Modelos y arquitecturas de Middleware

Raúl Castro Moreno

[1] Universidad de Granada, Hospital Real, Avenida del Hospicio, S/N C.P. 18010
[2] ETS Ingeniería Informática y Telecomunicaciones, Calle Periodista Daniel Saucedo Aranda, s/n, 18014 Granada
raulcastromor@correo.ugr.es

Resumen. El documento trata sobre un resumen de la charla impartida por Stefano Russo, con un contenido el cual empieza con una introducción especificando mejor sobre que iba la charla en general y con un contenido basado en los diferentes modelos y arquitecturas del Middleware. Estos modelos y arquitecturas son clasificados según su taxonimía. También, antes de pasar a explicar estos modelos y arquitecturas, se comentan los factores de complejidad que surgen a la hora de diseñar Sistemas Distribuidos. Por último se termina con unas conclusiones obtenidas de la charla en sí , y finalizando con una valoración personal donde se cuenta las impresiones obtenidas sobre lo aprendido en la charla.

Palabras Clave: Middleware, Stubs , Sistema Distribuido

1 Resumen de la Conferencia y Conclusiones

1.1 Introducción

La charla es impartida por **Stefano Russo**, el cuál es un integrante del grupo de investigación **DESSERT** [1], que pertenece a la Universidad de Nápoles Federico II . Se dedican a investigar el diseño, la evaluación y la verificación de sistemas fiables, seguros y en tiempo real. Estos sistemas suelen tener requisitos de confiabilidad, seguridad y comportamientos temporales. Estos requisitos pueden contradecirse entre sí. Al proporcionar metodologías de ingeniería de software unificadas y técnicas para evaluar los tiempos de respuesta, confiabilidad y seguridad del sistema durante todo el proceso de diseño, el grupo de investigación define y aplica nuevos principios de ingeniería al diseño y verificación de sistemas críticos.

Luego de saber quién es y a que se dedica, se comenta sobre lo que se va a explicar en la charla, lo cual tiene que ver con una visión y una taxonomía sobre **Modelos de Middleware.** Y así es como empieza, enseñandonos una cantidad de tecnologías las cuales son sistemas distribuidos los cuales hacen uso de algún nivel de Middleware.

Es con esto que se empieza comentando lo que es un Sistema distribuido, dando hasta una serie de características esenciales de ellos como son la **concurrencia de los componentes**, que **no hay un reloj global** sino que se sincronizan mediante paso de mensajes,etc.

A partir de aquí, ya se empiezan a hablar de varios conceptos y procedimientos y demás.

1.2 Factores de Complejidad

Se comentan algunos de los factores de complejidad que existen a la hora de diseñar Sistemas Distribuidos. Estos tiene que ver con:

La **Heterogeneidad** de tecnologías, protocolos, sistemas operativos, etc.

La **falta de restricciones** para que los servicios sean accesibles a nodos heterogéneos.

La **seguridad**, conformando esta la confidencialidad, integridad y la disponibilidad.

La **concurrencia** entre varios usuarios simultáneos.

Y se comenta también sobre la **transparencia, escalabilidad** y los posibles **fallos** que se transforman en **fallos parciales**, sobre un Sistema Distribuido.

Luego de esto, Stefano comenta que es el **Middleware** en general, al cuál el se refiere como una capa de software que hay entre las aplicaciones y servicios, y las plataformas (SO y hardware). El **Middleware [2]** es realmente un sistema de software que ofrece servicios y funciones comunes para las aplicaciones encargándose generalmete tareas como la gestión de datos, servicios de aplicaciones, mensajería, autenticación y gestión de API ayudando así a los desarrolladores a poder diseñar aplicaciones con una mejor eficiencia, rentabilizando así el desarrollo y ejecución de aplicaciones a gran escala.

1.3 Middleware

El escenario más común hoy en día en desarrollo de software es la integración de aplicaciones, donde hay que interpolar los nuevos servicios y aplicaciones con los nuevos que van surgiendo. Es aquí donde el Middleware se le conoce como **"glue technology"** o tecnología de pegamento, ya que hace de puente para unir todos estos servicios y aplicaciones.

- Modelos y Taxonomía

Los paragidmas principales de Middleware ordenados por su taxonomía serían:

- **Orientados a computación**
  - **Remote Procedure Call (RPC)**
  - **Distributed Objects (DOM)**
  - **Component Model (CM)**
  - **Service Oriented Architectures (SOA)**

- **Orientados a datos**
  - **Transactional (Transaction Processing, TP)**
  - **Tuple Space (TS)**
  - **Remote Data Access (RDA)**
- **Orientado a comunicación**
  - **Message-Oriented Middleware (MOM)**

- Remote Procedure Call (RPC)

Se comenta en primer lugar el porque existe RPC, y Stefano lo resume en que con esta tecnología, por ejemplo podemos reducir el tamaño de las aplicaciones al dividirlas en módulos, en vez de tenerlos todos en un programa en un solo ordenador, por lo que se puede tener a un modulo en el cliente que llame remotamente a otros que esten dentro de una arquitectura de servidor.

