W. LIBARDO PANTOJA YEPEZ

RESUMEN DEL LIBRO: FUNDAMENTALS OF SOFTWARE ARCHITECTURE

Resumen del libro

Popayán
Septiembre de 2021

Table de Contenido

Capitulo 1. Estilo de arquitectura dirigida por eventos

El estilo de arquitectura impulsada por eventos es un estilo de arquitectura **asíncrona distribuida** popular que se utiliza para producir aplicaciones altamente *escalables* y de *alto rendimiento*. También es muy adaptable y se puede utilizar tanto para aplicaciones pequeñas como para aplicaciones grandes y complejas. La arquitectura impulsada por eventos se compone de componentes de procesamiento de eventos desacoplados que reciben y procesan eventos de forma asincrónica. Se puede usar como un estilo de arquitectura independiente o incrustado en otros estilos de arquitectura (como una arquitectura de microservicios impulsada por eventos).

La mayoría de las aplicaciones siguen lo que se llama un modelo basado en solicitudes (ilustrado en la Figura 1). En este modelo, las solicitudes realizadas al sistema para realizar algún tipo de acción se envían a un orquestador de solicitudes. El orquestador de solicitudes suele ser una interfaz de usuario, pero también se puede implementar a través de una capa de API o un bus de servicio empresarial. La función del *orquestador de solicitudes* es dirigir de forma **determinista** y **sincrónica** la solicitud a varios procesadores de solicitudes. Los procesadores de solicitudes manejan la solicitud, ya sea recuperando o actualizando información en una base de datos.

Un buen ejemplo del modelo basado en solicitudes es una solicitud de un cliente para recuperar su historial de pedidos de los últimos seis meses. La recuperación de la información del historial de pedidos es una solicitud determinista basada en datos que se hace al sistema para obtener datos dentro de un contexto específico, no un evento que sucede al que el sistema debe reaccionar.

Un modelo basado en **eventos**, por otro lado, *reacciona a una situación particular y toma medidas basadas en ese evento*. Un ejemplo de un modelo basado en eventos es *presentar una oferta por un artículo* en particular dentro de una subasta en línea. El envío de la oferta no es una solicitud realizada al sistema, sino más bien *un evento que ocurre* después de que se anuncia el precio de venta actual. El sistema *debe responder a este evento* comparando la oferta con otras recibidas al mismo tiempo para determinar quién es el mejor postor actual.

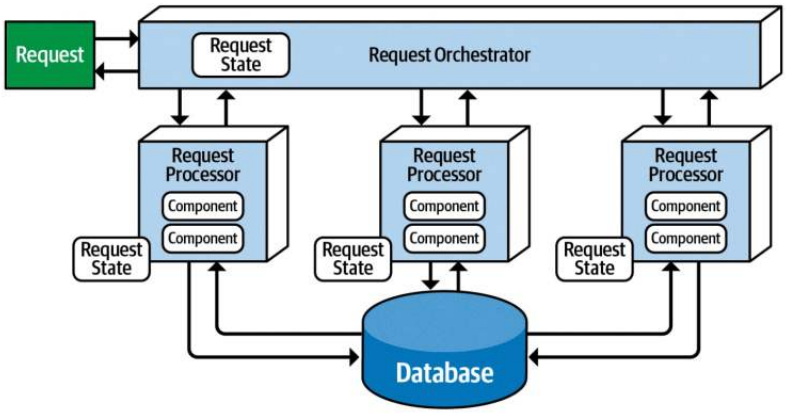

Figure 1: Modelo basado en solicitudes.

1.1. Topología

Hay dos topologías principales dentro de la arquitectura impulsada por eventos: la topología del **mediador** (mediator) y la topología del **intermediario** (broker). La topología de mediador se usa comúnmente cuando se requiere control sobre el flujo de trabajo de un proceso de eventos, mientras que la topología de intermediario se usa cuando se requiere un alto grado de capacidad de respuesta y control dinámico sobre el procesamiento de un evento. Debido a que las características de la arquitectura y las estrategias de implementación difieren entre estas dos topologías, es importante comprender cada una para saber cuál es la más adecuada para una situación particular.

1.2. Topología del intermediario (broker)

La topología del intermediario se diferencia de la topología del mediador en que *no hay un mediador de eventos central*. Más bien, *el flujo de mensajes se distribuye a través de los componentes del procesador de eventos en una forma de transmisión en cadena* a través de un intermediario de mensajes livianos (como RabbitMQ, ActiveMQ, HornetQ, etc.). Esta topología es útil cuando tiene *un flujo de procesamiento de eventos relativamente simple y no necesita orquestación* y *coordinación* central de eventos.

Hay cuatro componentes de arquitectura primarios dentro de la topología del intermediario: un *evento de inicio*, el *intermediario* de eventos, un *procesador de eventos* y un *evento de procesamiento*. El evento de inicio es el evento inicial que inicia todo el flujo del evento, ya sea un evento simple como hacer una oferta en una subasta en línea o eventos más complejos en un sistema de beneficios de salud como cambiar de trabajo o casarse. El evento de inicio se envía a un canal de eventos en el intermediario de eventos para su procesamiento. Dado que no hay un

componente mediador en la topología del broker que gestiona y controla el evento, un solo procesador de eventos acepta el evento iniciador del intermediario de eventos y comienza el procesamiento de ese evento. El procesador de eventos que aceptó el evento iniciador realiza una tarea específica asociada con el procesamiento de ese evento, luego anuncia de forma asincrónica lo que hizo al resto del sistema creando lo que se llama un evento de procesamiento. Este evento de procesamiento se envía de forma asincrónica al intermediario de eventos para su procesamiento posterior, si es necesario. Otros procesadores de eventos escuchan el evento de procesamiento, reaccionan a ese evento haciendo algo y luego anuncian a través de un nuevo evento de procesamiento lo que hicieron. Este proceso continúa hasta que nadie está interesado en lo que hizo un procesador de eventos final. La Figura 2 ilustra este flujo de procesamiento de eventos.

El broker de eventos suele estar federado (es decir, instancias agrupadas en clúster basadas en múltiples dominios), donde cada broker federado contiene todos los canales de eventos utilizados dentro del flujo de eventos para ese dominio en particular. Debido a la naturaleza de transmisión asíncrona desacoplada y olvidada de la topología del broker, los temas (o intercambios de temas en el caso de AMQP) se utilizan generalmente en la topología del broker mediante un *modelo de mensajería de publicación y suscripción*.

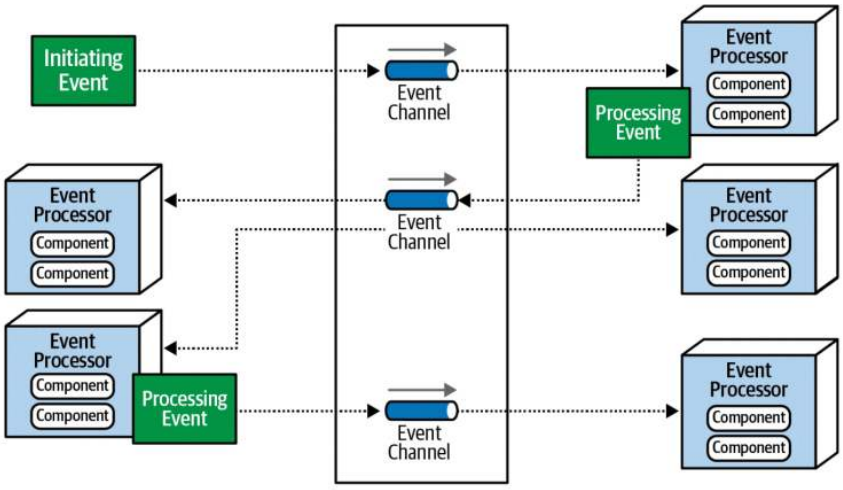

Figure 2: Topología de broker.

Siempre es una buena práctica dentro de la topología del broker que cada procesador de eventos anuncie lo que hizo al resto del sistema, independientemente de si a cualquier otro procesador de eventos le importa o no cuál fue esa acción. Esta práctica proporciona extensibilidad arquitectónica si se requiere funcionalidad adicional para el procesamiento de ese evento. Por ejemplo, suponga que como parte de un proceso de evento complejo, como se ilustra en la Figura 3, se genera un correo electrónico y se envía a un cliente notificándole de una acción particular tomada. El procesador de eventos de notificación generaría y enviaría el correo electrónico, luego anunciaría esa acción al resto del sistema a través de un nuevo evento de procesamiento enviado a un tema. Sin embargo, en este caso, ningún otro procesador de eventos está escuchando eventos sobre ese tema y, como tal, el mensaje simplemente desaparece.

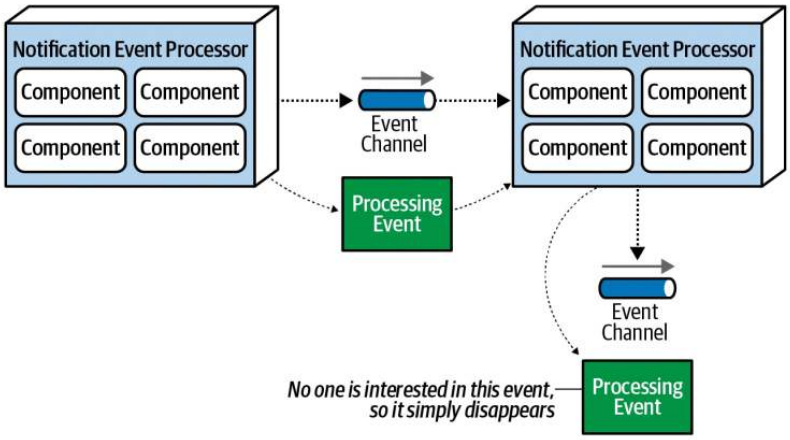

Figure 3: El evento de notificación se envía pero se ignora.

Este es un buen ejemplo de extensibilidad arquitectónica. Si bien puede parecer una pérdida de recursos enviar mensajes que se ignoran, no lo es. Supongamos que surge un nuevo requisito para analizar los correos electrónicos que se han enviado a los clientes. Este nuevo procesador de eventos se puede agregar al sistema general con un esfuerzo mínimo porque la información del correo electrónico está disponible a través del tema del correo electrónico en el nuevo analizador sin tener que agregar ninguna infraestructura adicional o aplicar cambios a otros procesadores de eventos.

Para ilustrar cómo funciona la topología del broker, considere el flujo de procesamiento en un sistema típico de entrada de pedidos minoristas, como se ilustra en la Figura 4, donde se coloca un pedido para un artículo (digamos, un libro). En este ejemplo, el procesador de eventos *OrderPlacement* recibe el evento de inicio (*PlaceOrder*), inserta el pedido en una tabla de base de datos y devuelve un ID de pedido al cliente. Luego anuncia al resto del sistema que creó un pedido a través de un evento de procesamiento creado por pedido. Observe que tres procesadores de

eventos están interesados en ese evento: el procesador de eventos de notificación, el procesador de eventos de pago y el procesador de eventos de inventario. Los tres procesadores de eventos realizan sus tareas en paralelo.

