Resumen charla Prof. Russo

Santiago Muñoz Castro

[1] Universidad de Granada, ETSIIT, ES
[2] Calle Periodista Daniel Saucedo Aranda, s/n, 18014 Granada
santiyagito22@correo.ugr.es

Abstract: En el transcurso de este documento vamos a relacionar los términos otorgados durante el desarrollo de la asignatura, es decir, conceptos y términos sobre sistemas distribuidos con la charla presentada por el profesor informático e italiano Stefano Russo, donde abarcaremos temas sobre RPC, middleware, MOM, objetos distribuidos y otros términos relacionados con los sistemas distribuidos.

Keywords: RPC, Objeto distribuido, MOM, Middleware, stub, cliente, servidor

1 Sistemas distribuidos

1.1 ¿Qué son?

Antes de empezar, necesitamos conocer el concepto de un sistema distribuido, según G.Coulouris un sistema distribuido es un sistema cuyos componentes están ubicados en computadoras conectadas a la red y que se comunican y coordinan sus acciones a través **solo** del paso de mensajes, esta definición conlleva a tres características:

- Concurrencia de componentes, dos formas de interacción entre componentes la cooperación (colaboran para alcanzar un objetivo común) y la competición.
- No tienen reloj global, la sincronización se realiza a través del paso de mensajes.
- Fallos independientes de componente en nodos o en el sistema de comunicación.

1.2 Factores de diseño

El diseño de los sistemas distribuidos es difícil, hay que tener varios factores en cuenta como la heterogeneidad (asegurar homogeneidad por ejemplo en los lenguajes de los distintos sistemas), franqueza, seguridad (confidencialidad, integridad y disponibilidad), concurrencia, transparencia, la escalabilidad y detección y control de fallos.

Uno de los significados de transparencia es la ubicación, si queremos enviar una petición o acceder un objeto, no queremos saber donde está la dirección física de este, sino que queremos acceder a este eventualmente y de forma sencilla, un ejemplo de transparencia de ubicación es el nombre de dominio.

2 Middleware

2.1 ¿Qué son?

Middleware es cualquier capa intermedia de un sistema distribuido, es decir, se encuentra entre las aplicaciones y los sistemas operativos, o dicho de otra forma entre las aplicaciones y las plataformas.

Aumentan la productividad del programador utilizando abstracciones para acceder a los recursos y para las comunicaciones de las capas por encima del sistema operativo. El paso de mensajes (send/receive) se realiza básicamente con TCP/IP o cualquier otro protocolo de internet.

2.2 RPC

Un buen ejemplo de middleware es RPC que permite abstraer e implementar las comunicaciones, como llamadas a procedimientos locales pero ejecutados en máquinas remotas[2].

Si tenemos un programa que se ejecuta en un mismo nodo, podemos hacer que este **programa tradicional se convierta en un programa distribuido** de forma inmediata. De forma normal tendríamos que tener bastante conocimiento sobre paso de mensajes con TCP/IP para comunicarnos con los distintos nodos (cosa que un programador de un solo lenguaje no sabría como hacerlo), pero con RPC esto se facilita teniendo que programar unicamente la llama al procedimiento de la funcion del servidor que aunque este programado en otro lenguaje siempre devolverá un resultado que será enviado al cliente.

¿Cómo funciona RPC?

Partimos de la base que tenemos una especie de prototipo, una especificación del procedimiento que deseamos que se invoque de forma remota, esta compilación genera automáticamente dos procedimientos denominados respectivamente stub del cliente y stub del servidor.

El stub del cliente es compilado y luego enlazado con el programa que lo invoca (*caller*) y el stub del servidor está vinculado al ser compilado con el procedimiento remoto. Tanto el *caller* como el procedimiento remoto no cambian y se comunican con los stubs mediante llamadas locales, ya que como habíamos dicho un sistema distribuido solo intercambia mensajes.

En resumen, al invocar al stub del cliente, este enviá un mensaje (por cualquier protocolo de internet), al stub del servidor, que recibe ese mensaje (por un protocolo de internet) e invoca al procedimiento remoto que corresponda , este procedimiento devolverá un resultado que el stub del servidor enviara al stub del cliente mediante un mensaje y este último le pasara el resultado finalmente al *caller* o programa que ha invocado al procedimiento. Lo único que se pasa de forma remota es el mensaje de solicitud del cliente y de respuesta del servidor, todo lo demás se realiza de forma local.

Si se produce un retraso en el envio de un mensaje se llega a la pregunta de si tenemos que volver a enviar la solicitud o tenemos que seguir esperando. Para ello RPC utiliza TCP o UDP, y la semántica de los mensajes dependerá de estos protocolos.

Un inconveniente de utilizar UDP es que el procedimiento remoto puede ser idempotente, es decir, que se llame numerosas veces provocando el mismo efecto.

2.3 MOM

Los mensajes orientados a Middleware estan basados en las abstracción de un cola de mensajes entre procesos. Si una aplicación “A” se quiere comunicar con una aplicación “B”, lo hacen mediante colas que son gestionadas por un manager. De esta forma cuando una aplicación envíe un mensaje, esta podrá seguir realizando tu trabajo ya que ese mensaje es guardado en la cola.

Los mensajes están tipados es decir que podemos decir que una cola guarde mensajes de un tipo X. MOM hace que el programador no tenga que preocuparse de aspectos de bajo nivel y expone una interfaz de alto nivel (todo lo contrario al simple send y receive de TCP/IP).

