Práctica 4 - Broker de mensajes

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

Esta práctica consta de dos partes. Primero, aprenderemos a usar RabbitMQ siguiendo los pasos indicados en el documento "P/S con RabbitMQ" que encontrarás en Moodle. Segundo, una vez entendidos los conceptos ilustrados por RabbitMQ, implementaremos un broker de mensajes básico. A continuación se describen las funcionalidades a desarrollar. El enunciado se plantea con diferentes niveles de dificultad. Cada equipo irá decidiendo hasta qué nivel desarrolla.

Versión básica

Se puede implementar en el lenguaje que desees, pero sin utilizar RabbitMQ. Las funcionalidades del broker de mensajes serán muy parecidas a las del primer tutorial de RabbitMQ. Por tanto, el broker permitirá que un programa *productor* escriba mensajes en una cola y que estos puedan ser leídos por un programa *consumidor*. El broker ofrecerá los siguientes métodos:

- *declarar_cola(string nombre_cola)*. Al igual que sucede en RabbitMQ esta operación será idempotente. Es decir, que aunque se invoque varias veces el broker creará una única cola con ese nombre. Por tanto, esta operación la usarán tanto el *productor* como el *consumidor*. Podéis usar el patrón Singleton [1] para ello.
- *publicar(string nombre_cola, string mensaje)*. Esta operación la usa el *productor* para publicar mensajes en una cola a la que previamente se haya unido.
- *consumir(string nombre_cola, string mensaje)*. Esta operación la usa el *consumidor* para consumir mensajes de una cola a la que esté suscrito.

El programa *consumidor* tendrá una operación de *call back* a la que llamará el broker para que se consuma el mensaje inmediatamente después de que se haya publicado. Si no hubiese ningún *consumidor* asociado a esa cola el mensaje se perderá. Si el mensaje no es consumido en 5 minutos el broker lo eliminará de la cola. Si más de un *consumidor* se suscribe a la misma cola entonces el broker asigna los mensajes con política *round robin*, como en el tutorial 2 de RabbitMQ.

Versiones avanzadas

- Los programas *productor*, *consumidor* y el broker pueden estar en máquinas diferentes.
- Implementar política *fair dispatch* como en el tutorial 2. Se deberá también implementar un mecanismo de reconocimiento de mensajes procesados.
- Implementar "message durability" como en el tutorial 2. Es decir, que si el broker se "cae" las colas que han sido creadas como duraderas se recuperarán cuando el broker sea "lanzado" de nuevo, y los mensajes duraderos que permaneciesen en esas colas también serán recuperados.

No es necesario implementar todos los ítems anteriores.

Notas finales

1. El código documentado.
2. Documentación de la arquitectura: Vista de módulos, Vista de CyC y Vista de distribución. Máximo 2 páginas (1 hoja).
3. Preguntadme las dudas que tengáis (no sobre codificación sino sobre qué implementar, qué no implementar, . . .).

Referencias

[1] **Design Patterns: Elements of Reusable Object-Oriented Software.** *Erich Gamma et al.* Addison Wesley.
`ISBN: 0201633612`