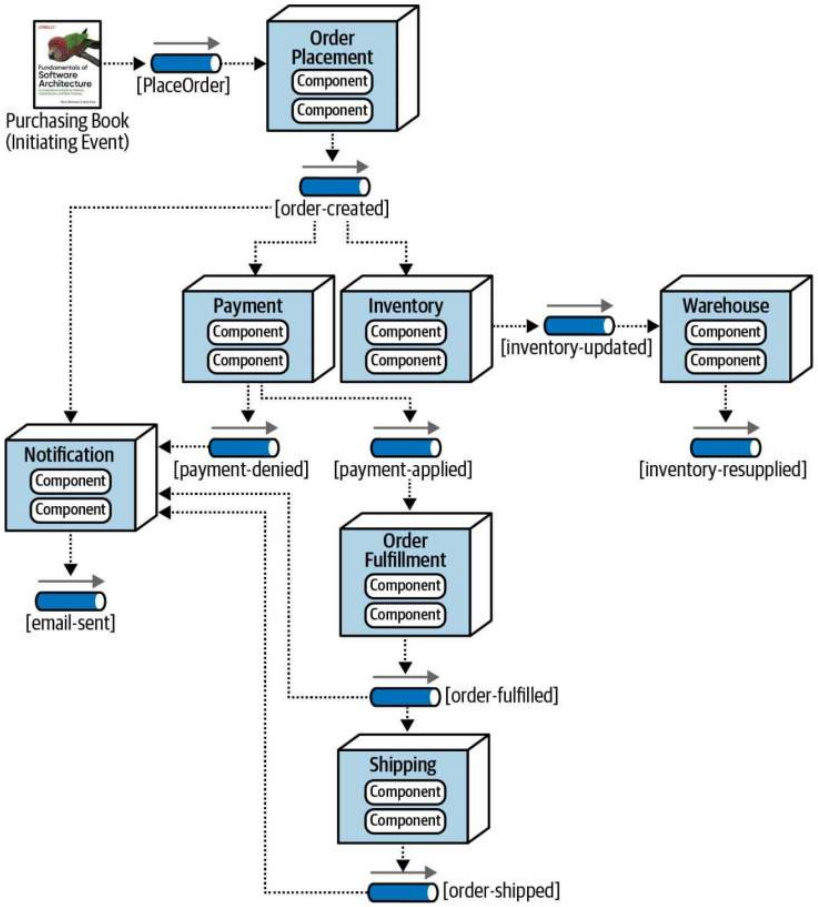

Figure 4: Ejemplo de topología de broker

El *procesador de eventos de notificación* recibe el evento de procesamiento creado por el pedido y envía un correo electrónico al cliente. Luego genera otro evento de procesamiento (correo electrónico enviado). Tenga en cuenta que ningún otro procesador de eventos está escuchando ese evento. Esto es normal e ilustra el ejemplo anterior que describe la extensibilidad arquitectónica: un gancho en el lugar

para que otros procesadores de eventos puedan eventualmente aprovechar esa fuente de eventos, si es necesario.

El *procesador de eventos de Inventario* también escucha el evento de procesamiento creado por pedido y disminuye el inventario correspondiente para ese libro. Luego anuncia esta acción a través de un evento de procesamiento actualizado de inventario, que a su vez es recogido por el procesador de eventos de Almacén para administrar el inventario correspondiente entre almacenes, reordenando los artículos si los suministros son demasiado bajos.

El *procesador de eventos de pago* también recibe el evento de procesamiento creado por el pedido y carga en la tarjeta de crédito del cliente el pedido que se acaba de crear. Observe en la Figura **4** que se generan dos eventos como resultado de las acciones tomadas por el procesador de eventos de Pago: uno para notificar al resto del sistema que se aplicó el pago (pago aplicado) y un evento de procesamiento para notificar al resto del sistema que el pago fue denegado (pago denegado). Tenga en cuenta que el procesador de eventos de Notificación está interesado en el evento de procesamiento de pago denegado, ya que debe, a su vez, enviar un correo electrónico al cliente informándole que debe actualizar la información de su tarjeta de crédito o elegir un método de pago diferente.

El procesador de eventos *OrderFulfillment* escucha el evento de procesamiento aplicado al pago y realiza la selección y empaquetado de pedidos. Una vez completado, anuncia al resto del sistema que cumplió con el pedido a través de un evento de procesamiento de pedidos cumplidos. Observe que tanto la unidad de procesamiento de notificaciones como la unidad de procesamiento de envíos escuchan este evento de procesamiento. Al mismo tiempo, el procesador de eventos de notificación notifica al cliente que el pedido se ha cumplido y está listo para su envío y, al mismo tiempo, el procesador de eventos de envío selecciona un método de envío. El procesador de eventos de envío envía el pedido y envía un evento de procesamiento de pedido enviado, que el procesador de eventos de notificación también escucha para notificar al cliente sobre el cambio de estado del pedido.

Al analizar el ejemplo anterior, observe que todos los procesadores de eventos están altamente desacoplados e independientes entre sí. La mejor manera de comprender la topología del broker es pensar en ella *como una carrera de relevos*. En una carrera de relevos, los corredores sostienen un bastón (un palo de madera) y corren una cierta distancia (digamos 1,5 kilómetros), luego entregan el bastón al siguiente corredor y así sucesivamente hasta que el último corredor cruce la línea de meta. En las carreras de relevos, una vez que un corredor entrega el testigo, ese corredor termina con la carrera y pasa a otras cosas. Esto también es cierto con la topología de broker. Una vez que un procesador de eventos entrega el evento, ya no está involucrado en el procesamiento de ese evento específico y está disponible para reaccionar a otros eventos de inicio o procesamiento. Además, cada procesador de eventos puede escalar independientemente uno del otro para manejar

condiciones de carga variables o copias de seguridad en el procesamiento dentro de ese evento. Los temas proporcionan el punto de contrapresión si un procesador de eventos se apaga o se ralentiza debido a algún problema del entorno.

Si bien el *rendimiento*, la *capacidad de respuesta* y la *escalabilidad* son grandes beneficios de la topología del intermediario, también tiene algunos aspectos negativos. En primer lugar, *no hay control sobre el flujo de trabajo genera*l asociado con el evento de inicio (en este caso, el evento *PlaceOrder*). Es muy dinámico basado en varias condiciones, y nadie en el sistema sabe realmente cuándo se completa la transacción comercial de realizar un pedido. El *manejo de errores* también es un gran desafío con la topología de broker. Debido a que no hay un mediador que supervise o controle la transacción comercial, si se produce una falla (como que el procesador de eventos de pago falla y no completa su tarea asignada), *nadie en el sistema se da cuenta de esa falla*. El proceso empresarial se atasca y no puede moverse sin algún tipo de intervención automática o manual. Además, todos los demás procesos avanzan sin tener en cuenta el error. Por ejemplo, el procesador de eventos de Inventario aún reduce el inventario y todos los demás procesadores de eventos reaccionan como si todo estuviera bien.

La capacidad de reiniciar una transacción comercial (*recuperabilidad*) también es algo que no es compatible con la topología del broker. Debido a que se han realizado otras acciones de forma asincrónica durante el procesamiento inicial del evento de inicio, no es posible volver a enviar el evento de inicio. Ningún componente en la topología del broker conoce el estado o incluso es propietario del estado de la solicitud comercial original y, por lo tanto, nadie es responsable en esta topología de reiniciar la transacción comercial (el evento de inicio) y saber dónde se detuvo. Las ventajas y desventajas de la topología de broker se resumen en la Tabla 1.

Table 1: Ventajas y desventajas del broker

Ventajas	*Desventajas*
Procesadores de eventos altamente desacoplados	Control de flujo de trabajo
Alta escalabilidad	Gestión del error
Alta capacidad de respuesta	Recuperabilidad
Alto desempeño	Reiniciar capacidades
Alta tolerancia a fallos	Inconsistencia de datos

1.3. Topología de mediador (mediator)

La topología de mediador de la arquitectura dirigida por eventos aborda algunas de las deficiencias de la topología de intermediario descrita en la sección anterior. Un *elemento central* de esta topología *es un mediador de eventos*, que administra y controla el flujo de trabajo para iniciar eventos que requieren la coordinación de múltiples procesadores de eventos. Los componentes de la arquitectura que conforman la topología del mediador son un *evento de inicio*, *una cola de eventos*, *un mediador de eventos*, *canales de eventos* y *procesadores de eventos*. Al igual que en la topología del broker, el evento de inicio es el evento que inicia todo el proceso de eventos. A diferencia de la topología del broker, el evento de inicio se envía a una cola de eventos de inicio, que es aceptada por el mediador de eventos. El mediador de eventos solo conoce los pasos involucrados en el procesamiento del evento y, por lo tanto, genera los eventos de procesamiento correspondientes que se envían a los canales de eventos dedicados (generalmente colas) en una forma de mensajería punto a punto. Los procesadores de eventos luego escuchan los canales de eventos dedicados, procesan el evento y, por lo general, responden al mediador que han completado su trabajo. A diferencia de la topología del intermediario, los procesadores de eventos dentro de la topología del mediador *no anuncian lo que hicieron al resto del sistema*. La topología del mediador se ilustra en la Figura 5.

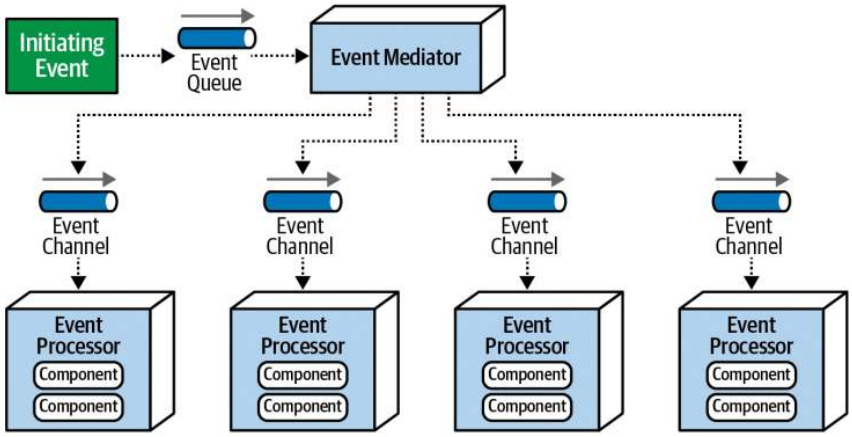

Figure 5: Topología de mediador.

En la mayoría de las implementaciones de la topología del mediador, existen *múltiples mediadores*, generalmente asociados con un dominio particular o agrupación de eventos. Esto reduce el problema del punto único de falla asociado con esta topología y también aumenta el rendimiento y el rendimiento general. Por ejemplo, puede haber un mediador de clientes que maneja todos los eventos relacionados con el cliente (como el registro de un nuevo cliente y la actualización del perfil) y otro mediador que maneja las actividades relacionadas con el pedido (como agregar un artículo a un carrito de compras y pagar).

