Patrones de arquitectura

Rubén Gómez García

Version 1.1.2 2020-10-05

Contenidos

Capítulo 1. Introducción a los patrones de arquitectura

- El software posee un rol vital en la sociedad actual.
- Existe gran cantidad de tecnologías de la información que generan nuevas herramientas, técnicas y trucos
- Su evolución es rápida y su explotación cada vez es más sencilla.
- Los patrones de arquitectura pretenden, desde hace décadas, establecer la solución de problemas comunes de arquitectura
- Pretenden indicar la forma de implementar las soluciones en base a patrones conocidos.
- Entre ellos, podemos destacar los siguientes patrones de arquitectura:
  - OOA (Object-Oriented Architecture)
  - CBS (Component-based assembly Architecture)
  - DDD (Domain-Drive Design Architecture)
  - Client/Server Architecture
  - Multi-tier distributed computing Architecture
  - Layered/tiered Architecture
  - EDA (Event-Driven Architecture)
  - SOA (Service-Oriented Architecture)
  - MSA (Microservices Architecture)
  - SBA (Space-based Architecture)

1.1. Necesidades

- Con la interconexión entre sistemas, la cantidad de datos generada, recolectada, filtrada y explotada crece exponencialmente.
- Existen plataformas integradas para IoT, streaming, orientadas a velocidad o a grandes cantidades de datos
- Esto pone en peligro el paradigma clásico de las bases de datos.
- El middleware comienza a ser heterogéneo, y con múltiples sistemas y servicios integrados para trabajar en común.
- Esto empuja a crear formas de desarrollar y gestionar nuestro software de formas distintas
- Para solucionarlo, existen los patrones de arquitectura de software
- Su objetivos es:
  - Escalabilidad: Se evita el uso de aplicaciones monolíticas que se ejecutan en un solo sistema cuya escalabilidad es sensiblemene menor a un sistema distribuido.
  - Fiabilidad y disponibilidad: Se pretende evitar que un sistema tenga un SPOF (Single Point

Of Failure) que interrumpa el servicio, tanto si es por una caida del mismo como una actualización programada.

- Agilidad: Se evita código monolítico que crece y se vuelve más complejo, realizando actualizaciones cada 6, 12 meses o más.

1.2. Patrones de arquitectura

- Aquí podemos destacar los siguientes patrones de arquitectura

1.2.1. OOA (Object Oriented Architecture)

- Permite usar los objetos como bloques de construcción para aplicaciones.
- La estructura y el entorno de la aplicación se representa como objetos que están relacionados entre sí.
- Permite encapsulación de propiedades y tareas.
- Los objetos se comunican por medio de interfaces bien definidas.
- Las propiedades de OOA como herencia, polimorfismo, encapsulación y composición permiten generar código altamente modular y de poco acoplamento.
- También fomenta su reusabilidad.

1.2.2. CBD (Component-based Assembly Architecture)

- Las aplicaciones monolíticas se pueden dividir en múltiples componentes más pequeños e interactivos.
- Estos componentes son los bloques de construcción para el desarrollo de aplicaciones empresariales.
- Los componentes exponen una interfaz bien definida con otros componentes para su búsqueda y comunicación.
- Permite realizar un nivel de abstracción mayor que OOA.
- No se basan en protocolos de comunicación y en estados, sino que pretenden definir componentes reutilizables, reemplazables, extensibles e independientes.
- Los patrones de diseño como DI (Dependency Injection) o el **Service Locator** permiten el desacoplamiento y la reutilización.

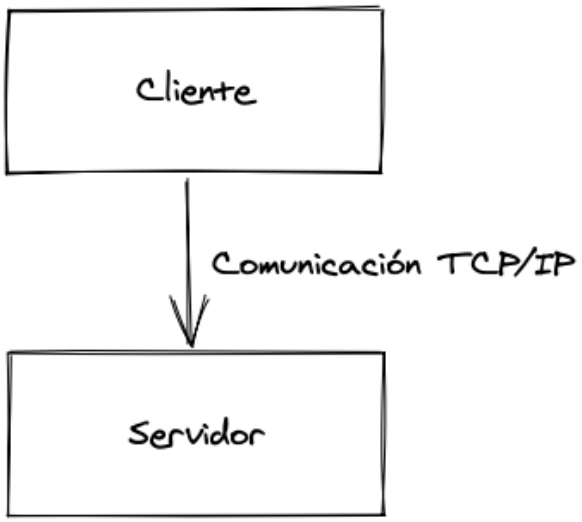
1.2.3. DDD (Domain Driven Design Architecture)

- Se trata de una aproximación al diseño de software basado en dominios de negocio.
- Sus elementos, su relación entre ellos y su entorno.
- Permite definir un modelo de dominio expresado en lenguaje de negocio.
- Los arquitectos deben conocer el dominio del negocio para su generación.
- El desarrollo se orienta por las necesidades de negocio.
- El lenguaje se focaliza a lenguaje de negocio sin apartados técnicos.