Hay muchos mecanismos que pueden ser utilizados en middleware, por ejemplo que MOM sea capaz de publicar contenido de interés para los consumidores que pueden ser notificados cuando hay nuevo contenido disponible, por ejemplo suscribirte a una marca de periódico y recibir nuevo contenido (mediante notificación) sin necesidad del que lo publica te conozca y viceversa, y pudiendo obtenerlo cuando se desee(comunicación push y pull) y sin necesidad de estar conectado en el momento de publicación.

2.4 Modelo espacial de tuplas

Basado en Linda, uno de los primeros middleware desarrollado por David Gelernter y Nicholas Carriero en 1986

¿En que consiste?

Tenemos unas tablas que son registros , las aplicaciones pueden escribir estructuras de datos llamando a las tablas del espacio de tuplas. También pueden leer y coger dichas tuplas. Este espacio es compartido virtualmente y facilita la sincronización entre aplicaciones de forma sencilla.

La diferencia entre leer y coger es que el primero lee la tupla pero sin eliminarla del espacio de tuplas, mientras que el segundo **coge** dicha tupla eliminándola de dicho espacio. Este modelo esta implementado para Java en JavaSpace donde el espacio de tuplas llega a ser un espacio de objetos porque estos no son simplemente objectos de una estructura de datos, sino que pueden llegar a ser más complejos.

3 Objetos distribuidos

Un objeto es un ente orientado a objetos que consta de un estado y de un comportamiento, que a su vez constan respectivamente de datos almacenados y de tareas realizables durante el tiempo de ejecución[3]. Dicho de otra forma, un objeto es la encapsulación de una estructura de datos concreta en las operaciones para acceder a él.

En los lenguajes tipados, en las clases o objetos, la estructura de datos es privada y también el algoritmo en los métodos, es decir, tenemos dos cosas, la estructura de datos que no se puede ver desde otro lado y los algoritmos de los métodos, esto es el concepto de un dato de tipo abstracto característico de los lenguajes tipados como C++, Java etc…

3.1 Cliente-Servidor

El servidor es una entidad pasiva que espera a una interacción que es iniciada por el cliente que es una entidad activa. El objetivo obviamente es una entidad pasiva ya que cuando utilizamos un método este responde, del mismo palo que el servidor. El obetivo en los sistemas distribuidos es ejecutar un método de un objeto local de forma remota.

De aquí surge la fórmula creada por el propio profesor Russo que dice que los objetos distribuidos es igual a los objetos orientados a programación más el cliente-servidor, según el es básica y esencial para un sistema distribuido.

3.2 Java RMI

Es una tecnología que permite construir objetos distribuidos que operen en el entorno de aplicación Java[4].

Para su funcionamiento basta unicamente con declarar la interfaz con un ***extends Remote*** para definir el servicio remoto.

4 Microservicios

Los microservicios son difíciles de definir ya que hay varias definiciones distintas sobre estos, esto es una nueva generación usado por muchas compañías como Amazon, Twitter y Netflix, son la evolución de los servicios orientados.

Poniendo como ejemplo a Netflix, si accedemos a su página observamos en una película por ejemplo, que tiene varios apartados como que está disponible en HD, la descripción de la película, el director y los actores etc… Todos estos provienen de diferentes microservicios. La ventaja de utilizar microservicios es que si uno falla el cliente ni siquiera se da cuenta de que este no esta, ya que no produce un gran cambio en la vista de este, básicamente nunca notas que falla ya que es prácticamente imposible que fallen al mismo tiempo todos los microservicios.

Otra ventaja de los microservicios es que si queremos actualizar uno no es necesario que deje de funcionar el sistema ya que este no depende de dicho microservicio, pudiendo cambiar el código de este mientras funciona el sistema.

5 Conclusión y opinión

Durante el desarrollo de la charla he aclarado los conceptos de sistemas distribuidos de forma más clara, he aprendido mejor el concepto de RPC que aunque lo entendía ahora, me ha quedado más claro que dos stubs se pueden comunicar de forma remota mediante clases, objetos y llamadas de forma local.

A parte he conocido otros conceptos nuevos como el de microservicio, que lo había escuchado pero nunca lo había entendido, ahora sé que es una aplicación que se puede implementar y probar de forma independiente y que tienen una sola responsabilidad[Johannes Thones].

Al final de la charla, por la falta de tiempo se habla de muchos conceptos en poco tiempo y extenderse en cada uno de ellos daría lugar a un nuevo trabajo (WSDL, SOAs, WS, UDDI, XML, EDA etc…).

Sobre la charla, me ha parecido muy buena, el profesor Russo explico bien la materia y puso buenos ejemplos, a pesar de mi mal nivel de inglés pude entender casi todo gracias al traductor de google meet y desarrollar este trabajo. Por poner una pega, como ya he dicho antes se abarcan muchos temas extensos en poco tiempo.

References

1. *G. Coulouris,2012,Distributed Systems: Concepts and Design (5th Edition).*
2. Transparencias Práctica 2, Sun RPC .
3. Wikipedia (Objeto programación), https://es.wikipedia.org/wiki/Objeto_(programación), último acceso el 04/06/2021
4. Transparencias Práctica 3, RMI
5. El resto de información está sacado de las diapositivas y apuntes de la charla.