

Middleware para sistemas distribuidos

Bruno Campos Uribe
0223329@up.edu.mx

Resumen—Un sistema distribuido es una colección elementos de computación autónomos que aparecen al usuario como un único sistema coherente. Esto se logra con la colaboración de estos elementos que se comunican entre ellos. El Middleware Orientado a Mensajes es un mecanismo basado en comunicación asíncrona diseñado para cumplir las necesidades de un sistema distribuido de alto rendimiento. Un cliente del sistema se puede conectar a uno o varios servidores que actúan como intermediarios para el envío y recepción de mensajes a otros clientes del sistema. Se realizó una calculadora distribuida basada en esta arquitectura.

1. Introducción

La arquitectura de un procesador convencional puede considerarse “internamente distribuido”. Distintos dispositivos separados dentro del procesador son responsables de ciertas tareas específicas (operaciones aritméticas, registros, almacenamiento). Estos dispositivos se interconectan por vías de comunicación que transportan información y mensajes. Estas vías están literalmente cableadas, y los “protocolos” que utilizan para transportar información son específicos y rígidamente definidos. Reorganizar el esquema de distribución implica cambiar la arquitectura física del dispositivo. Este nivel de inflexibilidad ofrece ventajas en términos de velocidad de procesamiento y latencia de transferencia de información.

El desarrollo de dispositivos periféricos, que se limitaban a realizar tareas específicas y se conectaban a dispositivos de computación central, creó la necesidad de desarrollar protocolos por los que varios dispositivos pudieran comunicarse. La popularización de la computadora personal sólo aumentó esta necesidad. El conjunto de protocolos estándar ha evolucionado hacia formar un sistema operativo de red mundial. Cada vez son más irrelevantes el tipo de hardware, sistema operativo y arquitectura de red que utilizan dispositivos específicos, lo que hace que las herramientas para procesar la información requieran cada vez más flexibilidad y disponibilidad. [2]

1.1. Sistemas Distribuidos

Hoy en día utilizamos redes de miles de estaciones de trabajo y computadoras personales para hacer nuestro trabajo, en lugar de enormes procesadores centrales. Esto significa un cambio en el paradigma sobre el que diseñamos programas de computadora. Debemos ser capaces de aprovechar nuestras redes de computadoras más pequeñas para

trabajar de forma conjunta en tareas de mayor complejidad computacional.[2] Estas computadoras suelen estar dispersas geográficamente, por lo que se suele decir que forman un *sistema distribuido*. [5]

Un sistema distribuido debe cumplir dos características. La primera es que un sistema distribuido es una colección de elementos de computación, cada uno de los cuales debe comportarse independiente de los demás. Un elemento puede ser un dispositivo de hardware o un proceso de software. La segunda característica es que para el usuario (ya sea una persona u otra aplicación), un sistema distribuido debe parecer como un único sistema coherente. En un sistema coherente el conjunto de elementos funciona igual, independiente de dónde, cuándo y cómo se produzca la interacción entre un usuario y el sistema. Esto implica que los elementos autónomos tienen que colaborar. Cómo establecer esta colaboración constituye la clave del desarrollo de los sistemas distribuidos.[5]

El diseño de un sistema distribuido debe cumplir cuatro objetivos principales: debe hacer que los recursos sean fácilmente accesibles; debe ocultar el hecho de que los recursos están distribuidos a través de una red; debe ser abierto; y debe ser escalable. [5]

1.1.1. Facilitar el intercambio de recursos. Un objetivo importante de un sistema distribuido es facilitar el acceso y el uso compartido de recursos remotos. Los recursos pueden ser periféricos, instalaciones de almacenamiento, datos, archivos, servicios y redes, entre otros.

La conexión de usuarios y recursos también facilita la colaboración y el intercambio de información.

1.1.2. Transparencia de distribución. Un sistema distribuido intenta que la distribución de procesos y recursos sea transparente, es decir, invisible, para los usuarios finales.

1.1.3. Ser abierto. Un sistema distribuido abierto es un sistema que ofrece componentes que pueden ser fácilmente utilizados o integrados en otros sistemas.

Tabla 1.
DIFERENTES FORMAS DE TRANSPARENCIA EN UN SISTEMA
DISTRIBUIDO. UN OBJETO PUEDE SER UN RECURSO O UN PROCESO[5,
FIGURA 1.2].

Transparencia	Descripción
Acceso	Ocultar las diferencias en la representación de los datos y la forma de acceder a un objeto.
Ubicación	Ocultar dónde se encuentra un objeto.
Reubicación	Ocultar que un objeto puede ser movido a otra ubicación mientras está en uso.
Migración	Ocultar que un objeto puede moverse a otra ubicación.
Réplica	Ocultar que un objeto está replicado.
Concurrencia	Ocultar que un objeto puede ser compartido por varios usuarios independientes.
Fallo	Ocultar el fallo y la recuperación de un objeto.