- El proceso DDD permite focalizar el objetivo
- También permite mejorar y refinar el lenguaje de dominio.
- Es muy interesante para casos que necesitan mejorar la comunicación entre el negocio y el equipo de desarrollo.
- Sobre todo si el dominio es muy complejo.

1.2.4. Client/Server Architecture

- El patron segrega el sistema en dos aplicaciones principales
- Los clientes mandan peticiones al servidor
- El servidor posee la lógica de la aplicación y una base de datos comunmente



- Los beneficios principales son:
  - Mayor seguridad: Los datos se almacenan en el servidor que posee mayor control de seguridad que las máquinas cliente.
  - Acceso a datos centralizado: Los accesos y actualizaciones de datos se administran de forma mas sencilla
  - Fácil de mantener: El servidor puede ser único o un cluster de múltiples máquinas. Puede ser un cluster activo/pasivo o un conjunto de subsistemas o microservicios. Cualquier reparación o mejora está centrada en la capa de servidor.

1.2.5. Multi/tier distributed computing architecture

- La arquitectura de dos capas no es flexible ni extensible
- Los componentes se despliegan en múltiples sistemas que pueden estar juntos o distribuidos geográficamente.
- Los componentes de aplicación se integran a través de mensajes o RPC (Remote Procedure Calls), RMI (Remote Method Invocations), CORBA (Common Object Request Broker Architecture), EJBs (Enterprise Java Beans), etc.
- Permite alta disponibilidad, escalabilidad, etc.

1.2.6. Layered/tiered architecture

- Se trata de una mejora del patrón de arquitectura cliente/servidor
- Es la más común de todas
- Una aplicación empresarial posee tres capas comunmente:
  - Capa de presentación/Interfaz de usuario
  - Capa de lógica de negocio
  - Capa de persitencia de datos
- Cada capa posee un componente de software asociado:
  - Presentación: Clientes pesados/Aplicaciones web
  - Negocio: Servidores de aplicaciones y servidores web
  - Persistencia: Base de datos
- La ventaja es la separación de responsabilidades, lo que permite:
  - Mejor mantenimiento
  - Mejor definición de pruebas
  - Asignar y separar roles específicos
  - Mejorar las actualizaciones y mejoras en cada capa de forma separada.

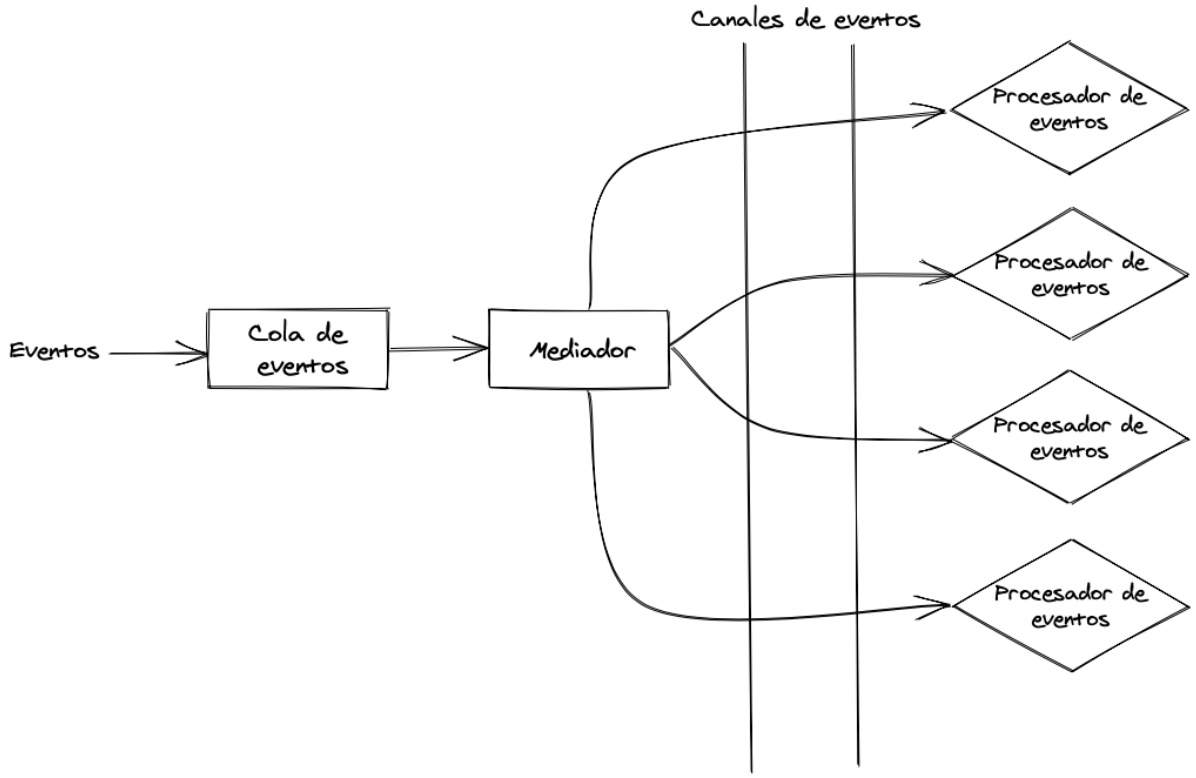
1.2.7. EDA (Event-driven Architecture)

