DEPARTAMENTO DE INFORMÁTICA
INGENIERÍA EN INFORMÁTICA.
DESARROLLO DE APLIACIONES DISTRIBUIDAS
EXAMEN OBLIGATORIO

29 de enero de 2008

Para la realización del examen se dispondrá de **2 horas. NO** se podrán utilizar libros ni apuntes.
Nombre y Apellidos:
NIA:

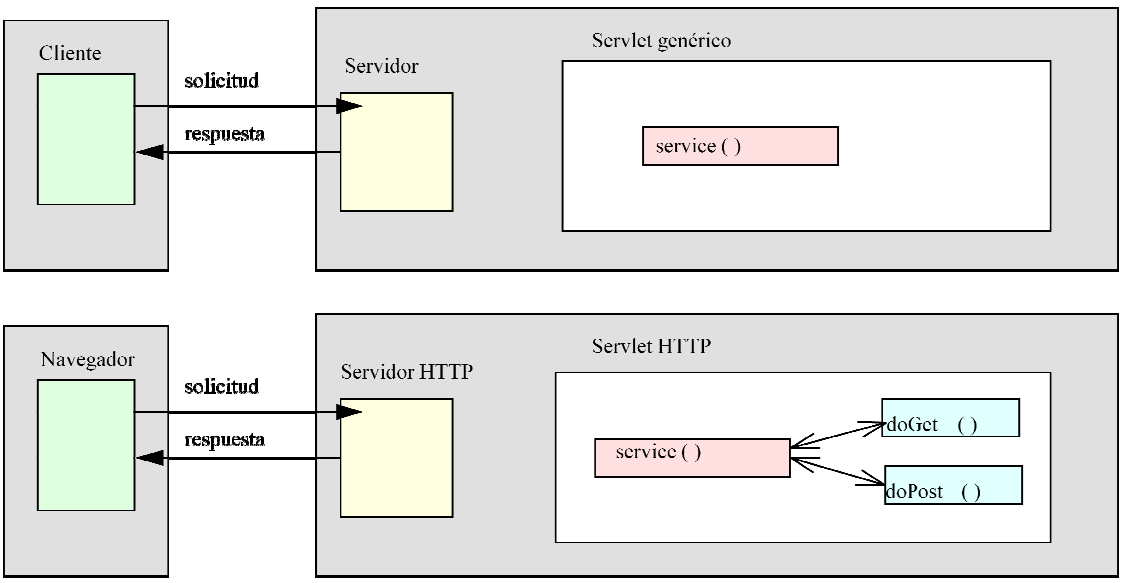
Ejercicio 1. (1.5 puntos)

Indica cuál es la estructura de una arquitectura cliente-servidor basada en *servlets*. Ilústrala con un diagrama explicativo.

¿Se puede desarrollar este paradigma en el servidor usando CORBA?

SOLUCIÓN:

- **Son ejecutados en la máquina servidora.**
- **Motor/contenedor de *servlets***

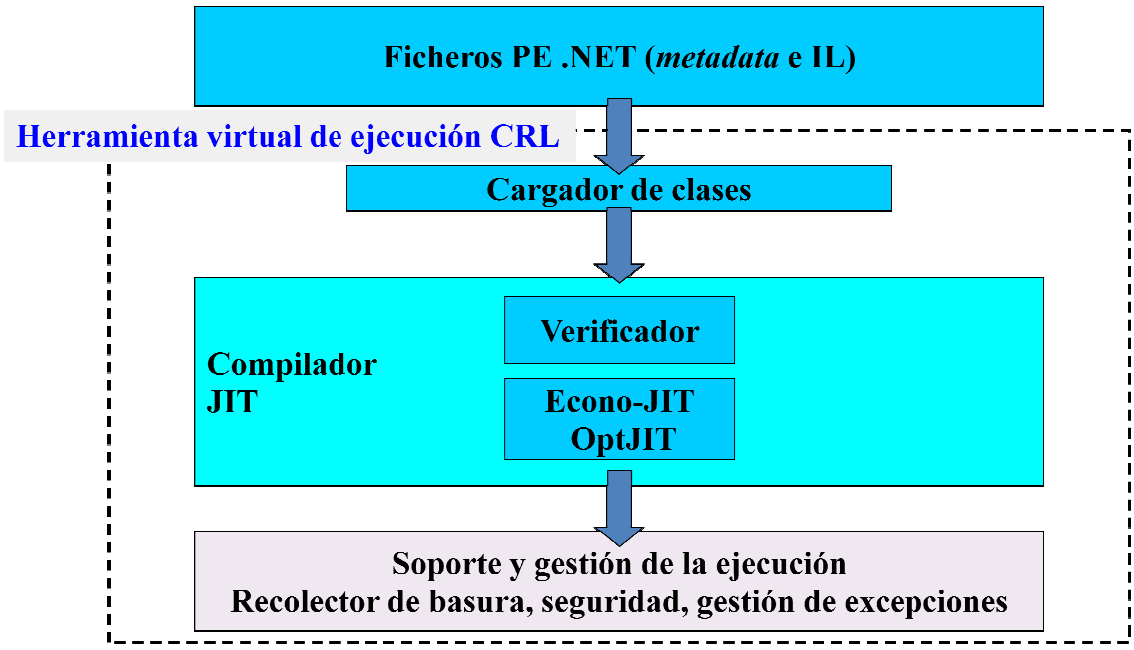


- **Si, se puede desarrollar dado que Corba da soporte a la creación y gestión de objetos servlets. Además, estos servlets pueden implementar comunicaciones bajo el protocolo http.**

Ejercicio 2. (2 puntos)

Describe la estructura de la Herramienta Virtual de ejecución CLR. Indica el papel de cada uno de los elementos que la componen.