El mediador de eventos se puede implementar de diversas formas, dependiendo de la naturaleza y complejidad de los eventos que está procesando. Por ejemplo, para eventos que requieren un manejo y una orquestación de errores simples, un mediador como Apache Camel, Mule ESB o Spring Integration generalmente será suficiente. Los flujos de mensajes y las rutas de mensajes dentro de estos tipos de mediadores generalmente se escriben de manera personalizada en código de programación (como Java o C #) para controlar el flujo de trabajo del procesamiento de eventos.

Sin embargo, si el flujo de trabajo del evento requiere mucho procesamiento condicional y múltiples rutas dinámicas con directivas complejas de manejo de errores, entonces un mediador como Apache ODE o Oracle BPEL Process Manager sería una buena opción. Estos mediadores se basan en Business Process Execution Language (BPEL), una estructura similar a XML que describe los pasos involucrados en el procesamiento de un evento. Los artefactos BPEL también contienen elementos estructurados que se utilizan para el manejo de errores, redirección, multidifusión, etc. BPEL es un lenguaje poderoso pero relativamente complejo de aprender y, como tal, generalmente se crea utilizando herramientas de interfaz gráfica proporcionadas en el paquete de motores BPEL del producto.

BPEL es bueno para flujos de trabajo complejos y dinámicos, pero no funciona bien para aquellos flujos de trabajo de eventos que requieren transacciones de larga duración que involucran la intervención humana durante todo el proceso del evento. Por ejemplo, suponga que se realiza una operación a través de un evento de inicio de operación de lugar. El mediador del evento acepta este evento, pero durante el procesamiento encuentra que se requiere una aprobación manual porque la operación supera una cierta cantidad de acciones. En este caso, el mediador del evento tendría que detener el procesamiento del evento, enviar una notificación a un comerciante senior para la aprobación manual y esperar a que se produzca esa aprobación. En estos casos, se necesitaría un motor de gestión de procesos de negocio (BPM) como jBPM.

Es importante conocer los tipos de eventos que serán procesados a través del mediador para poder tomar la decisión correcta para la implementación del evento mediador. Elegir Apache Camel para eventos complejos y de larga duración que involucren interacción humana sería extremadamente difícil de escribir y mantener. Del mismo modo, el uso de un motor BPM para flujos de eventos simples llevaría

meses de esfuerzo inútil cuando se podría lograr lo mismo en Apache Camel en cuestión de días.

Dado que es raro tener todos los eventos de una clase de complejidad, recomendamos clasificar los eventos como *simples*, *difíciles* o complejos y que cada evento pase siempre por un mediador simple (como Apache Camel o Mule). El mediador simple puede entonces interrogar la clasificación del evento y, basándose en esa clasificación, manejar el evento en sí mismo o reenviarlo a otro mediador de eventos más complejo. De esta manera, todo tipo de eventos pueden ser procesados de manera efectiva por el tipo de mediador necesario para ese evento. Este modelo de delegación de mediadores se ilustra en la Figura 6.

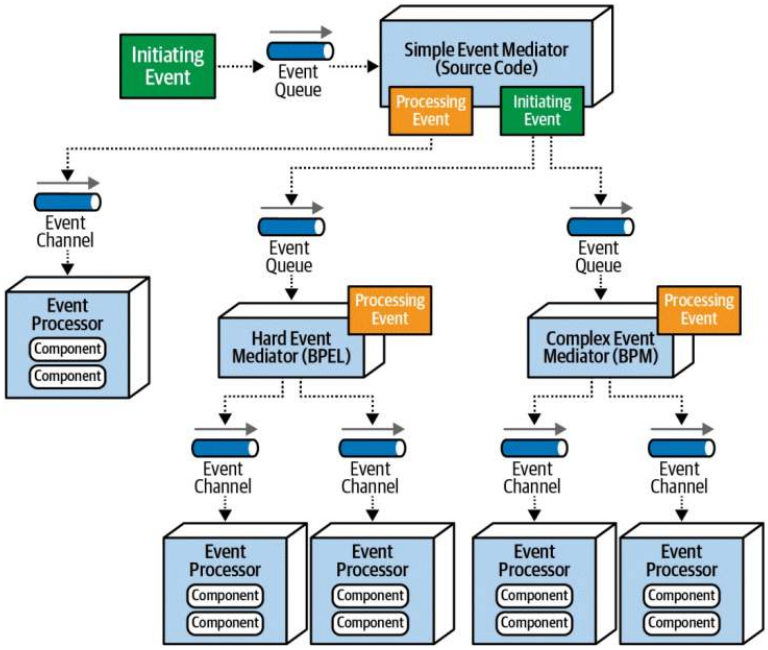

Figure 6: Delegar el evento al tipo apropiado de mediador de eventos..

Observe en la Figura 6 que el *Mediador de eventos simple* genera y envía un evento de procesamiento cuando el flujo de trabajo del evento es simple y puede ser manejado por el mediador simple. Sin embargo, observe que cuando el evento iniciador que ingresa al Mediador de evento simple se clasifica como difícil o complejo, reenvía el evento iniciador original a los mediadores correspondientes (BPEL o BMP). El Mediador de eventos simple, habiendo interceptado el evento original, aún puede ser responsable de saber cuándo ese evento está completo, o simplemente delega todo el flujo de trabajo (incluida la notificación al cliente) a los otros mediadores.

Para ilustrar cómo funciona la topología del mediador, considere el mismo ejemplo del sistema de venta al por menor o de entrada que se describe en la sección de topología del intermediario anterior, pero esta vez utilizando la topología del mediador. En este ejemplo, el mediador conoce los pasos necesarios para procesar este evento en particular. Este flujo de eventos (interno al componente mediador) se ilustra en la Figura 7.

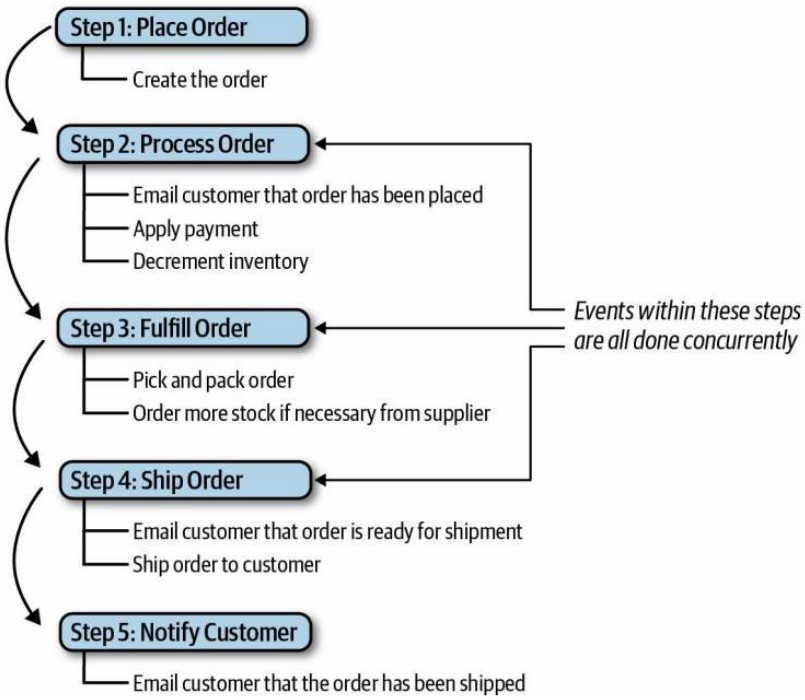

Figure 7: Pasos del mediador para realizar un pedido

De acuerdo con el ejemplo anterior, el mismo evento de inicio (*PlaceOrder*) se envía a la cola de eventos de cliente para su procesamiento. El mediador del Cliente capta este evento de inicio y comienza a generar eventos de procesamiento basados en el flujo de la Figura 7. Observe que los múltiples eventos que se muestran en los pasos 2, 3 y 4 se realizan de manera simultánea y en serie entre los pasos. En otras palabras, el paso 3 (completar pedido) debe completarse y reconocerse antes de que se pueda notificar al cliente que el pedido está listo para enviarse en el paso 4 (pedido de envío).

Una vez que se ha recibido el evento de inicio, el mediador del Cliente genera un evento de procesamiento de la orden de creación y envía este mensaje a la cola de colocación de la orden (ver Figura 8). El procesador de eventos *OrderPlacement* acepta este evento y valida y crea el pedido, devolviendo al mediador un reconocimiento junto con el ID del pedido. En este punto, el mediador puede enviar ese ID de pedido al cliente, indicando que se realizó el pedido, o podría tener que continuar hasta que se completen todos los pasos (esto se basaría en reglas comerciales específicas sobre la colocación de pedidos).

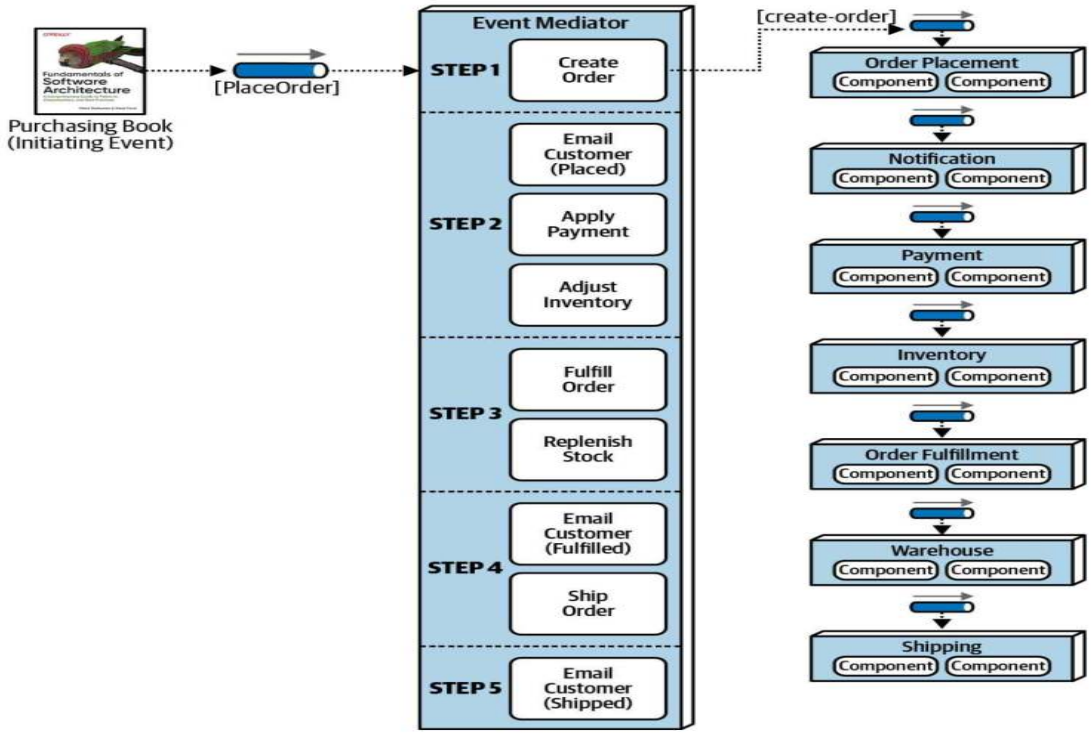

Figure 8: Paso 1 del ejemplo del mediador.