- Normalmente los servidores responden a peticiones cliente
- Es el método petición/respuesta
- La comunicación es comúnmente síncrona
- Ambas capas deben estar disponibles para efectuar las operaciones pertinentes
- Los clientes deben esperar hasta recibir la respuesta
- Un evento es algo que pasa dentro o fuera del negocio.
- Puede ser cualquier cosa, como un error, un cambio de estado, o un umbral sobrepasado.
- Cada evento posee una cabecera y un cuerpo
- La cabecera posee los detalles del mismo como su id, el tipo de evento, nombre, creador, fecha, etc.
- EL cuerpo tiene toda la información relevante que sea necesario para trabajar con ese evento.
- Está basado en un modelo de comunicación asíncrono orientado a mensajes.
- Permite la propagación de información
- Permite definir sistemas altamente desacoplados.
- Las dependencias entre sistemas suelen desaparecer al aplicar este patrón.
- Existen motores de procesamiento de eventos, soluciones MoM (Message-oriented Middleware) como colas y brokers para recolectar y almacenar mensajes de eventos.

- Como fuentes de eventos, poseemos productores que publican notificaciones
- Como receptores, podemos escuchar todos los eventos o solo los que pasen un filtro concreto.
- El objetivo es la notificación de eventos que facilite el envio de información en tiempo real y permita la toma de decisiones basada en la información recibida.
- Permite aplicaciones empresariales altamente reactivas, como aquellas basadas en análisis de datos en tiempo real.
- Existen dos grantes topologías en esta arquitectura:

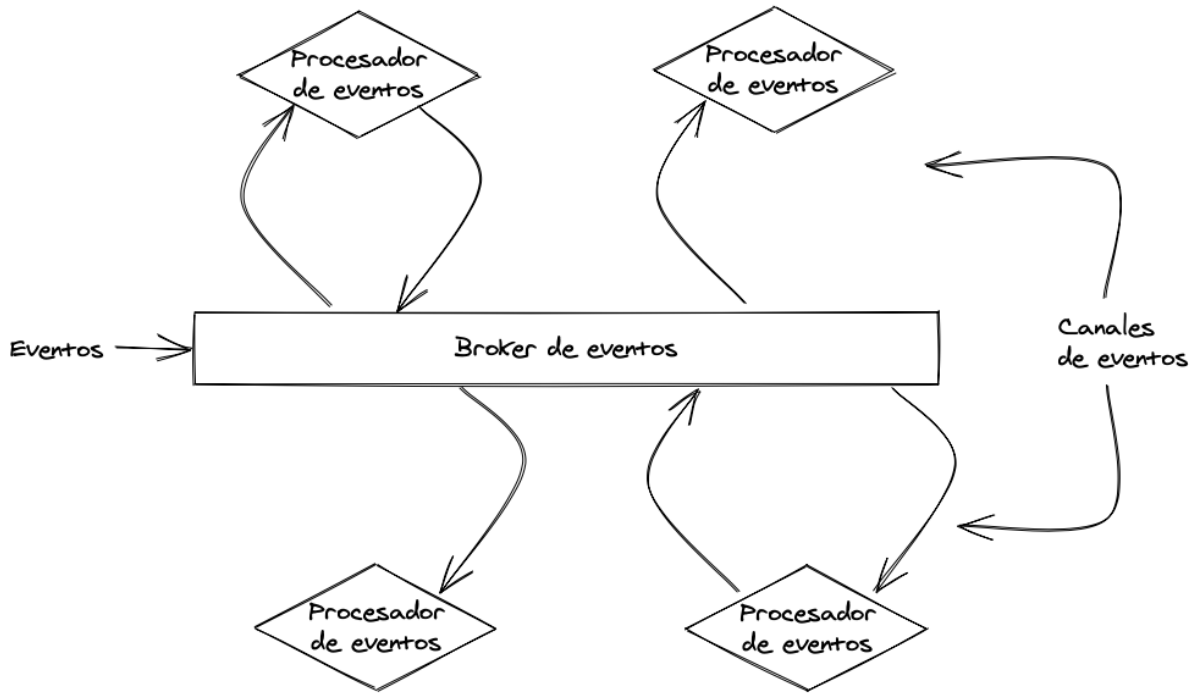
Mediator topology

- Es una sola cola de eventos y un mediador que redirige cada evento a su procesador de eventos
- Se suele pasar un evento a trave´s de un canal para filtrar y preprocesar los eventos antes de enviarlos.



