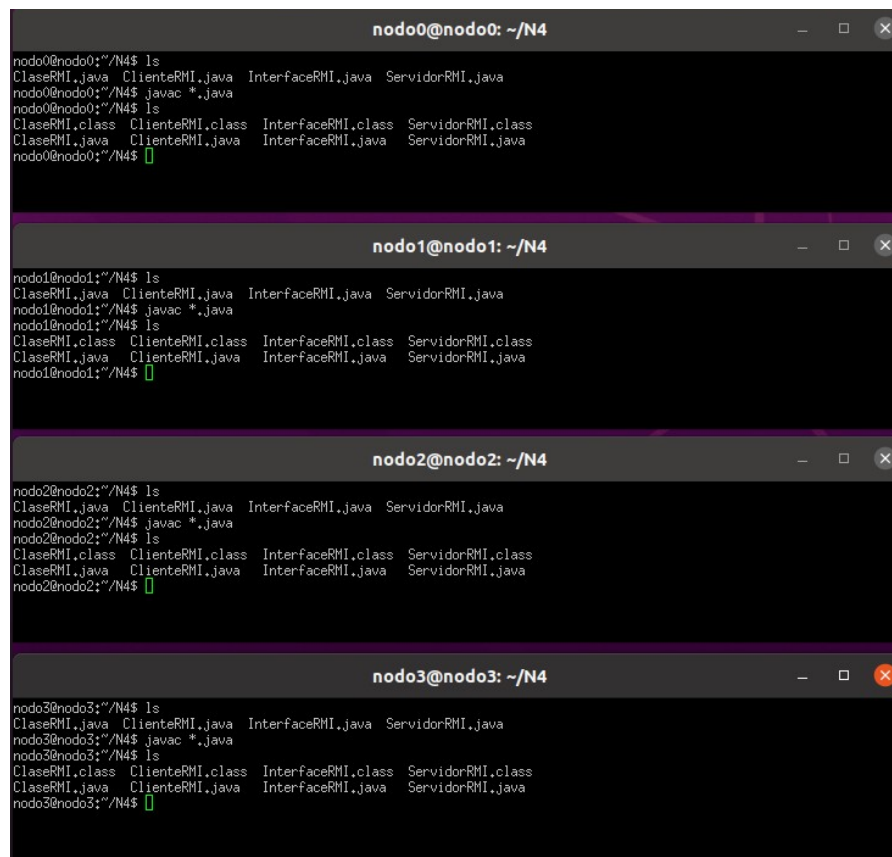
Figura 6. Ubuntu Server.

Uno de los requisitos es que las máquinas virtuales que creamos sean de Linux. En la figura 6, se muestra los 4 nodos y al iniciar sesión, se puede observar la bienvenida de Ubuntu Server.

Ahora, la ejecución del programa se divide en 2 casos:

- $N = 4$

En la siguiente figura, se observa como en cada uno de los nodos, los únicos archivos iniciales son los .java. Con ellos, los compilamos y generamos los ejecutables.



