GUIA SOBRE LA ARQUITECTURA ORIENTADA A EVENTOS

OBJETIVO

- Ofrecer una guía conceptual del patrón arquitectónico orientado a eventos que permita tener una visión completa de este patrón.

Arquitectura orientada a eventos

- Un ejemplo real donde se aplica esta arquitectura se puede apreciar en la Figura 1. Se trata de un sistema de cámaras que eviten mensajes que posteriormente serán utilizados por varios procesadores. A continuación se explican cada uno de sus componentes.

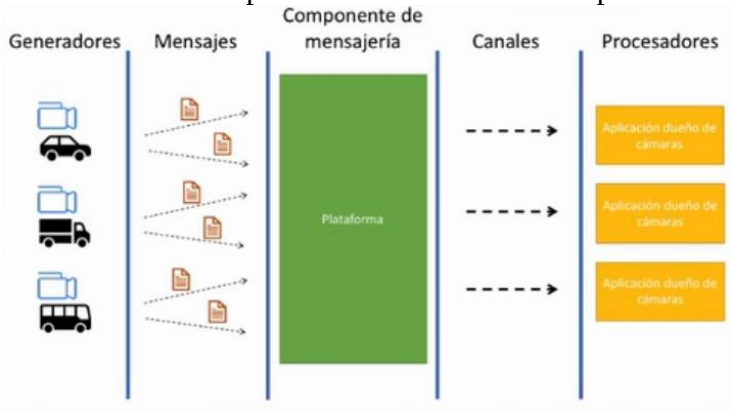

Figura 1. Ejemplo real utilizando la arquitectura orientada a eventos

  - **Mensajes** contienen la información del evento.
  - El componente de **mensajería** se encarga de recibir el mensaje, procesarlo si es necesario e informar a las partes interesadas.
  - Los **canales** crean el puente entre el componente de mensajería y los receptores. Los canales se pueden implementar con colas de mensajes o mediante el patrón publicador/subscriptor.
  - Los **procesadores** o consumidores son los que están interesados en esos eventos. Los procesadores tienen una una responsabilidad de lógica de negocio. Deben estar lo más desacoplados (independientes, que no tienen conocimiento de otros componentes) posibles de otras partes de la aplicación, es decir, no deberían tener conocimiento de quienes generan los eventos, ni de otros procesadores del sistema.
- En cuanto a la topología (la manera cómo organizamos los componentes) se tienen dos opciones: mediante un *mediador* o con un *broker*.
- El **mediador** es un componente dentro del componente de mensajería que se encarga de sincronizar todos los procesadores. Se encarga que los procesadores se ejecuten en la secuencia correcta para producir el resultado deseado. Es decir, toma el mensaje y se lo pasa a un procesador, luego recibe la respuesta y se lo pasa a otro procesador (ver Figura 2). Algunos

procesadores se pueden ejecutar de manera secuencial, otros en paralelo. El mediador hace esta orquestación correcta. Esos eventos que toma el mediador lo hace a partir de una **cola**. Puede haber una o varias colas dependiendo de qué tanta escalabilidad se necesite en el sistema. Para implementar un mediador se requiere partes de terceros, por su complejidad.

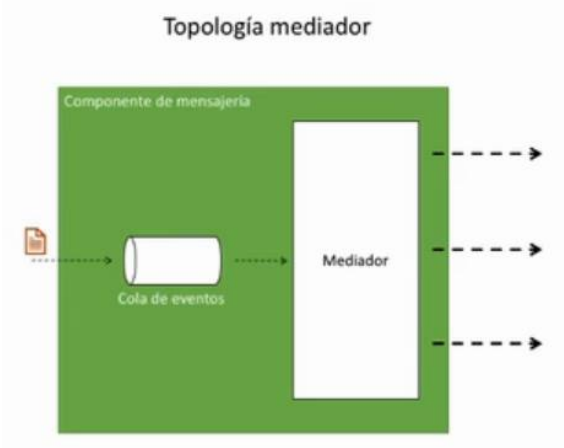

Figura 2. Componente de mensajería con mediador

- La segunda topología es un **broker** (agente, intermediario). Es más simple que un mediador porque simplemente recibe el evento (no lo pone en una cola) y lo pone en el canal apropiado (no hace orquestación). Lo que si hace es que el procesador una vez procesa el evento recibido, generan un evento nuevo y se lo envían al broker, y luego ese evento lo recibe otro procesador. Así se van encadenando las tareas a procesar.