Broker topology

- Esta topología no posee una cola de eventos.
- Los procesadores de eventos son los responsables de obtener los eventos, procesando y publicando otros eventos.
- Los procesadores de eventos actuan como brokers para encadenar eventos.
- Una vez que un evento es procesado por un procesador, otro evento es publicado y el procesador de eventos puede continuar.



- Podemos observar como algunos de los procesadores no generan nuevos eventos al broker.
- Los pasos de ciertas tareas se encadenan en forma de callbacks.
- Un ejemplo muy sencillo sería javascript.
- Los módulos que reaccionan a eventos como clicks o pulsaciones de teclas.
- El navegador orquesta las entradas y garantiza que el evento es visto solo por el módulo adecuado.

problemas

- El gran problema es que el patrón EDA carece de la atomicidad de las transacciones
- No hay una secuencia de ejecución de eventos.
- Los procesadores de eventos deben implementarse con la idea de que sean altamente distribuibles, desacoplados y asíncronos.
- Las pruebas en estos sistemas tampoco son sencillas debido a su naturaleza asíncrona.

1.2.8. SOA (Service Oriented Architecture)

- Con la llegada del paradigma de servicios, el desarrollo se orienta a que paquetes de software y librerías se desarrollan como una colección de servicios.
- Los servicios son autónomos, autodescritos, interoperables, autodescubribles, accesibles y reusables.
- Existen servicios de descubrimiento públicos y privados que permiten registrarlos.
- Los clientes pueden descubrir los servicios dinámicamente a través de estos servicios públicos
- Cada servicio posee dos partes:
  - Interfaz: El punto de contacto con el servicio. Se le considera el contrato de uso.
  - Implementación: Como está hecho, que queda oculto a los clientes.

1.2.9. SOI (service-inspired integration)

- Los buses de servicios (ESB) permiten la integración de servicios con otros recursos.
- Facilita la interconectividad, el enrutado, la corrección, el enriquecimiento, la gestión, etc.
- El ESB es el middleware de integración para cualquier entorno de servicios

1.2.10. MSA (Microservices Architecture)

- Se trata de un nuevo patrón de arquitectura que define, diseña, desarrolla, despliega y entrega aplicaciones de software distribuido empresarial.
- Es uno de los patrones más sencillos y rápidos de alcanzar que permite escalabilidad, alta disponibilidad y fiabilidad para cualquier aplicación de software

1.2.11. SBA (Space-based Architecture)

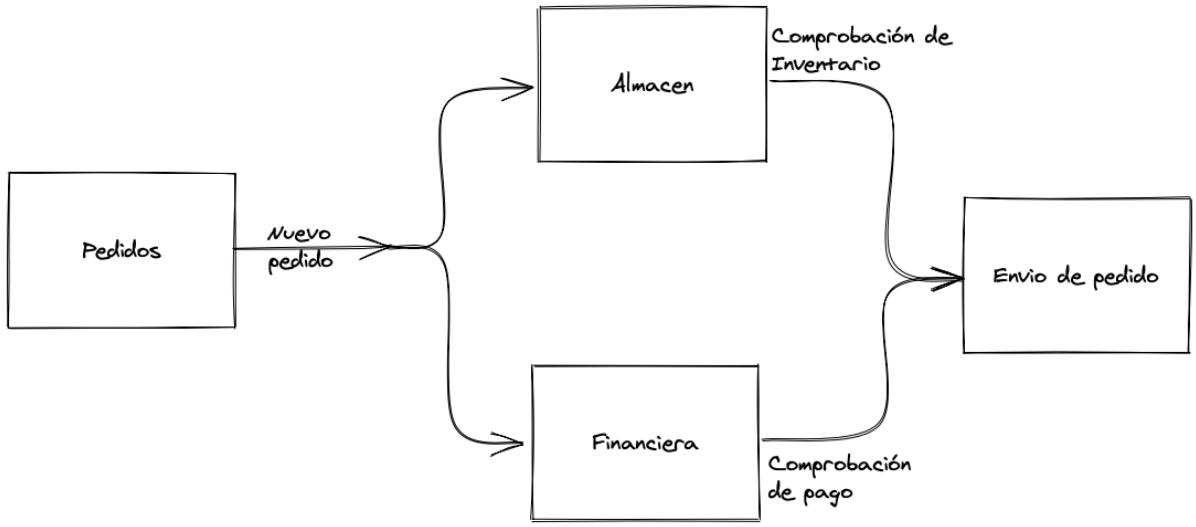
- Comunmente, las aplicaciones empresariales poseen un sistema de gestión de base de datos
- La aplicación puede escalar tanto como la carga que es capaz de gestionar la base de datos
- Sin embargo, en picos de uso, la base de datos no es capaz de escribir el **transaction log**, con lo cual, la aplicación es mas facil de fallar.
- Por eso se limita el número de transacciones que se pueden procesar concurrentemente.
- Con tecnología de caché y escalado de base de datos, podemos corregir esas deficiencias, pero sigue siendo una tarea compleja
- SBA se diseña para que los sistemas de software permitan grandes cargas de trabajo con gran carga de usuarios.
- La solución es dividir el almacenamiento en múltiples servidores.
- Los servidores envian la información a una gran cantidad de nodos.
- Esto permite corregir los errores de escalabilidad y concurrencia.
- La alta escalabilidad se alcanza eliminando la base de datos central y usando grids de datos en memoria
- Las unidades de proceso pueden iniciarse y pararse bajo demanda