SOLUCIÓN:



- **Ejecución CLR:**
  - **Cargador de clases.**
  - **Verificador.**
    - **Comprueba que *metadata* está bien construida.**
    - **El código IL es seguro.**
  - **Compiladores JIT.**
    - **Convierte código IL a código nativo.**
    - **Realiza optimizaciones dependientes de la arquitectura.**
  - **Soporte y gestión de la ejecución.**
    - **Recolector de basura.**
    - **Tratamiento de excepciones.**
    - **Seguridad.**
    - **Depurado.**

Soporte de interoperabilidad.

Ejercicio 3. (2 puntos)

¿Qué modos de activación de objetos remotos existen en .NET remoting? Indica las características de cada uno así como sus principales ventajas e inconvenientes.

SOLUCIÓN:

- **Activados por el cliente.**
  - **Se activa cada vez que el cliente crea el objeto.**
  - **Sólo sirve a un único cliente.**
  - **Puede producir un alto tráfico de red sin operaciones remotas asociadas.**
- **Activados por el servidor (*well-known objects*)**
  - **Se activan cada que el cliente invoca el objeto.**
  - **Reduce tráfico de red.**
  - **Existen dos clases:**

- **_Singleton_: existe un único objeto atendiendo a múltiples clientes. Permite mantener estado.**
- **_Singlecall_: Creados y destruidos en cada invocación de un cliente. Tiene una mayor sobrecarga debido a que el objeto se debe crear en cada invocación. No tiene estado asociado.**

Ejercicio 4. (1.5 puntos)

Se desea desarrollar una aplicación distribuida en la que el servidor debe tener estados. Indica, de forma justificada, qué alternativa resulta más adecuada:

1. ASP.NET Web services.
2. .NET Remoting.

SOLUCIÓN:

- **ASP.NET Web services en principio resulta la menos adecuada debido a que asume servidores sin estados.**
- **.NET Remoting permite la creación de servidores con estados mediante objetos singleton.**

Ejercicio 5. (1.5 puntos)

El proyecto Mono (impulsado por Novell) comprende un entorno de desarrollo de código libre para la ejecución de aplicaciones .NET en plataformas Linux, Solaris, Mac OS X, Windows y Unix. ¿Es posible compilar y crear un programa ejecutable .NET en Linux y después ejecutarlo en Windows?

SOLUCIÓN:

Si, debido a que un fichero PE no contiene código máquina nativo, sino código IL que es posible ejecutar en múltiples plataformas como las indicadas en el enunciado.

Ejercicio 6. (1.5 puntos)

Dado el siguiente código en Java RMI, del que forman parte las clases tipo A y tipo B. Indicar, de forma justificada:

1. A qué paradigma de programación se corresponde.
2. ¿Cómo se ejecutaría esta aplicación distribuida?

CLASE A

```
import java.rmi.*;

public interface ObjectInterface extends Remote {
    void execute()
    throws java.rmi.RemoteException;
}

import java.io.*;
import java.util.*;
import java.rmi.*;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;

public class MyObject extends UnicastRemoteObject implements ObjectInterface {

    int hostIndex;
    String name;
    Vector hostList;
    int RMIPort = 12345;
```

```
    public MyObject(String myName, Vector theHostList, int theRMIPort ) {
          name = myName;
          hostList = theHostList;
          hostIndex = 0;
          RMIPort = theRMIPort;}

    public void execute() {
      String thisHost, nextHost;
      thisHost = (String) hostList.elementAt(hostIndex);
      hostIndex++;
      if (hostIndex < hostList.size()) {
         nextHost = (String) hostList.elementAt(hostIndex);
         try {
            Registry registry = LocateRegistry.getRegistry("localhost", RMIPort);
            ServerInterface h = (ServerInterface) registry.lookup(nextHost);
  L1          h.receive(this);
         }catch(Exception e){}
    }
}
```

CLASE B

```
import java.rmi.*;

public interface ServerInterface extends Remote {
  public void receive(MyObject h)
     throws java.rmi.RemoteException;
}

import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.net.*;
import java.io.*;

public class Server extends UnicastRemoteObject implements ServerInterface{
  static int RMIPort = 12345;

       public void receive(MyObject h) throws RemoteException {
L2    h.execute();
  }

  public static void main(String args[]) throws Exception {
    InputStreamReader is = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(is);
    String s;
    String myName = "server" + args[0];
    System.setSecurityManager(new RMISecurityManager());
    Server h = new Server();
    Registry registry =  LocateRegistry.getRegistry(RMIPort);
    registry.rebind( myName, h);
}
```

SOLUCIÓN:

1. **El objeto MyObject es un objeto remoto debido a que extiende la clase UnicastRemoteObject. A diferencia de un agente (que extiende la clase Serializable), la invocación del método receive (L1) implica el envío de una referencia del objeto (en vez del objeto completo serializado). De este modo, la invocación del método del objeto (L2) implicará la ejecución de un método remodo (en vez de local).**
2. **El paradigma que tiene asociado es el de clientes-servidor.**