Ahora que el paso 1 está completo, el mediador pasa al paso 2 (consulte la Figura 9) y genera tres mensajes al mismo tiempo: enviar un correo electrónico al cliente, solicitar el pago y ajustar el inventario. Todos estos eventos de procesamiento se envían a sus respectivas colas. Los tres procesadores de eventos reciben estos mensajes, realizan sus respectivas tareas y notifican al mediador que el procesamiento se ha completado. Tenga en cuenta que el mediador debe esperar hasta recibir el reconocimiento de los tres procesos paralelos antes de pasar al paso 3. En este punto, si ocurre un error en uno de los procesadores de eventos paralelos, el mediador puede tomar medidas correctivas para solucionar el problema (esto se analiza más adelante en esta sección con más detalle).

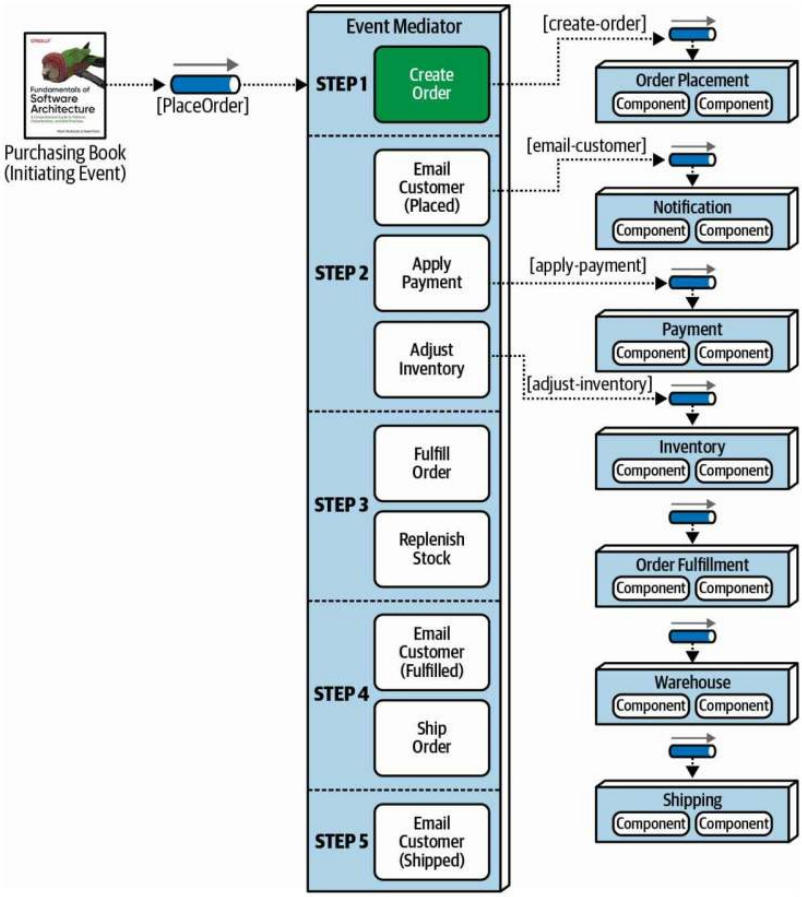

Figure 9: Paso 2 del ejemplo del mediador

Una vez que el mediador obtiene un reconocimiento exitoso de todos los procesadores de eventos en el paso 2, puede pasar al paso 3 para cumplir con la orden (ver Figura 10). Observe una vez más que ambos eventos (cumplimiento de pedido y pedido de stock) pueden ocurrir simultáneamente. Los procesadores de pedidos y eventos aceptan estos eventos, realizan su trabajo y devuelven un reconocimiento al mediador.

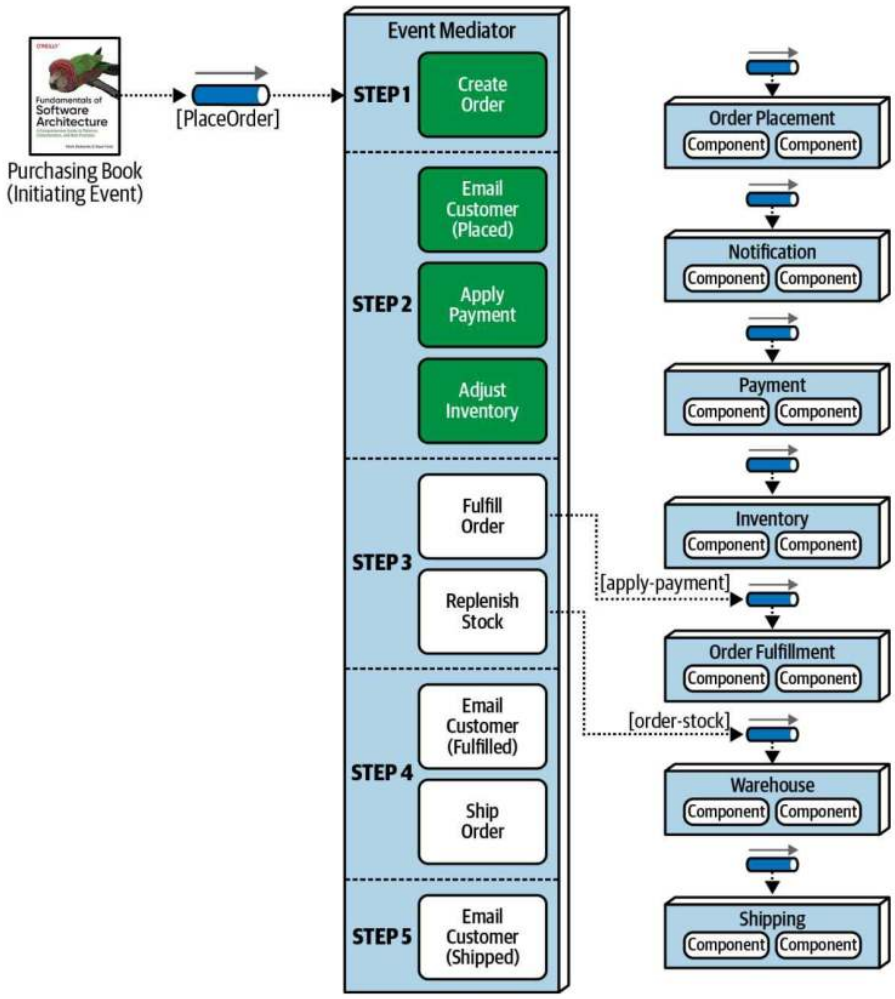

Figure 10: Paso 3 del ejemplo del mediador

Una vez que se completan estos eventos, el mediador pasa al paso 4 (ver Figura 11) para enviar el pedido. Este paso genera otro evento de procesamiento de cliente de correo electrónico con información específica sobre qué hacer (en este caso, notificar al cliente que el pedido está listo para ser enviado), así como un evento de orden de envío.

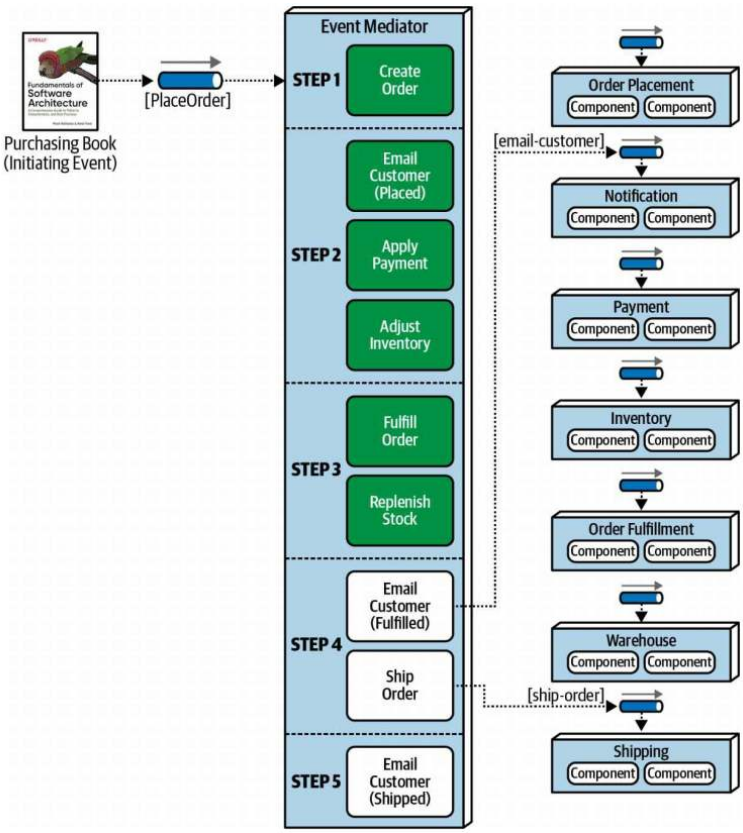

Figure 11: Paso 4 del ejemplo del mediador

Finalmente, el mediador pasa al paso 5 (ver Figura 12) y genera otro evento de cliente de correo electrónico contextual para notificar al cliente que el pedido ha sido enviado. En este punto, el flujo de trabajo finaliza y el mediador marca el flujo del evento de inicio como completo y elimina todos los estados asociados con el evento de inicio.