La tecnología **RPC [3]** funciona utilizando dos instancias especiales llamadas **Stubs**, una para el cliente y otra para el servidor. El **stub cliente** actúa como sustituto del procedimiento del servidor remoto en el lado del cliente y viceversa con el **stub servidor**. Por lo tanto los procedimientos cliente y servidor, llaman a los stubs mediante una llamada local, y estos se encargan de realizar el Marshalling(basado en el estándar de **XDR**) de los datos para enviarlos como mensajes usando como protocolo TCP/IP, y una vez les llegan se encargan también del Unmarshalling y de pasarle los parámetros al cliente o servidor a nivel local, dependiendo del stub que sea. Y una vez terminado se realiza el proceso en sentido contrario para llevarle la respuesta al otro nodo.

Se explica también, que con RPC pueden salir muchas cosas mal, y para esto hay un semantica específica para notificar que ha ocurrido con las llamadas, para saber si todo ha ido bien o ha habido errores, y depende del protocolo que se use de comunicación (TCP o UDP), pero se basa en: **Una vez exactamente**, **Al menos una vez** , C**omo máximo una vez** y **Cero o más veces.**

- Message-Oriented Middleware (MOM)

Se basa en una abstracción de una cola de mensajes(deben ser tipados) entre procesos siendo indirecta y asíncrona. Este modelo basado en **Productor/Consumidor**, provee de garantías de entregas de mensaje puesto que si una aplicación manda un mensaje a otra, y esta no está funcionando, **MOM** se encarga de guardar el mensaje y entregarlo cuando la aplicación que es la consumidora del mensaje este en funcionamiento.

De este modelo se evolucionó al modelo de **Publicador/Suscriptor**(o Sistemas Distribuidos Basados en Eventos (**DEBS**))**, [4]** el cual se basa en que el Publi-

cador, el cuál no sabe quien es el destinatario del mensaje, crea y envía mensajes sobre un tema, al cuál Suscriptor está suscrito y recibe mensajes de él. Esta comunicación puede ser push(se envía directamente el mensaje) o pull(se decide cuando obtener el mensaje).

Posee también propiedades de desacoplamiento espaciales, temporales y de sincronización.

- Tuple Space (TS)

Se basa en la implementación de memoria asociativa donde las aplicaciones se comunican mediante la abstracción de un espacio virtual compartido.**[5]** Este espacio son **entidades lógicas**, en las cuáles los participantes de la aplicación distribuida, comparten ese espacio y en el **publican sus datos** como tuplas **en ese espacio**, y luego las aplicaciones consumidoras, **utilizan esos datos del espacio** que coincidan con un patrón determinado.

Este modelo se comenta que es muy adecuado para el paradigma de computación del **Master/Worker**, y que Java lo tiene implementado en JavaSpace como parte de la tecnología **Jini**.

- Distributed Objects (DOM)

Sobre los objetos distribuidos, primero se empieza comentando que es un objeto y porque es mejor. Esto tiene que ver según Stefano, con que en un programa normal, nosotros tenemos las operaciones, las cuales se realizan con unos datos a menor nivel que desconocemos, mientras que en los objetos, tenemos a nuestra disposición, tanto la estructura de datos, como las operaciones sobre esos datos.

Después de esto, se lleva al terreno del paradigma **Cliente/Servidor,** donde se habla que los **Servidores** y los **Objetos son el mismo concepto**, siendo estas entidades reactivas, diferenciandose solo, en que los Objetos están en el mismo **espacio de direcciones** que el Cliente, mientras que el Servidor no. Por lo tanto, la tecnología de middleware de los objetos distribuidos permite invocar métodos de objetos remotos. Se puede decir que es similar a **RPC** pero en un **programa orientado a objetos** ya que hace uso de Stubs.

Se utiliza de ejemplo **Java RMI,** aportando así la información, de que los clientes necesitan una referencia del objeto al que van a invocarle un método remoto, y esta tecnología, tiene un registro, donde guarda las referencias a todos los objetos remotos, por tanto el cliente, consulta el registro primero, obtiene la referencia, y después usa esa referencia para invocar al método remoto del objeto referenciado.

Este modelo es la segunda generación del software de sistemas distribuidos, siendo la primera la del Cliente/Servidor. Tiene algunos problemas y es por ello que aparece la tercera generación, que es el modelo de Componentes.