1.1.4. Escalabilidad. La escalabilidad de un sistema puede medirse en al menos tres dimensiones diferentes[4]:

Tamaño: Un sistema puede ser escalable con respecto a su tamaño, lo que significa que podemos añadir fácilmente más usuarios y recursos al sistema sin que haya una pérdida de rendimiento.

Geografía: Un sistema escalable geográficamente es aquel en el que los usuarios y los recursos pueden estar muy alejados entre sí, sin que los retrasos en la comunicación afecten la funcionalidad del sistema.

Administrativa: Un sistema escalable desde el punto de vista administrativo es aquel que puede seguir siendo fácilmente gestionado aunque abarque muchas organizaciones administrativas independientes.

1.2. Arquitectura

Para asegurar el correcto funcionamiento de un sistema distribuido, es crucial sus elementos estén debidamente organizados. La organización de los sistemas distribuidos se refiere sobre todo a los componentes de software que constituyen el sistema. Estas arquitecturas de software nos indican cómo deben organizarse los distintos componentes de software y cómo deben interactuar.[5]

1.2.1. Cliente-servidor. En el modelo más sencillo de cliente-servidor, los procesos de un sistema distribuido se dividen en dos grupos. Un *servidor* es un proceso que implementa un servicio específico. Un *cliente* es un proceso que solicita un servicio a un servidor enviando una petición y espera posteriormente la respuesta del servidor[5].

En esta arquitectura, el servidor se encarga de todo, mientras que el cliente no es más que un terminal mudo. Este es el modelo más sencillo de una *arquitectura por capas*. Un enfoque para organizar los clientes y los servidores consiste en distribuir más de dos capas en diferentes máquinas. Sin embargo, se sigue haciendo una distinción entre sólo dos tipos de máquinas: las máquinas cliente y las máquinas servidoras, lo que conduce a lo que también se denomina una arquitectura (físicamente) de dos niveles[5].

1.2.2. Arquitectura descentralizada. Una distribución en capas es también conocida como una *distribución vertical*. En contraste, en la *distribución horizontal* un cliente o servidor se divide en partes lógicamente equivalentes, cada parte operando en su propia parte del conjunto de datos, equilibrando la carga de procesamiento[5].

1.3. Modelos de Interacción

1.3.1. Comunicación Síncrona. Cuando se llama a un procedimiento en el modelo de síncrono, el código que llama debe bloquear y suspender el procesamiento hasta que el código llamado complete la ejecución y le devuelva el control; el código que llama puede ahora continuar el procesamiento. Cuando se utiliza el modelo de interacción síncrona los sistemas no tienen independencia de control de procesamiento; dependen de la respuesta de los sistemas llamados [1].

El modelo de comunicación síncrona también se ilustra en la Figura 1.

1.3.2. Comunicación Asíncrona. El modelo de interacción asíncrona, ilustrado en la Figura 2, permite a quien llama mantener el control del procesamiento. El programa que llama no necesita bloquearse y esperar a que el código llamado responda. Este modelo permite al elemento que llama continuar el procesamiento independientemente del estado de procesamiento del procedimiento llamado. Con la interacción asíncrona, el procedimiento llamado puede no ejecutarse inmediatamente. Este modelo de interacción requiere un intermediario para gestionar el intercambio de peticiones [1].

1.4. Message-Oriented Middleware

Los sistemas que operan entornos críticos con exigencias de disponibilidad 24/7, alto rendimiento, y alta integridad los sistemas tradicionales centralizados fracasan para cumplir estas necesidades.

Un mecanismo basado en comunicación asíncrona diseñado para cumplir con estas necesidades se conoce como *Middleware Orientado a Mensajes* (MOM).[1] Un cliente de un sistema MOM se conecta a uno o varios servidores

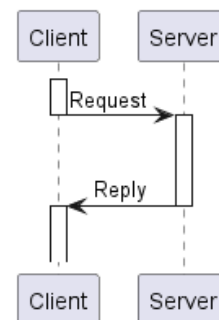


Figura 1. Interacción cliente-servidor.

que actúan como intermediarios para el envío y recepción de mensajes a y de otros clientes del sistema. Las plataformas basadas en MOM permiten crear sistemas flexibles y cohesivos.

Cuando se utiliza MOM, una aplicación emisora no tiene ninguna garantía de que su mensaje será leído por otra aplicación ni se le da una garantía sobre el tiempo que tardará el mensaje en ser entregado. Estos aspectos los determina principalmente la aplicación receptora[1].

