

Práctica Broker

Asignatura: Arquitectura Software
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza
jmerse@unizar.es

1. Objetivo

Desarrollar un broker de objetos sencillo haciendo uso de java RMI. Se proponen dos versiones (basic y pro). El alumno decidirá cuál de ellas implementar.

2. Entrega de la práctica

Se presentará y discutirá con el profesor la semana del 18 al 22 de mayo de 2020 durante la sesión de revisión del trabajo.

3. Componentes a implementar

■ ServerA

- Se implementará su interfaz remota y la clase correspondiente.
- Versión basic: Ofrece diferentes servicios/métodos-remotos: los que quieras, por ejemplo: *dar_fecha()* y *dar_hora()*. Además, es capaz de registrar su ubicación en el Broker.
- Versión pro: Es capaz de registrar sus servicios en el Broker.

■ ServerB

- Se implementará su interfaz remota y la clase correspondiente.
- Versión basic: Ofrece diferentes servicios/métodos-remotos: los que quieras. Además, es capaz de registrar su ubicación en el Broker.
- Versión pro: Es capaz de registrar sus servicios en el Broker.

■ Broker

- **Versión basic.** Los servicios de los servidores (ServerA y ServerB) los tiene ya programados en código (hard-coded). Si un servidor añadiese un nuevo servicio habría que recompilar el Broker. Pero los servidores pueden cambiar de máquina. Métodos a implementar:

API para los servidores:
`void registrar_servidor(String nombre_servidor,
 String host_remoto_IP_puerto)`

API para los clientes:

```
Respuesta ejecutar_servicio(String nom_servicio,
                           Vector parametros_servicio)
```

- **Versión pro.** Los servicios se ejecutan “dinámicamente”. Si se añade o se quita un servicio, no hay que recompilar el Broker. Métodos a implementar:

API para los servidores:

```
//Registrar servidor como en la versión basic
//Registrar servicio
void registrar_servicio(String nombre_servidor,
                       String nom_servicio, Vector lista_param,
                       String tipo_retorno)
//Dar de baja servicio
void baja_servicio(String nombre_servidor,
                  String nom_servicio)
```

API para los clientes:

```
//Ejecutar servicio síncrono
Respuesta ejecutar_servicio(String nom_servicio,
                           Vector parametros_servicio)
//Ejecutar servicio asíncrono
void ejecutar_servicio_asinc(String nom_servicio,
                           Vector parametros_servicio)
//Obtener la respuesta (caso asíncrono)
Respuesta obtener_respuesta_asinc(String nom_servicio)
```

- Caso asíncrono: El broker responde con mensaje de error si el objeto no hizo la petición con anterioridad, o no es el objeto que hizo la petición, o ya le sirvió la respuesta. Para simplificar un objeto no podrá pedir más de una vez un mismo servicio si no pide antes la respuesta.

■ ClientC

- Versión basic: Implementa la lógica de la aplicación (la que queramos) utilizando los servicios remotos de ServerA y de ServerB.
- Versión pro: Mirar los escenarios a defender.

4. Escenarios a defender

■ Versión basic.

1. Broker, ServerA y ServerB se registran en RMI y quedan escuchando las peticiones.
2. ServerA y ServerB se registran en el Broker.
3. Se ejecuta el ClientC. Muestra por pantalla los resultados.
4. Se cambia el ServerA (o el ServerB) de máquina. No se recompila el Broker.

5. Se ejecuta de nuevo el ClientC. Muestra por pantalla los resultados.

■ Versión pro síncrono.

1. Broker, ServerA y ServerB se registran en RMI y quedan escuchando las peticiones.
2. ServerA y ServerB registran sus servicios en el Broker.
3. El ClientC pide al Broker la lista de servicios registrados.
4. El ClientC muestra por pantalla la lista de servicios, el usuario escogerá uno de ellos.
5. El ClientC pide al Broker la ejecución de ese método y muestra el resultado por pantalla.
6. El ServerA registra un nuevo servicio y da de baja otro. El broker no se recompila.
7. Cuando el ClientC vuelva a pedir la lista de servicios para mostrarla por menú, ésta estará actualizada.
8. El usuario vuelve a ejecutar un servicio.

■ Versión pro asíncrono.

1. Simplemente implementad en el ClientC una lógica de programa que demuestre que la comunicación asíncrona funciona con las restricciones que se propusieron en la sección anterior.

5. Recomendaciones

1. Usad las máquinas del L1.02. Al estar todas en la misma subred evitamos configuraciones avanzadas de RMI. Las IPs de estas máquinas van 155.210.154.191 hasta 155.210.154.210
2. Usad tres máquinas: una para el broker, una para los dos servidores, una para el cliente.
3. Tened en cuenta que el “rmiregistry” es único por máquina y puerto.
 - Antes de lanzar el “rmiregistry” mirad si hay alguno ya lanzado. Para saber si hay alguno usa la orden “top”.
 - Si al lanzarlo obtienes un error es porque alguno de los que hay se está ejecutando en el puerto que has escogido. Intenta lanzarlo en otro puerto.
 - No hagas el “rebind” de tus objetos con nombres como “broker”, “serverA”, ya que colisionarás con los de otros compañeros si por algún motivo varios grupos estáis usando el mismo puerto. Es mejor llamarlos “broker545”, “serverB545”, donde los números pueden ser los tres últimos dígitos de tu NIA.
 - Cuando salgas de una máquina “mata” el proceso correspondiente al rmiregistry: `kill -9 pid`
4. ServerA, ServerB y ClientC, antes de ser compilados, saben en qué máquina está el Broker.

6. Notas finales

1. El código documentado.
2. Documentación básica de la arquitectura, máximo 2 páginas (1 hoja).
3. Preguntadme las dudas que tengáis (no de codificación sino de qué implementar, qué no implementar, ...).