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

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 punto) ¿Qué diferencia existe en hacer un bind y un rebind en el contexto de Java RMI?

SOLUCIÓN:

En ambos casos se crea una entrada en el registo de nombre (asociado al rmiregistry) que contiene la referencia de un objeto remoto y su etiqueta asociada. En un rebind, si la entrada (etiqueta) existe previamente, ésta borrada y reemplazada por la nueva (asociada al rebind). En el caso del bind no se puede sobreescribir esta entrada.

Ejercicio 2. (1 punto) ¿Qué información descarga el cliente del Servicio de Nombres (asociado al proceso RMIRegistry) cuando obtiene la referencia de un objeto remoto? Indica los campos que componen dicha información.

SOLUCIÓN:

El nombre del objeto remoto, la dirección del servidor y número de puerto asociado en el servidor.

Ejercicio 3. (2 puntos) Dentro de la comunicación en grupo:

1. ¿Qué diferencia existe entre un modelo de multidifusión FIFO y uno atómico? Indica un ejemplo para cada caso.
2. ¿Se puede realizar multidifusión en la invocación de métodos remotos de Java RMI? Razona la respuesta.

SOLUCIÓN:

- **FIFO:**
  A emite *A_1-A_2* y *B emite B_1-B_2*

  Puede llegar cualquiera de estos órdenes. Además, pueden llegar órdenes distintos a cada uno de los consumidores.
  ***A_1-A_2-B_1-B_2*,**
  ***A_1-B_1-A_2-B_2*,**
  ***A_1-B_1-B_2-A_2*,**
  B_1-A_1-A_2-B_2
  B_1-A_1-B_2-A_2
  ***B_1-B_2-A_1-A_2*.**

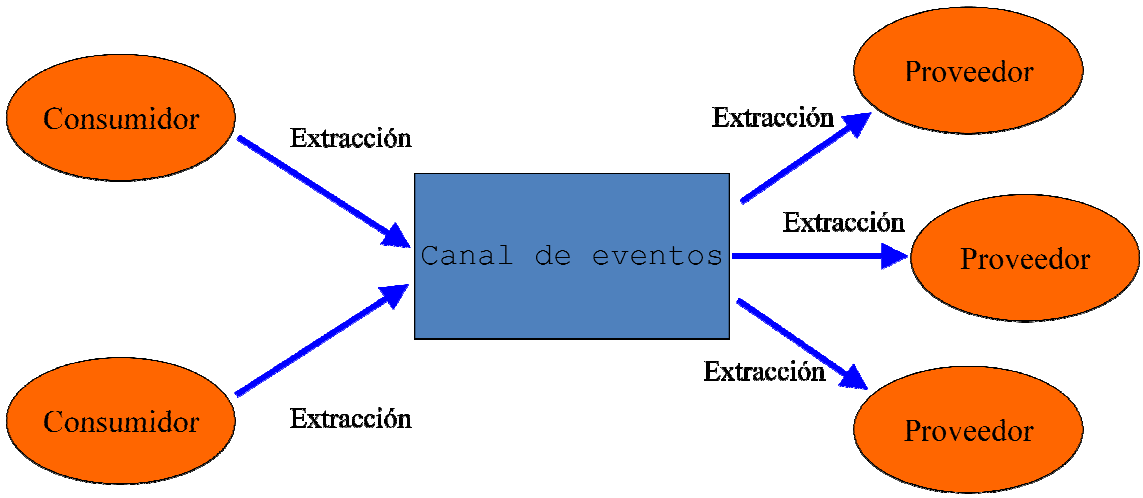
- **Orden atómico:**
  Existen distintos órdenes pero sólo uno llegará a todos los consumidores.
  Ejemplo 1:
  ***P_1* envía *m_1*, *P_2* *envía* *m_2*, y *P_3* envía *m_3*.**
  m_1-m_2- m_3, m_1- m_3- m_2, m_2- m_1-m_3,
  m_2-m_3-m_1, m_3-m_1- m_2, m_3-m_2-m_1.
  Ejemplo 2:
  ***P_1* envía *m_1* y luego *m_2*.**
  ***P_2* responde a *m_1* enviando *m_3*.**
  P_3* responde a *m_3* enviando *m_4
  Orden a respetar: m_1 - m_3 - m_4
  m_1- m_2- m_3- m_4, m_1- m_3- m_2- m_4, m_1- m_3- m_4- m_2.