1.3. Combinación de patrones de arquitectura

- Lo interesante realmente es la posibilidad de usar la combinación de distintos patrones de arquitectura para solventar los distintos problemas a los que nos enfrentamos.
- Esta ventaja permite fusionar algunos de ellos como el SBA y el EDA por ejemplo, dando servicio a una base de datos infinitamente escalable, y participando en un desarrollo orientado a eventos con brokers.

Capítulo 2. EDA (Event-driven Architecture)

- Un evento, de forma genérica, se refiere a un cambio de estado que es de interés



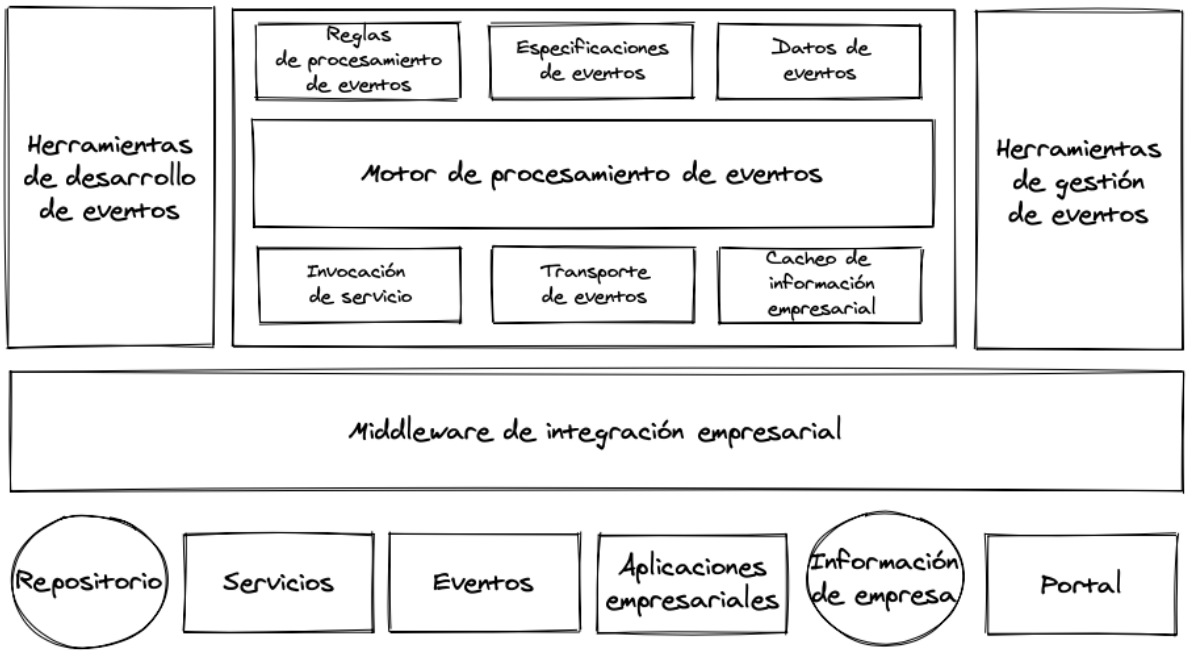
- En un sistema de gestión de ordenes de compra, el sistema debería notificar a todos los implicados la existencia de la nueva orden.
- La recepción de una orden es un evento.
- Este evento debe ser publicado a otros sitemas que puedan estar interesados en este evento.
- En este caso, podría ser el almacen para comprobar la existencia del pedido y la financiera para comprobar que el pago es correcto
- Cada uno de los elementos publica un evento a otro sistema que lo requiere para completar el proceso de venta.
- Sin embargo, no tiene porqué tener exclusivamente estos eventos, por ejemplo, en caso de inventario bajo, el almacen puede disparar otro evento para obtener mas objetos del inventario.
- Si la plataforma de pagos detecta un fallo en el pago, puede disparar un evento que mande un mensaje o un correo al cliente avisando del problema.
- En este caso, estamos usando el sistema de publicación/suscripcion.
- Podemos ver los siguientes elementos:
  - Evento: Operación en tiempo de ejecución, ejecutado por un componente de software para enviar información para su uso potencial por elementos no incluidos en la operación.
  - Publicador: Publicar es ejecutar un evento. Un elemento de software que ejecuta eventos.
  - Suscriptor: Un elemento de software que usa la información de los eventos
  - Contexto: En un diseño orientado a eventos, un contexto es una expresión booleana especificada por un suscriptor en tiempo de registro, evaluada en tiempo de ejecución. Esto permite ejecutar acciones solo si cumplen esa condición.