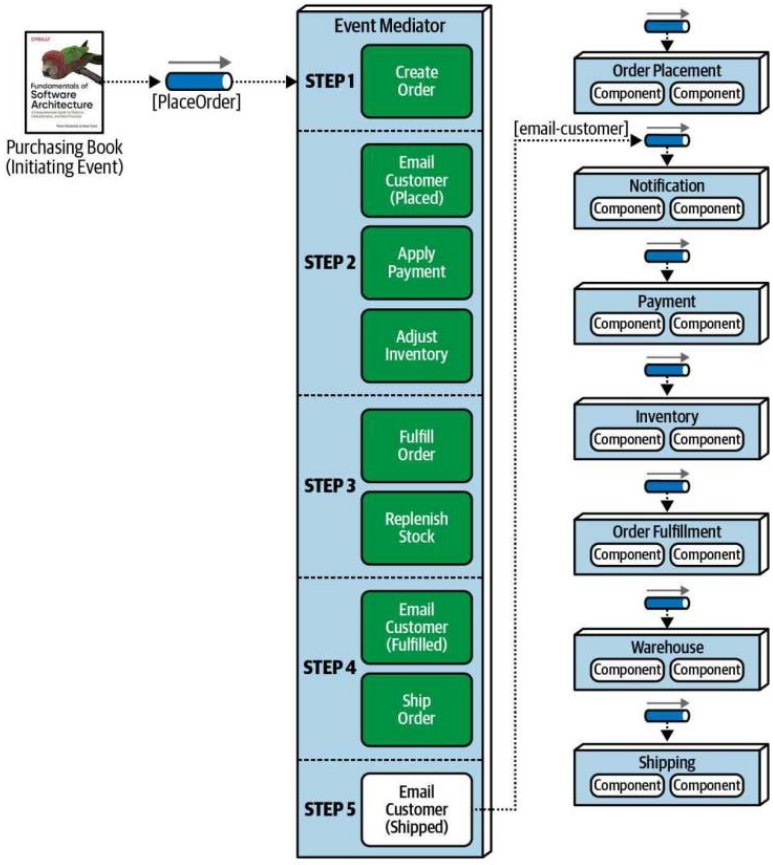

Figure 12: Paso 5 del ejemplo del mediador,

El componente mediador tiene conocimiento y control sobre el flujo de trabajo, algo que no tiene la topología del intermediario. Debido a que el mediador controla el flujo de trabajo, puede mantener el estado del evento y administrar el manejo de errores, la capacidad de recuperación y las capacidades de reinicio. Por ejemplo, suponga en el ejemplo anterior que el pago no se aplicó debido a que la tarjeta de crédito estaba vencida. En este caso, el mediador recibe esta condición de error, y sabiendo que el pedido no se puede cumplir (paso 3) hasta que se aplica el pago, detiene el flujo de trabajo y registra el estado de la solicitud en su propio almacén de datos persistente. Una vez que finalmente se aplica el pago, el flujo de trabajo se puede reiniciar desde donde se detuvo (en este caso, el comienzo del paso 3).

Otra diferencia inherente entre la topología del broker y del mediador es cómo los eventos de procesamiento difieren en términos de su significado y cómo se utilizan. En el ejemplo de topología de broker de la sección anterior, los eventos de procesamiento se publicaron como eventos que habían ocurrido en el sistema (como

pedido creado, pago aplicado y correo electrónico enviado). Los procesadores de eventos tomaron alguna acción y otros procesadores de eventos reaccionan a esa acción. Sin embargo, en la topología del mediador, las ocurrencias de procesamiento como colocar orden, enviar correo electrónico y completar orden son comandos (cosas que deben suceder) en contraposición a eventos (cosas que ya han sucedido). Además, en la topología del mediador, se debe procesar un comando, mientras que un evento se puede ignorar en la topología del intermediario.

Si bien la topología del mediador aborda los problemas asociados con la topología del intermediario, existen algunos *aspectos negativo*s asociados con la topología del mediador. En primer lugar, *es muy difícil modelar declarativamente* el procesamiento dinámico que ocurre dentro de un flujo de eventos complejo. Como resultado, muchos flujos de trabajo dentro del mediador solo manejan el procesamiento general, y se usa un modelo híbrido que combina las topologías del mediador y del corredor para abordar la naturaleza dinámica del procesamiento de eventos complejos (como condiciones de falta de existencias u otros errores no típicos). Además, aunque los procesadores de eventos pueden escalar fácilmente de la misma manera que la topología del broker, el *mediador también debe escalar*, algo que ocasionalmente produce un cuello de botella en el flujo general de procesamiento de eventos. Por último, los procesadores de eventos no están tan desacoplados en la topología del mediador como en la topología del intermediario, y *el rendimiento no es tan bueno* debido a que el mediador controla el procesamiento del evento. Estas compensaciones se resumen en la Tabla 2.

Table 2: Ventajas y desventajas de la topología del mediador

Ventajas	*Desventajas*
Control del flujo de trabajo	Mayor acoplamiento de los procesadores de eventos
Manejo del error	Baja escalabilidad
Recuperabilidad	Bajo desempeño
Reinicio de capacidades	Baja tolerancia a fallos
Mejor consistencia de datos	Modelamiento complejo de flujos de trabajo

La elección entre la topología del broker y del mediador se reduce esencialmente a una *compensación entre el control del flujo de trabajo y la capacidad de manejo de errores frente al alto rendimiento y la escalabilidad*. Aunque el rendimiento y la

escalabilidad siguen siendo buenos dentro de la topología del mediador, no son tan altos como con la topología del intermediario.

1.4. Capacidades asincrónicas

El estilo de arquitectura dirigido por eventos ofrece una característica única sobre otros estilos de arquitectura en el sentido de que se basa únicamente en la comunicación asincrónica tanto para el procesamiento de fuego y olvido (no se requiere respuesta) como para el procesamiento de solicitud/respuesta (respuesta requerida del consumidor de eventos). La comunicación asincrónica puede ser una técnica poderosa para aumentar la capacidad de respuesta general de un sistema.

Considere el ejemplo ilustrado en la Figura 13 donde un usuario publica un comentario en un sitio web para una revisión de producto en particular. Suponga que el servicio de comentarios en este ejemplo tarda 3000 milisegundos en publicar el comentario porque pasa por varios motores de análisis: un corrector de palabras incorrectas para comprobar si hay palabras inaceptables, un corrector gramatical para asegurarse de que las estructuras de las oraciones no digan algo abusivo y, finalmente, un verificador de contexto para asegurarse de que el comentario sea sobre un producto en particular y no solo una perorata política. Observe en la Figura 13 que la ruta superior utiliza una llamada RESTful sincrónica para publicar el comentario: 50 milisegundos de latencia para que el servicio reciba la publicación, 3000 milisegundos para publicar el comentario y 50 milisegundos de latencia de red para responder al usuario que el comentario fue publicado. Esto crea un tiempo de respuesta para el usuario de 3,100 milisegundos para publicar un comentario. Ahora mire la ruta inferior y observe que con el uso de mensajería asincrónica, el tiempo de respuesta desde la perspectiva del usuario final para publicar un comentario en el sitio web es de solo 25 milisegundos (en lugar de 3,100 milisegundos). Todavía se necesitan 3025 milisegundos para publicar el comentario (25 milisegundos para recibir el mensaje y 3000 milisegundos para publicar el comentario), pero desde la perspectiva del usuario final ya se ha hecho.

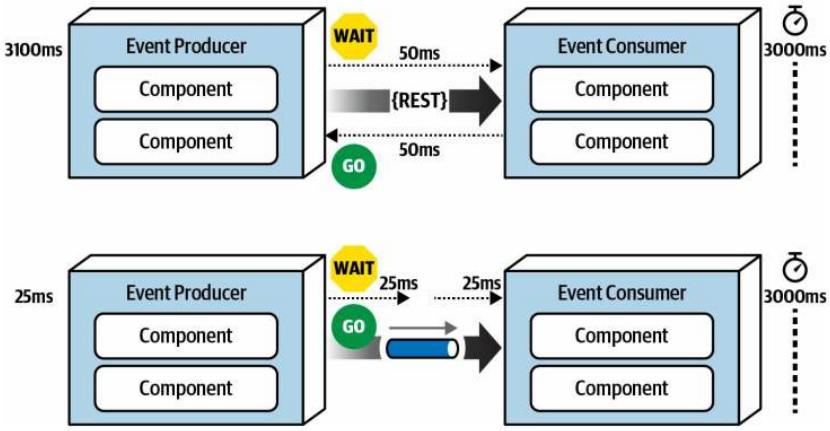

Figure 13: Comunicación sincrónica versus asincrónica.

Este es un buen ejemplo de la diferencia entre capacidad de respuesta y rendimiento. Cuando el usuario no necesita ninguna información de regreso (que no sea un reconocimiento o un mensaje de agradecimiento), ¿por qué hacer esperar al usuario? La capacidad de respuesta consiste en notificar al usuario que la acción ha sido aceptada y se procesará momentáneamente, mientras que el rendimiento se trata de hacer que el proceso de un extremo a otro sea más rápido. Observe que no se hizo nada para optimizar la forma en que el servicio de comentarios procesa el texto; en ambos casos es todavía tarda 3.000 milisegundos. Abordar el desempeño habría sido optimizar el servicio de comentarios para ejecutar todo el texto y el análisis gramatical motores en paralelo con el uso de almacenamiento en caché y otras técnicas similares. El ejemplo inferior de la Figura 13 aborda la capacidad de respuesta general del sistema, pero no el rendimiento del sistema.

La diferencia en el tiempo de respuesta entre los dos ejemplos en la Figura 13 de 3100 milisegundos a 25 milisegundos es asombrosa. Hay una salvedad. En la ruta síncrona que se muestra en la parte superior del diagrama, se garantiza al usuario final que el comentario se ha publicado. Sin embargo, en la ruta inferior solo está el reconocimiento de la publicación, con una promesa futura de que eventualmente se publicará el comentario. Desde la perspectiva del usuario final, se ha publicado el comentario. Pero, ¿qué pasa si el usuario ha escrito una mala palabra en el comentario? En este caso, el comentario sería rechazado, pero no hay forma de volver al usuario final. ¿O hay? En este ejemplo, asumiendo que el usuario está

registrado en el sitio web (cuál debería ser para publicar un comentario), se podría enviar un mensaje al usuario indicándole un problema con el comentario y algunas sugerencias sobre cómo repararlo. Este es un ejemplo simple. ¿Qué tal un ejemplo más complicado en el que la compra de algunas acciones se realiza de forma asincrónica (llamada negociación de acciones) y no hay forma de volver al usuario?

El principal problema de las comunicaciones asincrónicas es el manejo de errores. Si bien la capacidad de respuesta se mejora significativamente, es difícil abordar las condiciones de error, lo que aumenta la complejidad del sistema dirigido por eventos. La siguiente sección aborda este problema con un patrón de arquitectura reactiva llamado *patrón de eventos de flujo de trabajo* (workflow event pattern).

1.5. Manejo del error

El *patrón de eventos de flujo de trabajo* de la arquitectura reactiva es una forma de abordar los problemas asociados con el manejo de errores en un flujo de trabajo asincrónico. Este es un patrón de arquitectura reactiva que aborda tanto la *resiliencia*[1] como la *capacidad de respuesta*. En otras palabras, el sistema puede ser resistente en términos de manejo de errores sin afectar la capacidad de respuesta.

El patrón de eventos del flujo de trabajo aprovecha la delegación, la contención y la reparación mediante el uso de un *delegado de flujo de trabajo*, como se ilustra en la Figura 14. El productor de eventos pasa de forma asíncrona los datos a través de un canal de mensajes al consumidor de eventos. Si el evento del consumidor experimenta un error al procesar los datos, inmediatamente delegue ese error al procesador de flujo de trabajo y se traslada al siguiente mensaje en la cola de eventos. De esta manera, la capacidad de respuesta general no se ve afectada porque el siguiente mensaje se procesa inmediatamente. Si el consumidor del evento pasara el tiempo tratando de descubrir el error, entonces no está leyendo el siguiente mensaje en la cola, por lo tanto, afectando la capacidad de respuesta no solo del siguiente mensaje, sino también todos los demás mensajes esperando en la cola para procesar.