The image displays four terminal windows, each representing a different node in a distributed system. Each window shows the execution of a series of commands to compile Java source files into class files. The commands and their outputs are as follows:

```
nodo0@nodo0: ~/N4
nodo0@nodo0:~/N4$ ls
ClaseRMI.java  ClienteRMI.java  InterfaceRMI.java  ServidorRMI.java
nodo0@nodo0:~/N4$ javac *.java
nodo0@nodo0:~/N4$ ls
ClaseRMI.class  ClienteRMI.class  InterfaceRMI.class  ServidorRMI.class
ClaseRMI.java  ClienteRMI.java  InterfaceRMI.java  ServidorRMI.java
nodo0@nodo0:~/N4$
```

```
nodo1@nodo1: ~/N4
nodo1@nodo1:~/N4$ ls
ClaseRMI.java  ClienteRMI.java  InterfaceRMI.java  ServidorRMI.java
nodo1@nodo1:~/N4$ javac *.java
nodo1@nodo1:~/N4$ ls
ClaseRMI.class  ClienteRMI.class  InterfaceRMI.class  ServidorRMI.class
ClaseRMI.java  ClienteRMI.java  InterfaceRMI.java  ServidorRMI.java
nodo1@nodo1:~/N4$
```

```
nodo2@nodo2: ~/N4
nodo2@nodo2:~/N4$ ls
ClaseRMI.java  ClienteRMI.java  InterfaceRMI.java  ServidorRMI.java
nodo2@nodo2:~/N4$ javac *.java
nodo2@nodo2:~/N4$ ls
ClaseRMI.class  ClienteRMI.class  InterfaceRMI.class  ServidorRMI.class
ClaseRMI.java  ClienteRMI.java  InterfaceRMI.java  ServidorRMI.java
nodo2@nodo2:~/N4$
```

```
nodo3@nodo3: ~/N4
nodo3@nodo3:~/N4$ ls
ClaseRMI.java  ClienteRMI.java  InterfaceRMI.java  ServidorRMI.java
nodo3@nodo3:~/N4$ javac *.java
nodo3@nodo3:~/N4$ ls
ClaseRMI.class  ClienteRMI.class  InterfaceRMI.class  ServidorRMI.class
ClaseRMI.java  ClienteRMI.java  InterfaceRMI.java  ServidorRMI.java
nodo3@nodo3:~/N4$
```

Figura 7. Compilación .java con $N = 4$.

Después, viene la ejecución de `rmiregistry` en cada uno de los nodos en el que se ejecutará el servidor. De igual manera, se tienen que ejecutar los servidores antes de iniciar con el cliente.

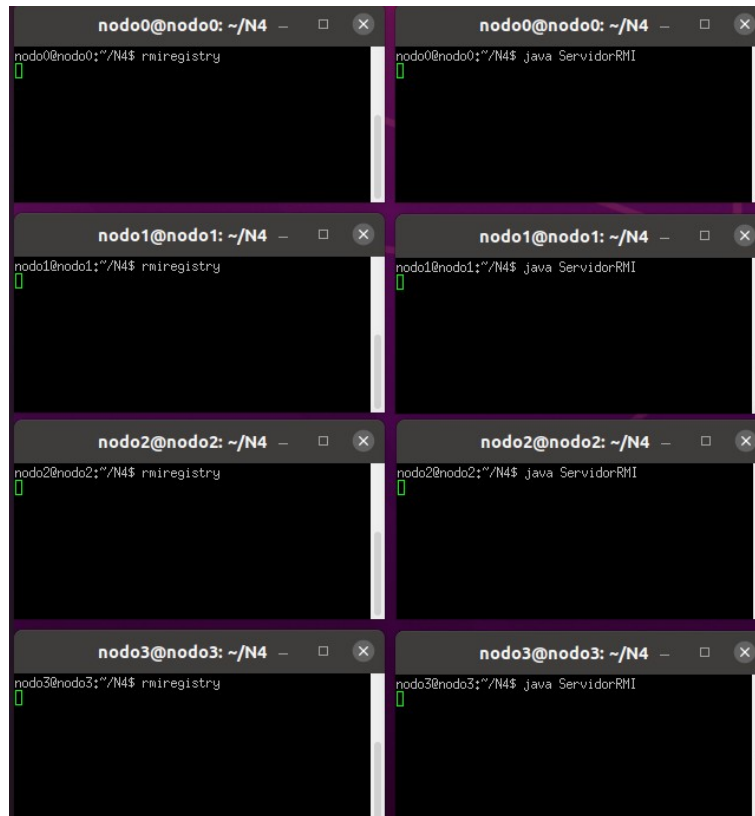


Figura 8. Ejecución rmiregistry y servidores.

Una vez que tenemos todos los servidores corriendo, procedemos a ejecutar el cliente para poder realizar la multiplicación de las matrices y obtener la resultante, así como su checksum.

```
nodo0@nodo0: ~/N4
nodo0@nodo0:~/N4$ java ClienteRMI

Matriz A
0      -1      -2      -3
2       1       0      -1
4       3       2       1
6       5       4       3

Matriz B
0       2       4       6
1       3       5       7
2       4       6       8
3       5       7       9

Matriz C
-28     -34     -40     -46
-4       -2       0       2
20      30      40      50
44      62      80      98

Resultado del checksum = 272
nodo0@nodo0:~/N4$
```

Figura 9. Resultado del cliente.