2.1. Características clave

- Las características principales son:
  - Comunicaciones multicast: Los publicadores poseen la capacidad de enviar eventos a múltiples sistemas suscritos. No se trata de una comunicación unicast.
  - Transmisiones en tiempo real: Los publicadores ejecutan los eventos en tiempo real a los suscriptores. No usa el procesamiento batch.
  - Comunicaciones asíncronas: El publicador no espera por el proceso receptor a que procese un evento antes de enviar otro.
  - Comunicaciones de grano fino: Los publicadores continuan publicando eventos de grano fino en vez de esperar por un evento agregado único.
  - Ontología: Los sistemas EDA siempre poseen una técnica para clasificación y agrupación de eventos basado en características comunes. Esto permite a un suscriptor suscribirse a un tipo específico de eventos o a un grupo completo.

2.2. Componentes del patrón EDA

- Los compontes que los conforman son los siguientes
  - Especificaciones de evento
  - Procesamiento de evento
  - Herramientas de evento
  - Integración empresarial
  - Orígenes y destinos



- Los componentes de EDA poseen una fuerte arquitectura de metadatos.
- Los componentes del nucleo de metadatos de eventos son los siguientes:

2.2.1. Especificaciones de eventos

- Estas especificaciones deben estar disponibles para los generadores de eventos, los motores de eventos y los transformadores de eventos.
- No existe un estándar para definición y procesamiento de eventos por el momento.

2.2.2. Procesamiento de eventos

- Es una tecnica de análisis de streams de datos con el objetivo de obtener una conclusión sobre él.
- Por ejemplo, el sistema de predicción del clima tiene como objetivo detectar tormentas buscando patrones.
- Todos estos datos forman los datos del evento, y deben ser procesados por un motor de eventos para obtener una conclusión.
- Los elementos esenciales son:
  - El motor de eventos
  - Los datos de evento

2.2.3. Herramientas de desarrollo de eventos

- Provee de funciones para definir especificaciones de eventos
- Define reglas de procesamiento de eventos
- Gestiona las suscripciones de eventos
- Posee también gestión de monitorización del procesamiento de eventos y flujos de eventos

2.2.4. Integración empresarial

- Algunos de los servicios necesarios para la integración son:
  - Procesamiento de eventos
  - Canal de transporte del evento
  - Invocación del servicio
  - Publicación y suscripción
  - Acceso a la información de empresa

2.2.5. Origenes y destinos

- Los orígenes se refiere a componentes empresariales que generan eventos
- Sistemas, servicios, agentes automatizados o usuarios responsables de crear eventos.
- Los destinos se refieren a componentes que realizan una acción basada en la ocurrencia de eventos basados en salidas de los mismos.
- La topología se gobierna con distintos parámetros como:

    - Flujos de eventos
    - Volumen de ocurrencia de eventos
    - Localizacion de orígenes y destinos, etc.

2.3. Capas de Flujos de eventos

- Las cuatro capas de flujos de eventos son:

2.3.1. Generadores de eventos

- Los orígenes de los eventos se les llama generadores de eventos.
- Puede ser una aplicación, un servicio, un sensor, etc.
- Un evento generado es evaluado por un filtro de eventos que comprueba si cumple la condición.
- Los eventos que la cumplen, se convierten a un formato compatible antes de enviarse a un canal de eventos.

2.3.2. Canales de eventos

- Actua como medio transmisor y como base de EDA.
- Recibe eventos estándar formateados de un generador de eventos y lo manda a otros generadores de eventos, a motores de eventos o a suscriptores

2.3.3. Procesamiento de eventos

- Una vez recibido el evento, es procesado y evaluado basado en reglas almacenadas en el motor de procesamiento de eventos.
- Según el resultado, se ejecuta una tarea distinta.
- El resultado puede ser notificaciones, inicio de procesos de negocio, etc.