1.4.1. Acoplamiento. MOM inyecta una capa entre emisores y receptores. Esta capa independiente actúa como intermediaria para el intercambio de mensajes. El acoplamiento flexible entre los participantes de un sistema enlaza aplicaciones sin tener que adaptar los sistemas de origen y destino entre sí, lo que da lugar a un despliegue de sistemas altamente cohesivo y desacoplado[1].

1.4.2. Confiabilidad. Con MOM, la pérdida de mensajes por fallos de la red o del sistema se evita utilizando un mecanismo de almacenamiento y retransmisión para la persistencia de los mensajes. Esta capacidad de MOM introduce un alto nivel de confiabilidad en el mecanismo de distribución. El almacenamiento y retransmisión evita la pérdida de mensajes cuando partes del sistema no están disponibles o están ocupadas[1].

1.4.3. Escalabilidad. Además de desacoplar la interacción de los subsistemas, MOM también desvincula las características de rendimiento de los subsistemas entre sí. Los subsistemas pueden ampliarse de forma independiente, con poca o ninguna alteración de otros subsistemas. MOM también permite al sistema hacer frente a picos de actividad en un subsistema sin afectar a otras áreas del sistema. Los modelos MOM permiten balanceo de carga, al permitir que un subsistema elija aceptar un mensaje cuando esté preparado para hacerlo[1].

2. Implementación

Se desarrolló una calculadora utilizando una arquitectura MOM. Para implementar esta arquitectura se desarrollaron

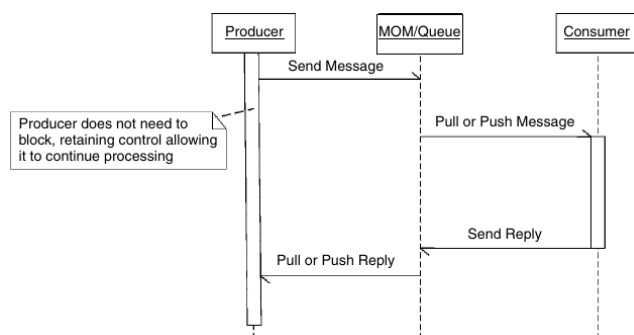


Figura 2. Modelo de interacción asíncrono [1, Figura 1.2].

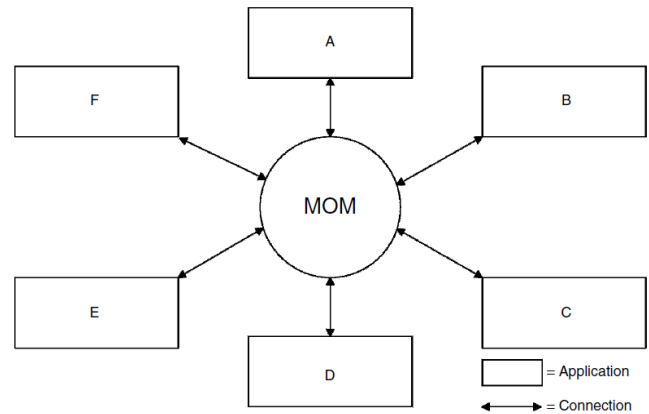


Figura 3. Diagrama de un sistema distribuido basado en MOM [1, Fig. 1.4].

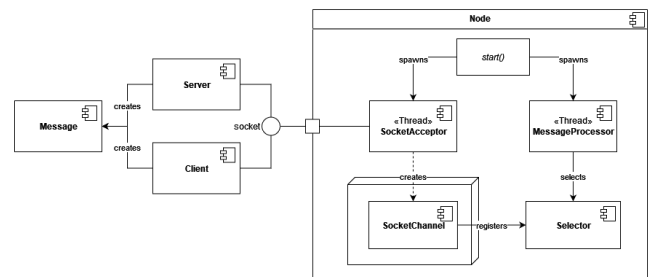


Figura 4. Componentes del Nodo.

los siguientes componentes, que se ilustran en la Figura 4, que se describirán a detalle más adelante.

Mensaje. Encapsula el mensaje para enviar al Nodo. Incluye un encabezado con un código de contenido, que describe el contenido del mensaje.

Nodo. El Nodo es el canal de mensaje que actúa como intermediario entre las células.

Célula. En el contexto de la arquitectura MOM, una célula puede ser cualquier aplicación que se conectará al Nodo para comunicarse con las otras células conectadas. Los distintos tipos de célula distinguen qué mensajes procesar o rechazar basado en su código de contenido. Para la calculadora se desarrollaron los siguientes tipos de célula:

Interfaz de Usuario. Una interfaz gráfica que permite al usuario introducir operaciones y ver los resultados. También mantiene un registro de texto de todos los mensajes que salen y entran. La interfaz se muestra en la Figura 6.

Servidor. Reciben las operaciones de usuario, las evalúan, y envían el resultado.