  - **RMI da soporte a multidifusión mediante la clase MulticastRemoteObject. Esta clase no está implementada en la distribución básica de RMI, pero puede desarrollarse libremente.**
  - **Otra alternativa consiste en emplear sockets factories con soporte a multidifusión.**

Ejercicio 4. (2 puntos) Canal de eventos de CORBA, **modelo canónico de extracción**, se pide:

1. Definir dicho modelo indicando el rol que desempeña el canal de eventos.
2. ¿Qué elemento tiene un diseño más simple, el proveedor o el consumidor de eventos?
3. ¿Qué desventaja tiene dicho modelo?

SOLUCIÓN:



Dirección del flujo de eventos

SOLUCIÓN:

El rol del canal de eventos es el de un procurador de eventos.

Los consumidores actúan como clientes y los proveedores como servidores. En este sentido, la implementación del consumidor es más simple que la del proveedor.

- **La principal desventaja de este modelo es que :**

- **Delega en el muestreo por parte del consumidor.**
- **Es necesaria una política de eliminación de eventos.**
- **Puede producir alto tráfico de red.**

Ejercicio 5. (2 puntos) ¿Qué alternativas ofrece CORBA para acceder, por parte del cliente al *stub* del servidor?. Indica razonadamente cuál es la más recomendada en cada uno de los siguientes casos:

1. Clientes invocan con frecuencia los métodos remotos y los objetos de los servidores no cambian.
2. Clientes descubre servidores en tiempo de ejecución.
3. Clientes se ejecutan desde un navegador y descubren nuevos objetos remotos.

SOLUCIÓN:

Existen diversas alternativas: stubs estáticos (como los utilizados en las prácticas), descarga dinámica de stubs y Dynamic Invocation Interface.

En los casos comentados en el enunciado:

- **Clientes invocan con frecuencia y los objetos de los servidores no cambian ⇒ stubs státicos.**
- **Clientes descubre servidores en tiempo de ejecución ⇒ DII.**
- **Clientes se ejecutan desde un navegador y descubren nuevos objetos ⇒ descarga dinámica de *applets* o *stubs*.**

Ejercicio 6. (2 puntos) Dado el siguiente código de CORBA, indicar:

1. ¿A qué elemento se corresponde?
2. Comentar qué se realiza en las líneas: L1, L2, L3, L4, L5 y L6.
3. ¿Cómo se genera este código?

```
package HelloApp;
public class xxx extends org.omg.CORBA.portable.ObjectImpl implements
HelloApp.Hello
{
  public String sayHello (String p_a, double p_b)
  {
         org.omg.CORBA.portable.InputStream $in = null;
         try {
                org.omg.CORBA.portable.OutputStream $out = _request
                                                ("sayHello", true);
L1              $out.write_string (p_a);
L2              $out.write_double (p_b);
L3              $in = _invoke ($out);
L4              String $result = $in.read_string ();
L5              return $result;
            }
            catch (org.omg.CORBA.portable.ApplicationException $ex) {
                $in = $ex.getInputStream ();
                String _id = $ex.getId ();
                throw new org.omg.CORBA.MARSHAL (_id);
            } catch (org.omg.CORBA.portable.RemarshalException $rm) {
L6              return sayHello (p_a, p_b        );
            } finally {
                _releaseReply ($in);
            }
```

```
}
```

SOLUCIÓN:

El código se corresponde a un stub de CORBA

L1: almacena en el stream de datos el primer argumento del método remoto.
L2: almacena en el stream de datos el segundo argumento.
L3: se invoca el método remoto obteniendo un stream de datos
L4: Se extrae el valor devuelto del método remoto del stream de datos.
L5: Se devuelve el valor devuelto.
L6: En caso de haber una excepción, se repite el proceso iterativamente.

Este código es creado automáticamente a partir del interface IDL empleando el compilador idlj.