2.3.4. Actividad interna

- Cualquier evento puede disparar una serie de actividades interna que puede ser la respuesta al evento.
- El evento puede ser una notificación push por el motor de eventos, o un pull de notificaciones por parte de los suscriptores.

2.4. Consideraciones de diseño

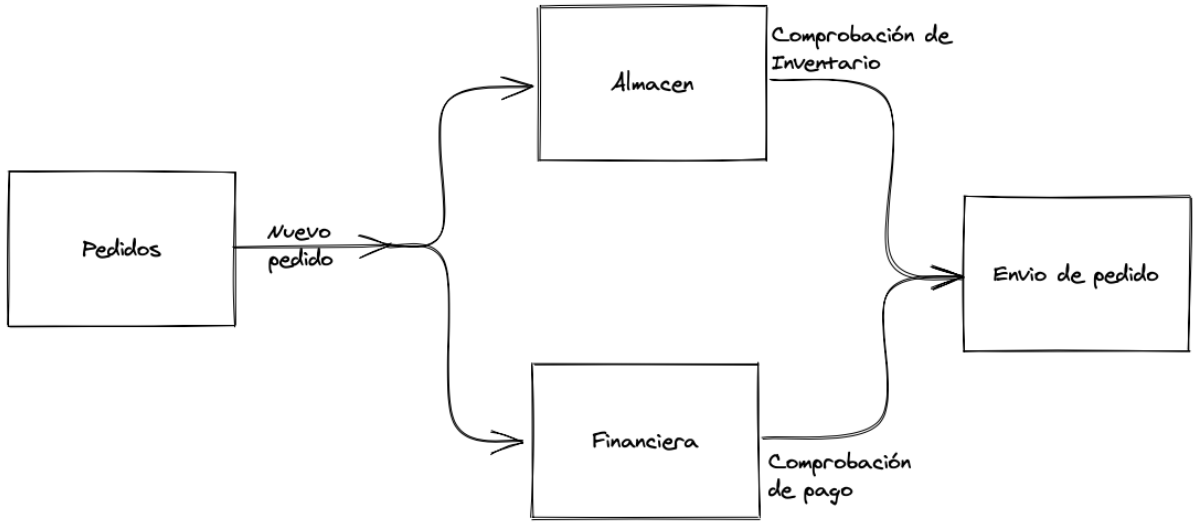
- Hay que tener en cuenta las consideraciones de diseño de patrones EDA.
- Las principales son:
    - Agilidad:
        - Ser capaz de realizar cambios rápidamente en el entorno.
        - Los componentes están poco acoplados, lo que implica que los cambios no afectan a

otros componentes del sistema.
        - EDA es ideal para sistemas con cambios continuos sin caida de servicio.
    - Despliegue sencillo:
        - Los componentes, al estar poco acoplados, permiten despliegues sencillos.
        - La mejor topología para estos casos es la de event broker, en contraposición a la de mediador.
        - El acoplamiento es mayor entre el mediador de eventos y el procesador de los mismos.
    - Testing:
        - Los test unitarios del patrón EDA es dificil, ya que requiere de clientes y herramientas de generación de eventos específicos.
    - Rendimiento:
        - EDA realiza operaciones asíncronas en paralelo.
        - Le otorga un gran rendimiento independientemente del tiempo de retraso producido por encolar y desencolar mensajes.
    - Escalabilidad:
        - EDA ofrece un alto nivel de escalabilidad por el desacoplamiento de sus componentes
    - Fácil desarrollo:
        - Sencillo de desarrollar debido a la naturaleza asíncrona del patrón.

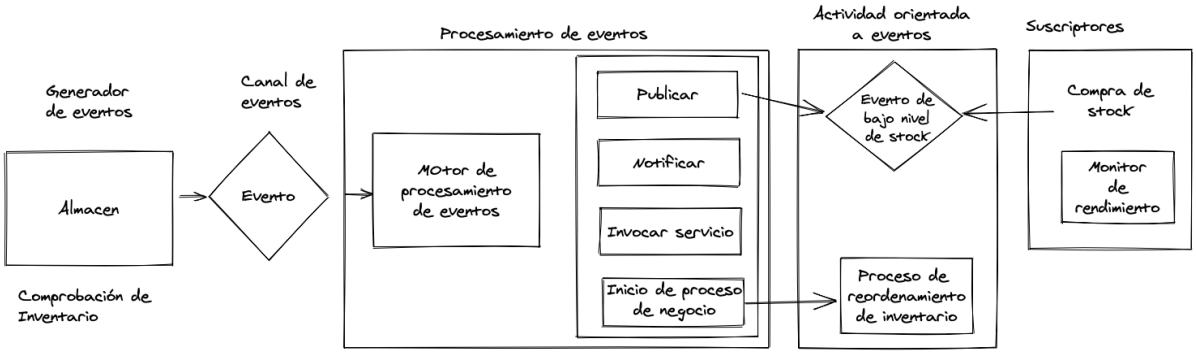
2.5. Variantes de implementación de los patrones EDA