- Component Model (CM)

Los componentes son **modulos software** con una interfaz como los objetos, pero tienen las **dependencias definidas en las interfaces,** y son módulos reusables y ejecutables, y se pueden sumar a otros módulos para crear módulos más complejos. Son **entidades más grandes que los objetos**,

Con este modelo aparece el concepto de **Contenedores,** que son un entorno de ejecución de aplicaciones, con todas las dependencias, librerías y configuración que necesiten , estando aislados y pudiendo ser ejecutados en cualquier sitio.

- Service Oriented Architectures (SOA)

Esta arquitectura, es parte también de la tercera generación, junto con el CM , es unenfoque para acoplarse libremente mediante protocolos independientes basados en estándares de la computación distribuida. Todos los recursos softwares que estén disponibles en la red son considerados servicios, siendo estos una solución de tecnología empresarial.

Tiene unos principios fundamentales como son: Reutilización, modularidad , interoperabilidad, que cumplan los estándares, etc.

Los bloques más importantes que componene SOA son el **Registro de Servicios**, que guarda interfaces y información de acceso de los servicios; el **Proveedor de Servicios**, que es el que crea el servicio y lo publica en el Registro; y el **Solicitante de Servicios**, que obtiene del Registro la información de acceso de los Servicios, y así consigue invocarlos usando uno de esos servicios del Proveedor de Servicio.

- Microservice Arquitectures (MSA)

Es de lo último que se habla en la charla, naciendo de la anterior, siendo estos microservicios una aplicación, que puede ser desarrollada, escalada y testada independientemente y solo tiene un sola responsabilidad.

Stefano comenta sobre esto, mostrando un ejemplo sobre Netflix, que los microservicios, si fallaran por ejemplo, no producen errores en el servicio completo, solo que no se mostraría o daría error el microservicio solo, pudiendo arreglarlo con solo modificar el microservicio, sin necesidad de modificar nada más. Además sobre su mantenibilidad, se dice que no hay entornos de test y entornos de producción, sino solo uno de producción, el cual puede lanzar actualizaciones sobre el microservicio cada unos pocos minutos, siendo muy flexible y evolutivo.

1.4 Conclusiones

Las conclusiones sacadas de esta charla, son varias, basadas en lo importante que es el uso del Middleware a la hora de utilizar sistemas distribuidos, puesto que es una forma de abordar la complejidad de utilizar diferentes infraestructuras, consiguiendo así agilizar el proceso del desarrollo de aplicaciones. Se puede observar la cantidad de vertientes en las cuáles se mueve el Middleware,(datos, computación y comunicación) teniendo así formas de mejorar los desarrollos en todos los aspectos posibles.

2 Valoración personal

Sobre esta charla, tengo que decir, que en un primer momento me sorprendió, porque no sabía que era en inglés, y no estaba mentalizado para entender bien lo que decía Stefano, eso y que tenía un poco de saturación en su audio y a veces costaba escuchar que decía.

Sobre el contenido en sí, decir que la mayoría de las cosas las habiamos visto conceptualmente en clase, como puede ser el caso de RPC y RMI, cosas de las cuáles hemos realizado hasta prácticas, por lo tanto eran cosas que ya tenía asumidas de antes.

Sin embargo decir, que Stefano le daba un toque de experiencia a todo lo que comentaba, como el caso de cuando preguntaba lo que buscaban los ingenieros de coches de F1, haciendo una comparación con los ingenieros del Middleware. Y también, ha explicado contexto muy útiles, los cuáles utilizamos todos los días, y no sabemos el trasfondo, como puede ser los microservicios etc.

En opinión personal, la charla ha servido para profundizar conocimientos que ya teníamos una base, y aprender algunos que no sabíamos.

Referencias

1. DESSERT research group, http://www.mobilab.unina.it/ , last accessed 2021/05/30.
2. Middleware según RedHat, https://www.redhat.com/es/topics/middleware/what-is-middleware, last accessed 2021/05/31.
3. Funcionamiento de la tecnología RPC , https://www.ionos.es/digitalguide/servidores/know-how/que-es-rpc/#:~:text=La%20llamada%20a%20procedimiento%20remoto,informaci%C3%B3n%20entre%20procesos%20de%20sistema.&text=El%20proceso%20de%20comunicaci%C3%B3n%20con,de%20un%20valor%20de%20funci%C3%B3n. , last accessed 2021/05/31.
4. Pub/Sub según Google Cloud, https://cloud.google.com/pubsub/docs/overview?hl=es , last accessed 2021/06/1.
5. Tuple Model , https://en.wikipedia.org/wiki/Tuple_space#:~:text=A%20tuple%20space%20is%20an,that%20can%20be%20accessed%20concurrently.&text=Producers%20post%20their%20data%20as,that%20match%20a%20certain%20pattern. , last accessed 2021/06/01
6. Transparencias de Stefano Russo, de ahí saco la mayoría de información relatada en el resumen.