Una vez que el procesador de flujo de trabajo recibe un error, intenta averiguar qué está mal con el mensaje. Esto podría ser un error estático, determinista, o podría aprovechar algunos algoritmos de aprendizaje de la máquina para analizar el mensaje para ver alguna anomalía en los datos. De cualquier manera, el procesador de flujo de trabajo programáticamente (sin intervención humana) realiza cambios en los datos originales para intentar repararlo, y luego lo envía de vuelta a la cola de origen. El consumidor del evento ve este mensaje como nuevo y intenta procesarlo nuevamente, con suerte esta vez con algún éxito. Por supuesto, hay muchas veces en que el procesador de flujo de trabajo no puede determinar qué está mal con el

1 La palabra *resiliencia* se refiere a la capacidad de sobreponerse a momentos críticos

mensaje. En estos casos, el procesador de flujo de trabajo envía el mensaje a otra cola, que luego se recibe en lo que generalmente se llama "Dashboard", una aplicación que se parece a la Outlook de Microsoft o el correo de Apple. Este tablero de instrumentos generalmente reside en el escritorio de una persona de importancia, que luego analiza el mensaje, aplica las correcciones manuales, y la reenvía a la cola original (generalmente a través de una variable de encabezado de mensajes de respuesta).

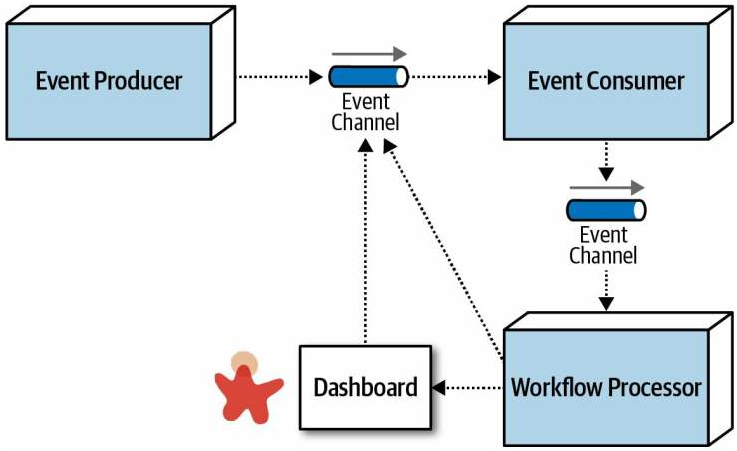

Figure 14: Patrón de evento de flujo de trabajo de la arquitectura reactiva..

Para ilustrar el patrón de eventos del flujo de trabajo, supongamos que un asesor comercial en una parte del país acepta órdenes comerciales (instrucciones sobre qué acciones comprar y cuántas acciones) en nombre de una gran empresa comercial en otra parte del país. El asesor gestiona en lote las órdenes comerciales (lo que generalmente se llama canasta) y envía asíncrona a aquellos a la gran empresa comercial que se colocará con un corredor para que se puedan comprar las acciones. Para simplificar el ejemplo, suponga que el contrato para las instrucciones comerciales debe cumplir con lo siguiente:

```
ACCOUNT(String),SIDE(String),SYMBOL(String),SHARES(Long)
```

Supongamos que la gran firma de comercio recibe la siguiente canasta de órdenes comerciales de Apple (AAPL) del asesor comercial:

```
12654A87FR4,BUY,AAPL,1254
```

```
87R54E3068U,BUY,AAPL,31226R4NB7609JJ,BUY,AAPL,5433
2WE35HF6DHF,BUY,AAPL,8756 SHARES
764980974R2,BUY,AAPL,1211
1533G658HD8,BUY,AAPL,2654
```

Observe la última instrucción comercial (`2WE35HF6DHF,BUY,AAPL,8756 SHARES`) tiene la plabra SHARES después de la cantidad de acciones para el comercio. Cuando estas órdenes comerciales asíncronas son procesadas por la gran firma de comercio sin ninguna capacidad de manejo de errores, se produce el siguiente error dentro del servicio de colocación comercial:

```
Exception in thread "main" java.lang.NumberFormatException:
For input string: "8756 SHARES"
at java.lang.NumberFormatException.forInputString
(NumberFormatException.java:65)
at java.lang.Long.parseLong(Long.java:589)
at java.lang.Long.<init>(Long.java:965)
at trading.TradePlacement.execute(TradePlacement.java:23)
at trading.TradePlacement.main(TradePlacement.java:29)
```

Cuando se produce esta excepción, no hay nada que pueda hacer el servicio de colocación comercial, porque esta era una solicitud asíncrona, excepto para registrar la condición de error. En otras palabras, no hay ningún usuario que responda de forma síncrona y solucione el error.

Aplicar el patrón de eventos del flujo de trabajo puede solucionar programáticamente este error. Debido a que la gran firma comercial no tiene control sobre el asesor comercial y los datos de orden comercial correspondientes que envía, debe reaccionar para solucionar el error en sí (como se ilustra en la Figura 15). Cuando se produce el mismo error (`2WE35HF6DHF,BUY,AAPL,8756 SHARES`), el servicio de colocación comercial delega inmediatamente el error a través de la mensajería asíncrona al servicio de error de colocación comercial para el manejo de errores, pasando con la información de error sobre la excepción:

```
Trade Placed: 12654A87FR4,BUY,AAPL,1254
Trade Placed: 87R54E3068U,BUY,AAPL,3122
Trade Placed: 6R4NB7609JJ,BUY,AAPL,5433
Error Placing Trade: "2WE35HF6DHF,BUY,AAPL,8756 SHARES"
Sending to trade error processor <-- delegate the error fixing and move on
Trade Placed: 764980974R2,BUY,AAPL,1211
…
```

El servicio de error de colocación comercial (actuando como del delegado del flujo de trabajo) recibe el error e inspecciona la excepción. Al ver que es un problema con la palabra SHARES en el número de acciones de acciones, el servicio de errores de colocación comercial se quita la palabra SHARES y reenvía el tráfico para el reprocesamiento:

```
Received Trade Order Error: 2WE35HF6DHF,BUY,AAPL,8756 SHARES
Trade fixed: 2WE35HF6DHF,BUY,AAPL,8756
Resubmitting Trade For Re-Processing
```

El comercio fijo es luego procesado con éxito por el servicio de colocación comercial:

```
...
trade placed: 1533G658HD8,BUY,AAPL,2654
trade placed: 2WE35HF6DHF,BUY,AAPL,8756 <-- this was the original trade in
error
```

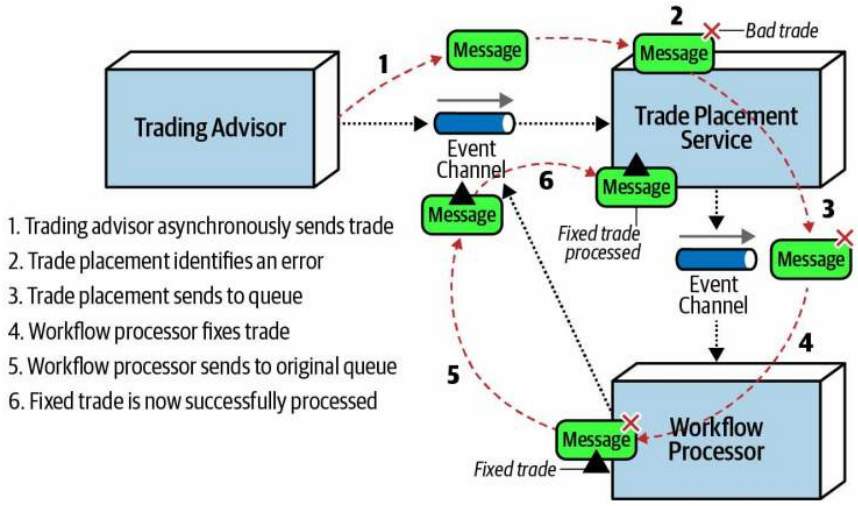

Figure 15: Manejo del error con el patrón workflow event

Una de las consecuencias del patrón de eventos del flujo de trabajo es que los mensajes en error se procesan fuera de secuencia cuando se vuelven a enviar. En

nuestro ejemplo de comercio, el orden de los mensajes importa, porque todos los operaciones dentro de un acopio dada deben procesarse en orden (por ejemplo, una venta para IBM debe ocurrir antes de comprar una compra para AAPL dentro de la misma cuenta de corretaje). Aunque no es imposible, es una tarea compleja para mantener el pedido de mensajes dentro de un contexto determinado (en este caso, el número de cuenta de corretaje). Una forma en que se puede abordar esto es por la colaboración del servicio de colocación comercial y almacenando el número de cuenta del error de comercio. Cualquier intercambio con ese mismo número de cuenta se almacenará en una cola temporal para el procesamiento posterior (en orden de FIFO). Una vez que el comercio originalmente en error sea fijo y procesado, el servicio de colocación comercial luego disminuye las operaciones restantes para esa misma cuenta y los procesa en orden.

1.6. Prevención de la pérdida de datos

La pérdida de datos es siempre una preocupación principal cuando se trata de comunicaciones asíncronas. Desafortunadamente, hay muchos lugares para que se produzca la pérdida de datos dentro de una arquitectura de eventos. Por la pérdida de datos nos referimos a *un mensaje que nunca llegará a su destino final*. Afortunadamente, hay técnicas básicas que se pueden aprovechar para prevenir la pérdida de datos al usar la mensajería asíncrona.

Para ilustrar los problemas asociados con la pérdida de datos dentro de la arquitectura de eventos, supongamos que el procesador de eventos A envía de manera asíncrona un mensaje a una cola. El procesador de eventos B acepta el mensaje e inserta los datos dentro del mensaje en una base de datos. Como se ilustra en la Figura 16, se pueden producir tres áreas de pérdida de datos dentro de este escenario típico:

1. El mensaje nunca llega a la cola del procesador de eventos A; O incluso si lo hace, el broker se cae antes de que el próximo procesador de eventos pueda recuperar el mensaje.
2. Procesador de eventos B encola el siguiente mensaje disponible y se bloquea antes de que pueda procesar el evento.
3. El procesador de eventos B no puede persistir el mensaje a la base de datos debido a algún error de datos.