2.1. Mensaje

El mensaje se implementa como una clase de Java con dos campos: un campo de texto, que es el contenido del

mensaje; y el código de contenido. Se serializan utilizando la interfaz `java.io.Serializable` para poder ser enviados como un buffer de bytes por una conexión TCP/IP.

En esta etapa del proyecto sólo se definen dos códigos de contenido: `OPERATION` y `RESPONSE`, pero la clase se diseña de manera que sea fácil expandir el número de códigos.

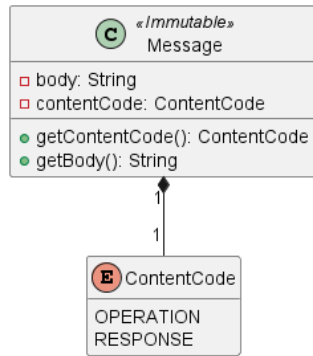


Figura 5. Diagrama de clase del Mensaje.

2.2. Nodo

Cuando se ejecuta el Nodo, inicia un socket TCP de servidor en un puerto proporcionado por el usuario, o el puerto 50000 por defecto. El Nodo se compone de dos hilos concurrentes: en uno se mantiene abierto el socket de servidor y las nuevas conexiones se registran en un Selector. Los sockets que se registran al selector se configuran para notificar un evento de escucha—cuando hay un mensaje pendiente por recibir— El segundo hilo espera para un evento del Selector y procesa los mensajes entrantes; el Nodo propaga los mensajes a todas las conexiones abiertas, a excepción de la que recibió el mensaje.

El nodo se implementó con la librería de comunicación asíncrona de Java `java.nio` [3].

2.3. Células

Cuando se ejecuta una célula (Interfaz o Servidor), se intenta establecer una conexión TCP al Nodo a través del puerto proporcionado por el usuario, o el puerto 50000 por defecto.

Cuando una célula recibe un mensaje del Nodo, puede decidir procesarlo o rechazarlo leyendo su código de contenido. La *Interfaz de Usuario* envía mensajes con código de contenido `OPERATION` y muestra al usuario los mensajes con código de contenido `RESPONSE`. El *Servidor* sólo procesa mensajes con código de contenido `OPERATION` y encapsula sus respuestas con código de contenido `RESPONSE`.

3. Conclusión

Para expandir este modelo basado en mensajes a una arquitectura ADSOA, el siguiente paso en el desarrollo es

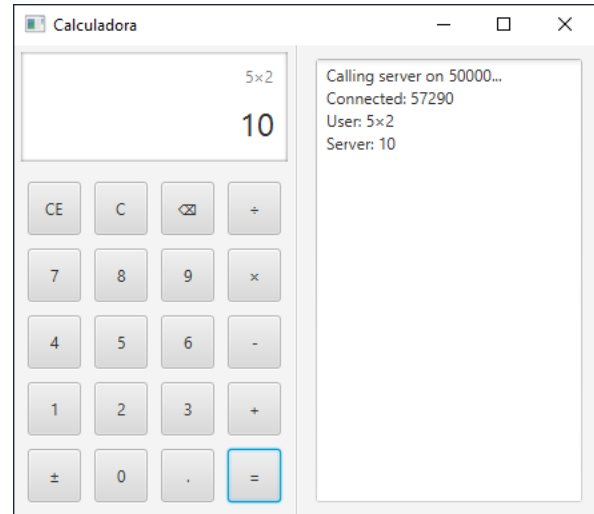


Figura 6. Interfaz gráfica de la calculadora. Se puede observar en el registro el historial de comunicación con el Nodo al que está conectada.

distribuir lo que ahora es la célula de *Servidor* con una arquitectura orientada a servicios (SOA). Esto involucra generar más códigos de contenido para que el sistema pueda distribuir las distintas operaciones.

Para implementar transparencia de réplica, los mensajes deberán ser filtrados de tal manera que el usuario sólo vea una respuesta, sin perder la confiabilidad de un sistema distribuido.

Referencias

- [1] Edward Curry. «Message-Oriented Middleware». En: *Middleware for communications* (jun. de 2004), págs. 1-28. DOI: 10.1002/0470862084.
- [2] Jim Farley. *Java distributed computing*. O'Reilly, 1998. URL: <https://docstore.mik.ua/oreilly/java-ent/dist/>.
- [3] *Java Development Kit version 17 API specification. java.nio*. Oracle. URL: <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/nio/package-summary.html>.
- [4] B Neuman. «Scale in Distributed Systems». En: *Readings in Distributed Computing Systems* (1994).
- [5] Maarten van Steen y Andrew S. Tanenbaum. *Distributed Systems*. 3.^a ed. Published by Maarten van Steen, 2017. URL: <https://www.distributed-systems.net/index.php/books/ds3/>.