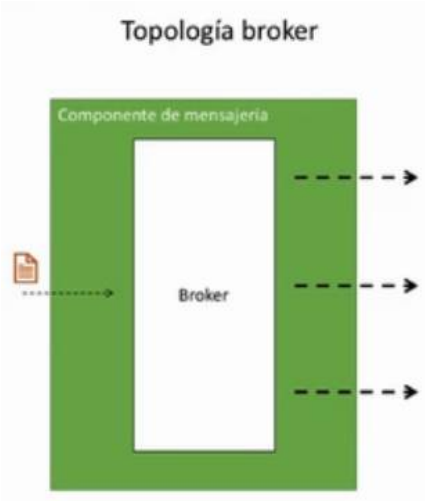

Figura 2. Componente de mensajería con broker

- Es posible hace una implementación de este patrón desde cero, es decir, no utilizando framework mediador. La otra opción es utilizar un framework mediador como RabbitMQ o Apache Kafka. Se recomienda hacer el Hola Mundo con RabbitMQ siguiendo este enlace https://www.rabbitmq.com/getstarted.html.
- ¿Cuándo usar esta arquitectura?

- **Beneficios**: altamente escalable, altamente flexible. Esto debido al desacople entre los productores de eventos, mensajería central y procesadores. Se puede utilizar en aplicaciones pequeñas (un par de componentes, web) y grandes. Los componentes están altamente desacoplados, esto garantiza que se puedan agregar nuevos componentes sin mayor problema, sin tener que modificar lo que ya se tiene (o modificando muy poco).
- **Desventajas y retos**. No es tan fácil de implementar. Hay retos como, ¿qué pasa si se envía un mensaje y uno de los procesadores no está disponible? Se requiere lógica adicional como: reenvío de mensajes. El otro reto, es ¿qué pasa si el procesador está conectado, recibe el mensaje pero no responde? ¿Se debe determinar cuánto tiempo esperar, en cuanto tiempo se considera que la petición es fallida? El tercer reto ¿Qué pasa si el broker o mediados se cae? Qué pasaría entonces, si los generadores de eventos envían mensajes y el broker no está disponible.
- **Otra desventa** radica en que es difícil ejecutar transacciones. Si las operaciones de una transacciones tiene que ser implementas por distintos consumidores, entonces, es algo difícil de implementar (así haya mediador).
- **Otra desventaja**, es difícil controlar y mantener los contratos entre las partes del sistema. Se aconseja ir versionando los contratos de mensajes.
- **¿Cuándo usarlo?** (i) Aplicaciones con múltiples subsistemas o componentes, que necesitan procesar los eventos al mismo tiempo (como en el ejemplo de las cámaras, donde muchos procesadores requieren procesar los eventos) (ii) aplicaciones que tiene que procesar datos en tiempo real. A pesar que hay colas, los eventos se envían muy rápido. (iii) Aplicaciones de internet de las cosas (como en el ejemplo de las cámaras – relojes, electrodomésticos). (iv) procesamiento de grandes volúmenes de datos de redes sociales. En general se usa esta arquitectura cuando se producen eventos y en *event sourcing* (provisionamiento de eventos), esto es, una base de datos que almacena información de eventos que ocurren.

- **Calificación**:
  - **Agilidad en los cambios**. Alta.
  - **Rendimiento**. Alta. Es asíncrono, por lo que tiene buen rendimiento.
  - **Escalabilidad**. Alta. Cada procesador se puede desarrollar y desplegar por separado, entonces se escala sin tiener que escalar toda la aplicación.
  - **Facilidad de despliegue**. Regular. Depende cómo esté hecho. Si se usa mediador, éste está pegado a los consumidores. Por el lado del broker es más fácil, cada consumidor es independiente.
  - **Testeabilidad**. Baja. Para poderse probar bien, se necesita generar eventos, esto no es tan fácil de coordinar. Además todo lo asíncrono es más difícil de probar.
  - **Facilidad de desarrollo**. Bajo. No es tan fácil de empezar a desarrollar. Cuando hay errores es algo difícil de resolverlos.