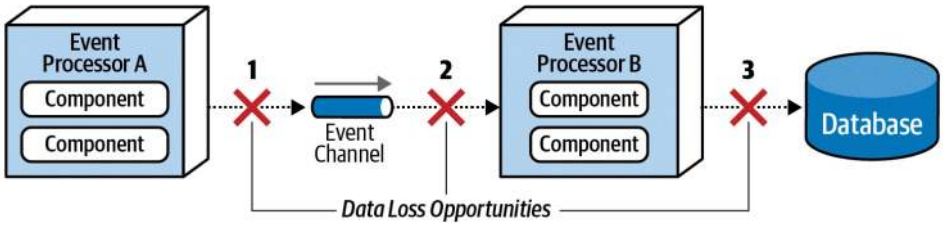

Figure 16: Dónde puede ocurrir la pérdida de datos dentro de una arquitectura dirigida por eventos

Cada una de estas áreas de pérdida de datos se puede mitigar mediante técnicas básicas de mensajería. El problema 1 (el mensaje nunca llega a la cola) se resuelve fácilmente aprovechando las colas de mensajes persistentes, junto con algo llamado envío síncrono. Las colas de mensajes persistentes admiten lo que se conoce como entrega garantizada. Cuando el intermediario de mensajes recibe el mensaje, no solo lo almacena en la memoria para una rápida recuperación, sino que también conserva el mensaje en algún tipo de almacén de datos físico (como un sistema de archivos o una base de datos). Si el broker de mensajes deja de funcionar, el mensaje se almacena físicamente en el disco para que cuando el intermediario de mensajes vuelva a funcionar, el mensaje esté disponible para su procesamiento. El envío síncrono realiza una espera de bloqueo en el productor de mensajes hasta que el intermediario reconoce que el mensaje persiste. Con estas dos técnicas básicas, no hay forma de perder un mensaje entre el productor de eventos y la cola porque el mensaje aún está en el productor de mensajes o persiste en la cola.

El problema 2 (el procesador de eventos B retira de la cola el siguiente mensaje disponible y se bloquea antes de que pueda procesar el evento) también se puede resolver utilizando una técnica básica de mensajería llamada modo de reconocimiento del cliente. De forma predeterminada, cuando un mensaje se quita de la cola, se elimina inmediatamente de la cola (algo llamado modo de reconocimiento automático). El modo de reconocimiento de cliente mantiene el mensaje en la cola y adjunta el ID de cliente al mensaje para que ningún otro consumidor pueda leer el mensaje. Con este modo, si el procesador de eventos B falla, el mensaje aún se conserva en la cola, evitando la pérdida de mensajes en esta parte del flujo de mensajes. El problema 3 (el procesador de eventos B no puede persistir el mensaje en la base de datos debido a algún error de datos) se

aborda aprovechando las transacciones ACID (atomicidad, consistencia, aislamiento, durabilidad) a través de una confirmación de la base de datos. Una vez que ocurre la confirmación de la base de datos, se garantiza que los datos se conservarán en la base de datos. Aprovechar algo llamado soporte del último participante (Last Participant Support - LPS) elimina el mensaje de la cola persistente al reconocer que el procesamiento se ha completado y que el mensaje persiste. Esto garantiza que el mensaje no se pierda durante el tránsito desde el Procesador de eventos A hasta la base de datos. Estas técnicas se ilustran en la Figura 17.

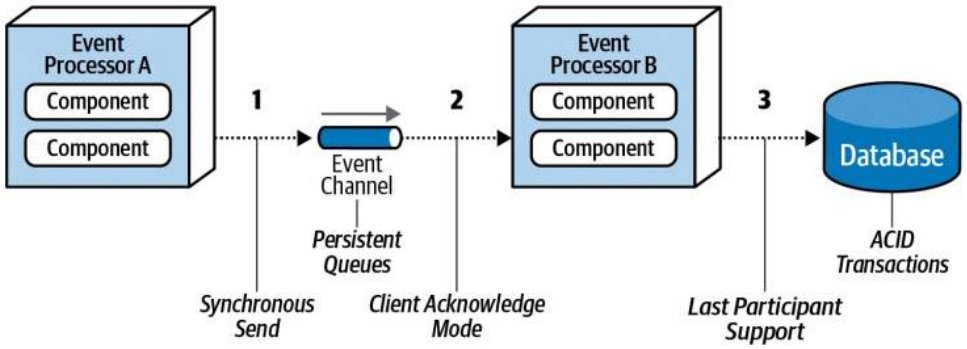

Figure 17: Previniendo la pérdida de datos en una arquitectura dirigida por eventos.

1.7. Capacidades Broadcast (transmisión)

Una de las otras características únicas de la arquitectura dirigida por eventos es la capacidad de transmitir eventos sin saber quién está recibiendo el mensaje y qué hacen con él. Esta técnica, que se ilustra en la Figura 18, muestra que cuando un productor publica un mensaje, varios suscriptores reciben el mismo mensaje.

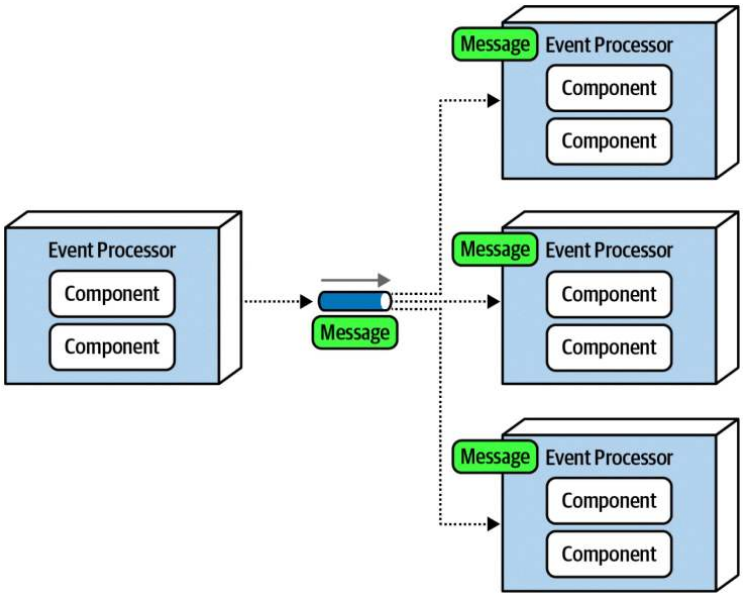

Figure 18: Transmitir eventos a otros procesadores de eventos

La transmisión es quizás el nivel más alto de desacoplamiento entre procesadores de eventos porque el productor del mensaje de transmisión generalmente no sabe qué procesadores de eventos recibirán el mensaje de transmisión y, lo que es más importante, qué harán con el mensaje. Las capacidades de transmisión son una parte esencial de los patrones para una eventual consistencia, procesamiento de eventos complejos y una serie de otras situaciones. Considere los cambios frecuentes en los precios de las acciones de los instrumentos que se negocian en el mercado de valores. Cada ticker (el precio actual de una acción en particular) puede influir en varias cosas. Sin embargo, el servicio que publica el último precio simplemente lo transmite sin saber cómo se utilizará esa información.

1.8. Solicitud respuesta

Hasta ahora, en este capítulo, hemos tratado las solicitudes asincrónicas que no necesitan una respuesta inmediata del consumidor de eventos. Pero, ¿qué pasa si se necesita una identificación de pedido al pedir un libro? ¿Qué sucede si se necesita un número de confirmación al reservar un vuelo? Estos son ejemplos de comunicación entre servicios o procesadores de eventos que requieren algún tipo de comunicación síncrona. En la arquitectura impulsada por eventos, la comunicación síncrona se logra *a través de mensajes de respuesta de solicitud* (a veces denominados comunicaciones *pseudosincrónicas*). Cada canal de eventos dentro de la mensajería de respuesta a solicitudes consta de dos colas: una cola de solicitudes y una cola de respuestas. La solicitud inicial de información se envía de forma asincrónica a la cola de solicitudes y luego se devuelve el control al productor del mensaje. A continuación, el productor de mensajes realiza una espera de bloqueo en la cola de respuestas, esperando la respuesta. El consumidor de mensajes recibe y procesa el mensaje y luego envía la respuesta a la cola de respuestas. El productor del evento luego recibe el mensaje con los datos de respuesta. Este flujo básico se ilustra en la Figura 19.

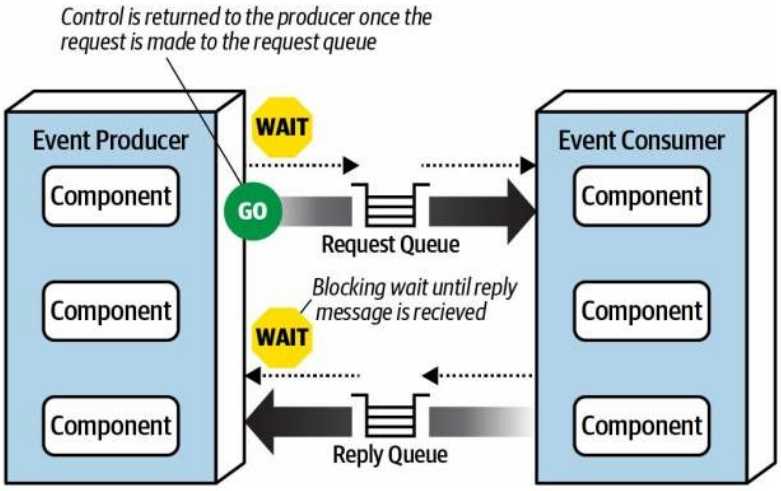

Figure 19: Procesamiento de mensaje de solicitud-respuesta.

Hay dos técnicas principales para implementar mensajes de respuesta a solicitudes. La primera (y más común) técnica es utilizar un *ID de correlación contenido* en el encabezado del mensaje. Un ID de correlación es un campo en el mensaje de respuesta que generalmente se establece en el ID de mensaje del mensaje de solicitud original. Esta técnica, como se ilustra en la Figura 20, funciona de la siguiente manera, con el ID de mensaje indicado con ID y el ID de correlación indicado con CID:

1. El productor de eventos envía un mensaje a la cola de solicitudes y registra el ID de mensaje único (en este caso el ID 124). Tenga en cuenta que el ID de correlación (CID) en este caso es nulo.
2. El productor de eventos ahora realiza una espera de bloqueo en la cola de respuestas con un filtro de mensajes (también llamado selector de mensajes), donde el ID de correlación en el encabezado del mensaje es igual al ID del mensaje original (en este caso 124). Observe que hay dos mensajes en la cola de respuesta: ID de mensaje 855 con ID de correlación 120 y ID de mensaje 856 con ID de correlación 122. Ninguno de estos mensajes se recogerá porque el ID de correlación no coincide con lo que el consumidor de eventos está buscando (CID 124).
3. El consumidor de eventos recibe el mensaje (ID 124) y procesa la solicitud.
4. El consumidor de eventos crea el mensaje de respuesta que contiene la respuesta y establece el ID de correlación (CID) en el encabezado del mensaje al ID del mensaje original (124).

5. El consumidor de eventos envía el nuevo mensaje (ID 857) a la cola de respuestas.
6. El productor de eventos recibe el mensaje porque el ID de correlación (124) coincide con el selector de mensajes del paso 2.