- $N = 500$

Después, tenemos que realizar el mismo procedimiento, pero con $N = 500$. Al igual que en el caso anterior, hay que ejecutar `rmiregistry` y los servidores. Se puede observar en las terminales como ha cambiado la carpeta.

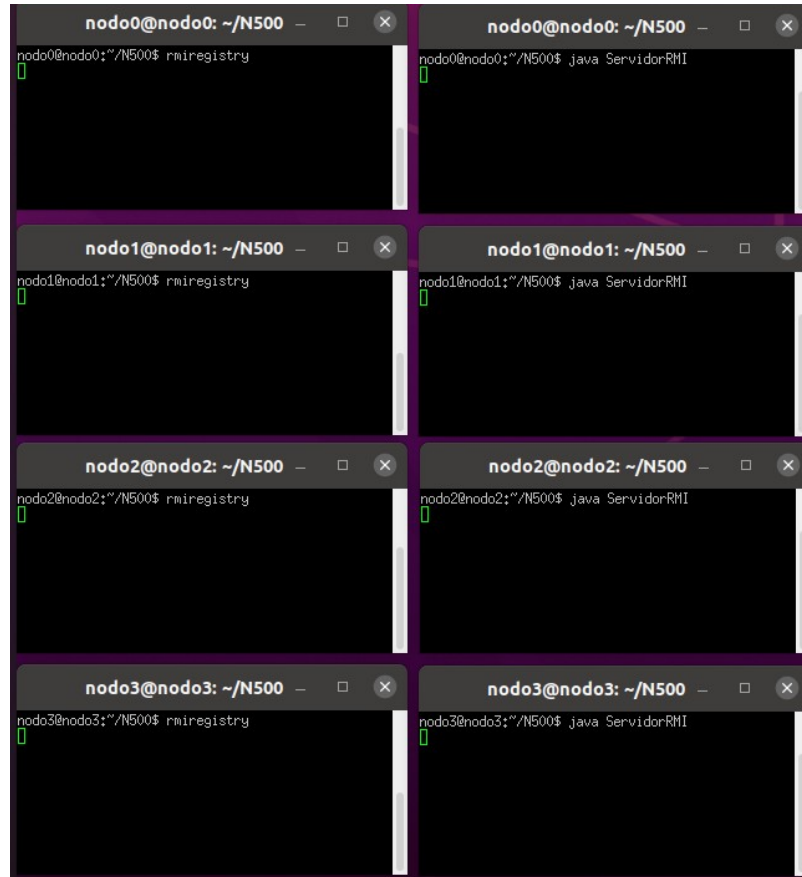


Figura 10. Ejecución de `rmiregistry` y servidores.

Una vez que todos los servidores están en ejecución, procedemos a ejecutar el cliente. Para este caso, no se imprimirán las matrices, únicamente aparece el checksum de la matriz resultante.

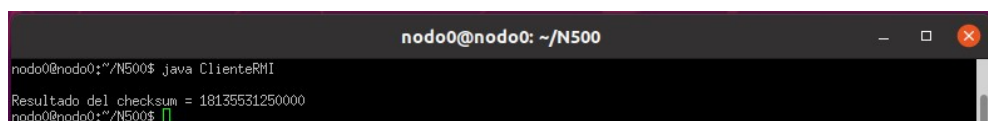


Figura 11. Checksum de la matriz.

Conclusión

Los sistemas distribuidos requieren que las operaciones corran en diferentes espacios de direcciones, es decir sobre diferentes hosts. RMI es un sistema que nos permite el intercambio de objetos. Este intercambio se realiza de manera transparente de un espacio de dirección a otro. Además que nos permite el llamado de los métodos remotamente, sin tener la necesidad de tener los métodos localmente.

Debido a que los sistemas necesitan manejo de datos en sistemas distribuidos cuando estos residen en direcciones de distintos Host, los métodos de invocación remota son una buena alternativa para solucionar estas necesidades.