Sistemas Distribuidos
Tarea # 2: Protocolos de Middleware

Raúl Monge Anwandter
rmonge@inf.utfsm.cl

Roberto Catricura Silva
roberto.catricura@alumnos.usm.cl

14 de octubre de 2016

1. Introducción

En un sistema distribuido, los componentes se comunican y coordinan sus acciones a través del paso de mensajes. Para agregar valor a la comunicación, se observa el desarrollo de protocolos de middleware. Los servicios asociados a este tipo de soporte se ubican sobre la capa transporte, permitiendo que las aplicaciones se comuniquen con mayor nivel de abstracción y calidad de servicio, facilitando por lo tanto el desarrollo de aplicaciones distribuidas. Entre los protocolos de comunicación de middleware se distinguen: **invocación remota** (RPC, RMI) y **middleware orientado a mensajes** (MoM).

En esta actividad se pretende familiarizar a los estudiantes con estos dos tipos de comunicación de middleware, para lo cual se utilizará invocación remota basada en Java RMI para implementar una versión simplificada de un middleware orientado a mensajes.

2. Objetivos

- Experimentar en programación distribuida con llamadas a métodos remotos utilizando Java RMI.
- Tratar dificultades que surgen en la implementación de un servicio de mensajería a ser compartido por múltiples clientes.

3. Actividades

3.1. Motivación

Entre los métodos existentes para comunicar aplicaciones o procesos está el paso de mensajes, que permite el envío y recepción de mensajes. Un middleware orientado a mensajes (MoM) soporta directamente este tipo de modelo de comunicación, pero con un mayor nivel de abstración y calidad de servicio que lo que provee por ejemplo UDP de TPC/IP. Un caso de este tipo es el estándar Java Message Service (JMS), que define una API para que componentes de una aplicación distribuida (todos ellos clientes) puedan enviar y entregar mensajes indirectamente a través del servicio, desacoplando por lo tanto las interacciones entre los clientes. Estos deben ser capaces de crear recibir, y leer mensajes de manera asíncrona.

La actividad consistirá en desarrollar una versión simplificada de una API tipo JMS, donde el proveedor del servicio se implementará a través de llamadas a métodos remotos utilizando Java RMI. De esta manera los clientes podrán enviar o recibir mensajes, comunicándose implícitamente con un servidor el cual administrará colas de mensajes con nombres. Digamos que este servicio se llama SimpleMoM (nada que ver con PokeMon).

3.2. Arquitectura

La arquitectura de SimpleMoM corresponde a un esquema cliente-servidor, donde los clientes al usar la API generan comunicación implícita por medio de llamadas a métodos remotos a un servidor. Un cliente debe ser capaz de suscribirse a una cola de mensajes, enviando y recibiendo mensajes de manera **asíncrona**. Cada cliente consumidor debe ser capaz de recibir todos los mensajes de una cola a la cual está suscrito, aun cuando éstos puedan ser consumidos por otros clientes con anterioridad.

3.2.1. Requerimientos

- **Servidor:** Es un objeto remoto Java, cuyos métodos pueden invocados (remotamente) por los clientes para administrar mensajes las colas. El servidor es el que realmente implementa el servicio (es el proveedor). Éste debe incluir:
  - Implementación de método remoto que permite el encolamiento/envío de mensaje a una cola específica.
  - Implementación de método remoto que permite sacar/recibir un mensaje desde una cola específica.
  - Registrar el objeto remoto (servidor) en el registro RMI.
- **API-Cliente:** Implementa la API para que un cliente pueda realizar invocaciones a métodos remotos al servidor través de un stub. El cliente propiamente tal accede al servicio a través de una interfaz Java de la API-Cliente, pero no requiere (y no debiera) conocer la implementación de esta interfaz. Ésta debe incluir:
  - Suscripción a una cola.
  - Envío de mensaje a cola suscrita.
  - Recepción de mensaje desde la cola suscrita.

3.2.2. Restricciones

- La actividad debe realizarse en lenguaje **Java** utilizando Java RMI.
- El servidor puede utilizar un archivo de texto para dar a conocer las colas disponibles para ser utilizadas por los clientes.
- Tanto el envío como recepción de mensaje es de manera asíncrona; es decir, tanto el emisor como el receptor nunca se bloquean para enviar o recibir un mensaje respectivamente.
- El output tanto de servidores como clientes debe seguir el formato de la sección Ejemplo de Ejecución.

3.3. Ejemplo de Ejecución

Suscripción a una cola:

```
[Cliente] Suscripción a Cola:
[Cliente] [1] Cola 1
[Cliente] [2] Cola 2
[Cliente] [3] Cola 3
```

Consola Principal:

```
[Cliente] Seleccionar opción
[Cliente] [1] Enviar mensaje
[Cliente] [2] Recibir mensaje
[Cliente] [3] Suscribir a otra cola
```

Cliente 1 - Opción [1] Enviar mensaje:

```
[Cliente] Escribir mensaje a enviar:
>Mensaje de Prueba 1
```

Cliente 2 - Opción [1] Enviar mensaje:

```
[Cliente] Escribir mensaje a enviar:
>Mensaje de Prueba 2
```

Cliente 1 - Opción [2] Recibir mensaje:

```
[Cliente] Mensaje recibido:
[Cliente] Mensaje de Prueba 1
```

Cliente 2 - Opción [2] Recibir mensaje:

```
[Cliente] Mensaje recibido:
[Cliente] Mensaje de Prueba 1
```

4. Consideraciones

- La explicación de la tarea se realizará en horario de clases, el Lunes 17 para Casa Central y el Martes 18 para Campus San Joaquín.
- Consultas sobre la tarea se deben realizar en Moodle o enviar un correo a **roberto.catricura@alumnos.usm.cl**
- La revisión se realizará en el Laboratorio de Integración Tecnológica, tanto para Casa Central como para Campus San Joaquín.

5. Condiciones de entrega

- La tarea se realiza en grupos de 2 personas.
- La fecha de entrega es el día **Viernes 4 de Noviembre a las 23:59**
- Se debe subir a Moodle un archivo gzip con los archivos necesarios para compilar. El nombre del archivo debe ser rol1-rol2.tar.gz (por ejemplo: 29730997-29730377.tar.gz)
- El contenido debe respetar las siguientes especificaciones, dejando en el mismo nivel los siguientes archivos:
  - Todos los códigos fuentes de las clases e interfaces
  - Makefile con opciones para compilar los binarios.
  - Un archivo README que explique cómo ejecutar los binarios, estrategia de implementación y supuestos tomados.