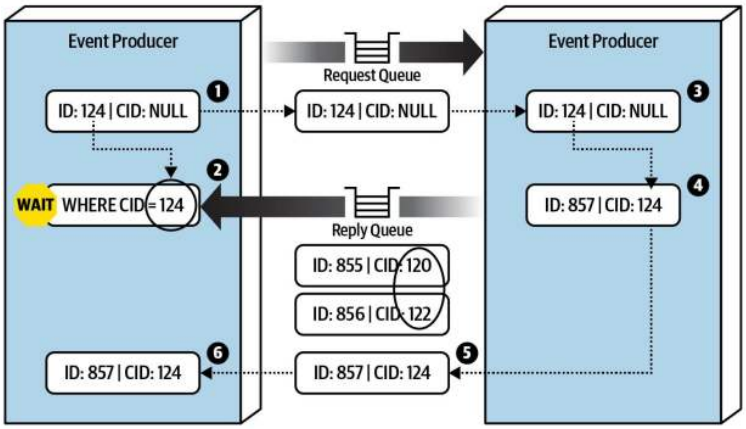

Figure 20: Procesamiento de mensaje de solicitud-respuesta utilizado ID de correlación.

La otra técnica utilizada para implementar la mensajería de respuesta a solicitud es usar una cola temporal para la cola de respuesta. Se dedica una cola temporal a la solicitud específica, que se crea cuando se realiza la solicitud y se elimina cuando finaliza la solicitud. Esta técnica, como se ilustra en la Figura 21, no requiere un ID de correlación porque la cola temporal es una cola dedicada que solo conoce el productor de eventos para la solicitud específica. La técnica de la cola temporal funciona de la siguiente manera:

1. El productor de eventos crea una cola temporal (o se crea una automáticamente, según el agente de mensajes) y envía un mensaje a la cola de solicitudes, pasando el nombre de la cola temporal en el encabezado de respuesta (o algún otro atributo personalizado acordado en el mensaje encabezamiento).
2. El productor de eventos realiza una espera de bloqueo en la cola de respuesta temporal. No se necesita ningún selector de mensajes porque

cualquier mensaje enviado a esta cola pertenece únicamente al productor de eventos que se envió originalmente al mensaje.

3. El consumidor de eventos recibe el mensaje, procesa la solicitud y envía un mensaje de respuesta a la cola de respuesta nombrada en el encabezado de respuesta.
4. El procesador de eventos recibe el mensaje y elimina la cola temporal.

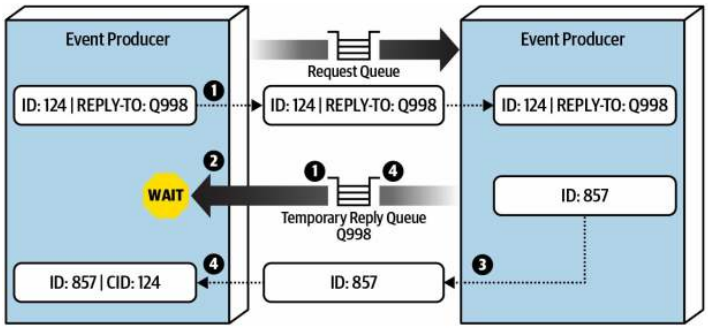

Figure 21: Procesamiento de solicitud-respuesta utilizando una cola temporal.

Si bien la técnica de la cola temporal es mucho más simple, el broker de mensajes debe crear una cola temporal para cada solicitud realizada y luego eliminarla inmediatamente después. Los grandes volúmenes de mensajería pueden ralentizar significativamente al intermediario de mensajes y afectar el rendimiento y la capacidad de respuesta generales. Por esta razón, generalmente recomendamos utilizar la técnica de identificación de correlación.

1.9. Elegir entre basado en solicitudes y basado en eventos

El modelo basado en solicitudes y el modelo basado en eventos son enfoques viables para diseñar sistemas de software. Sin embargo, elegir el modelo correcto es esencial para el éxito general del sistema. Recomendamos elegir el modelo basado en *solicitudes* para *solicitudes bien estructuradas y basadas en datos* (como la recuperación de datos del perfil del cliente) cuando se necesita certeza y control sobre el flujo de trabajo. Recomendamos elegir el modelo basado en *eventos* para eventos flexibles basados en acciones que requieren altos niveles de *capacidad de respuesta y escalabilidad*, *con un procesamiento de usuario complejo y dinámico*. Comprender las compensaciones con el modelo basado en eventos también ayuda a decidir cuál es el que mejor se ajusta. La Tabla 3 enumera las ventajas y desventajas del modelo basado en eventos de arquitectura dirigida por eventos.

Table 3: Trade-offs del modelo dirigido por eventos.

Ventajas	*Desventajas*
Mejor respuesta al contenido dinámico del usuario	Solo admite consistencia eventual
Mejor escalabilidad y elasticidad	Menos control sobre el flujo de procesamiento
Mejor agilidad y gestión de cambios	Menos certeza sobre el resultado del flujo de eventos
Mejor adaptabilidad y extensibilidad	Difícil de probar y depurar
Mejor capacidad de respuesta y rendimiento	
Mejor toma de decisiones en tiempo real	
Mejor reacción a la conciencia situacional	

1.10. Arquitecturas híbridas dirigidas por eventos

Si bien muchas aplicaciones aprovechan el estilo de arquitectura dirigida por eventos como la arquitectura general principal, en muchos casos la arquitectura dirigida por eventos se usa junto con otros estilos de arquitectura, formando lo que se conoce como arquitectura *híbrida*. Algunos estilos de arquitectura comunes que aprovechan la arquitectura dirigida por eventos como parte de otro estilo de arquitectura incluyen microservicios y arquitectura basada en el espacio. Otros híbridos que son posibles incluyen una arquitectura de *microkernel dirigida por eventos* y una *arquitectura de espacio (space-based) dirigida por eventos*.

Agregar arquitectura dirigida por eventos a cualquier estilo de arquitectura ayuda a eliminar los cuellos de botella, proporciona un punto de contrapresión en la copia de seguridad de las solicitudes de eventos y proporciona un nivel de respuesta del usuario que no se encuentra en otros estilos de arquitectura. Tanto los microservicios como la arquitectura basada en el espacio aprovechan la mensajería para las bombas de datos, enviando datos de forma asincrónica a otro procesador que, a su vez, actualiza los datos en una base de datos. Ambos también aprovechan la arquitectura dirigida por eventos para proporcionar un nivel de escalabilidad programática a los servicios en una arquitectura de microservicios y unidades de procesamiento en una arquitectura basada en el espacio cuando se utiliza la mensajería para la comunicación entre servicios.

1.11. Calificaciones de las características de la arquitectura dirigida por eventos

Una calificación de una estrella en la tabla de calificaciones de características en la Figura 22 significa que la característica de la arquitectura específica no está bien soportada en la arquitectura, mientras que una calificación de cinco estrellas significa que la característica de la arquitectura es una de las características más fuertes en el estilo de la arquitectura.

La arquitectura dirigida por eventos es principalmente una arquitectura técnicamente particionada en la que cualquier dominio particular se extiende a través de múltiples procesadores de eventos y se vincula a través de mediadores, colas y temas. Los cambios en un dominio en particular generalmente afectan a muchos procesadores de eventos, mediadores y otros artefactos de mensajería, por lo que la arquitectura dirigida por eventos *no tiene particiones de dominio*.

Architecture characteristic	Star rating
Partitioning type	Technical
Number of quanta	1 to many
Deployability	★★★
Elasticity	★★★
Evolutionary	★★★★★
Fault tolerance	★★★★★
Modularity	★★★★
Overall cost	★★★
Performance	★★★★★
Reliability	★★★
Scalability	★★★★★
Simplicity	★
Testability	★★

Figure 22: Calificaciones de las características de la arquitectura dirigida por eventos.

El número de cuantos dentro de la arquitectura dirigida por eventos puede variar de uno a muchos cuantos, lo que generalmente se basa en las interacciones de la base de datos dentro de cada procesador de eventos y el procesamiento de la respuesta a la solicitud. Aunque toda la comunicación en una arquitectura dirigida por eventos es asincrónica, si varios procesadores de eventos comparten una sola instancia de base de datos, todos estarían contenidos dentro del mismo cuanto arquitectónico. Lo mismo es cierto para el procesamiento de solicitudes de respuesta: aunque la comunicación sigue siendo asíncrona entre los procesadores de eventos, si se necesita una solicitud directamente del consumidor de eventos, vincula esos procesadores de eventos entre sí sincrónicamente; por tanto, pertenecen al mismo cuanto.

Para ilustrar este punto, considere el ejemplo en el que un procesador de eventos envía una solicitud a otro procesador de eventos para realizar un pedido. El primer procesador de eventos debe esperar a que continúe un ID de pedido del otro procesador de eventos. Si el segundo procesador de eventos que realiza el pedido y genera un ID de pedido está inactivo, el primer procesador de eventos no puede continuar. Por lo tanto, son parte de la misma arquitectura cuántica y comparten las mismas características arquitectónicas, aunque envían y reciben mensajes asincrónicos.

La arquitectura impulsada por eventos gana cinco estrellas por *rendimiento*, *escalabilidad* y *tolerancia a fallas*, las principales fortalezas de este estilo de arquitectura. El alto *rendimiento* se logra mediante comunicaciones asíncronas combinadas con un procesamiento altamente paralelo. La alta *escalabilidad* se logra mediante el equilibrio de carga programático de los procesadores de eventos (también llamados consumidores competidores). A medida que aumenta la carga de solicitudes, se pueden agregar procesadores de eventos adicionales mediante programación para manejar las solicitudes adicionales. La t*olerancia a fallas* se logra a través de procesadores de eventos altamente desacoplados y asíncronos que brindan consistencia eventual y procesamiento eventual de flujos de trabajo de eventos. Proporcionar la interfaz de usuario o un procesador de eventos que realiza una solicitud no necesita una respuesta inmediata, las promesas y los futuros se pueden aprovechar para procesar el evento en un momento posterior si no hay otros procesadores posteriores disponibles.

La *simplicidad general* y la tasa de capacidad de *prueba* son relativamente bajas con la arquitectura impulsada por eventos, principalmente debido a los flujos de

eventos dinámicos y no deterministas que normalmente se encuentran dentro de este estilo de arquitectura. Si bien los flujos deterministas dentro del modelo basado en solicitudes son relativamente fáciles de probar porque las rutas y los resultados generalmente se conocen, este no es el caso del modelo controlado por eventos. A veces, no se sabe cómo reaccionarán los procesadores de eventos a los eventos dinámicos y qué mensajes pueden producir. Estos "diagramas de árbol de eventos" pueden ser extremadamente complejos, generando cientos e incluso miles de escenarios, lo que hace que sea muy difícil de controlar y probar.

Finalmente, las arquitecturas impulsadas por eventos son altamente evolutivas, de ahí la calificación de cinco estrellas. Agregar nuevas funciones a través de procesadores de eventos nuevos o existentes es relativamente sencillo, particularmente en la topología del broker. Al proporcionar enlaces a través de mensajes publicados en la topología del agente, los datos ya están disponibles, por lo que no se requieren cambios en la infraestructura o en los procesadores de eventos existentes para agregar esa nueva funcionalidad.