2.5.1. Patrones de procesamiento de eventos simples

- Se usan para medir eventos relacionados a cambios medibles en ciertas condiciones.
- Se usan para procesamiento de flujos en tiempo real
- No se persigue el tiempo de retardo ni los costes relacionados con el negocio
- Orientado a detección de cambios de presión y temperatura o similares.
- Si nos fijamos en el primer diagrama, podría ser el evento de pocas unidades de inventario.



- Si nos centramos ahora en el evento de nivel de stock, observamos como actuan los distintos roles



2.5.2. CEP (Patrones de procesamiento de eventos complejos)

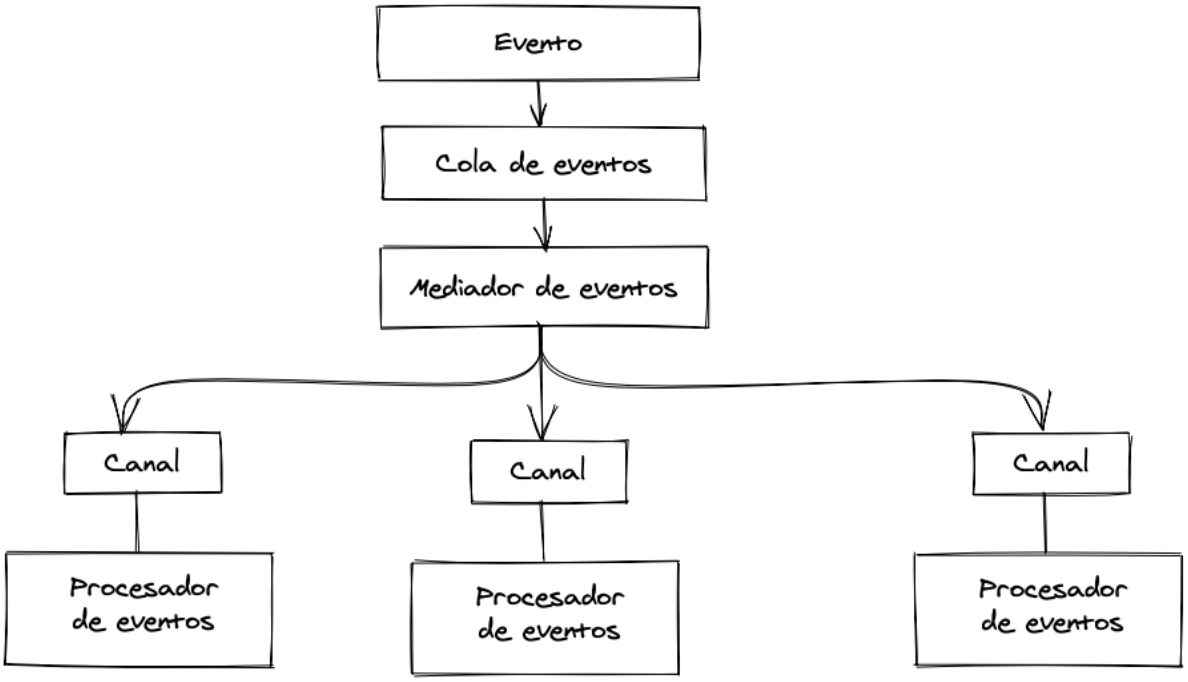
- Permite responder a anomalías en el negocio.
- Permite combinar eventos simples con otros eventos que se deban evaluar en un periodo de tiempo largo.
- La correlación ocurre en el espacio y en el tiempo.
- Por ello, se necesitan los siguientes componentes:
  - Interpretes de eventos
  - Definición de patrones de eventos
  - Coincidencias de patrones de eventos
  - Técnicas de correlación de eventos

2.6. Tipos de patrones orientados a eventos

- Existen dos tipos de topologías:
  - Mediador
  - Broker

2.6.1. Patrón de topología de mediador de eventos

- Se usa para definir o diseñar sistemas o procesos que necesitan algún nivel de orquestación o coordinación para procesar el evento.
- Por ejemplo, un procesamiento de múltiples pasos, ya que requieren de un orden y hay que comprobar aquellos que se generan en serie o en paralelo.
- Existen cuatro componentes principales
  - Cola de eventos
  - Mediador de eventos
  - Canales de evento
  - Procesadores de evento

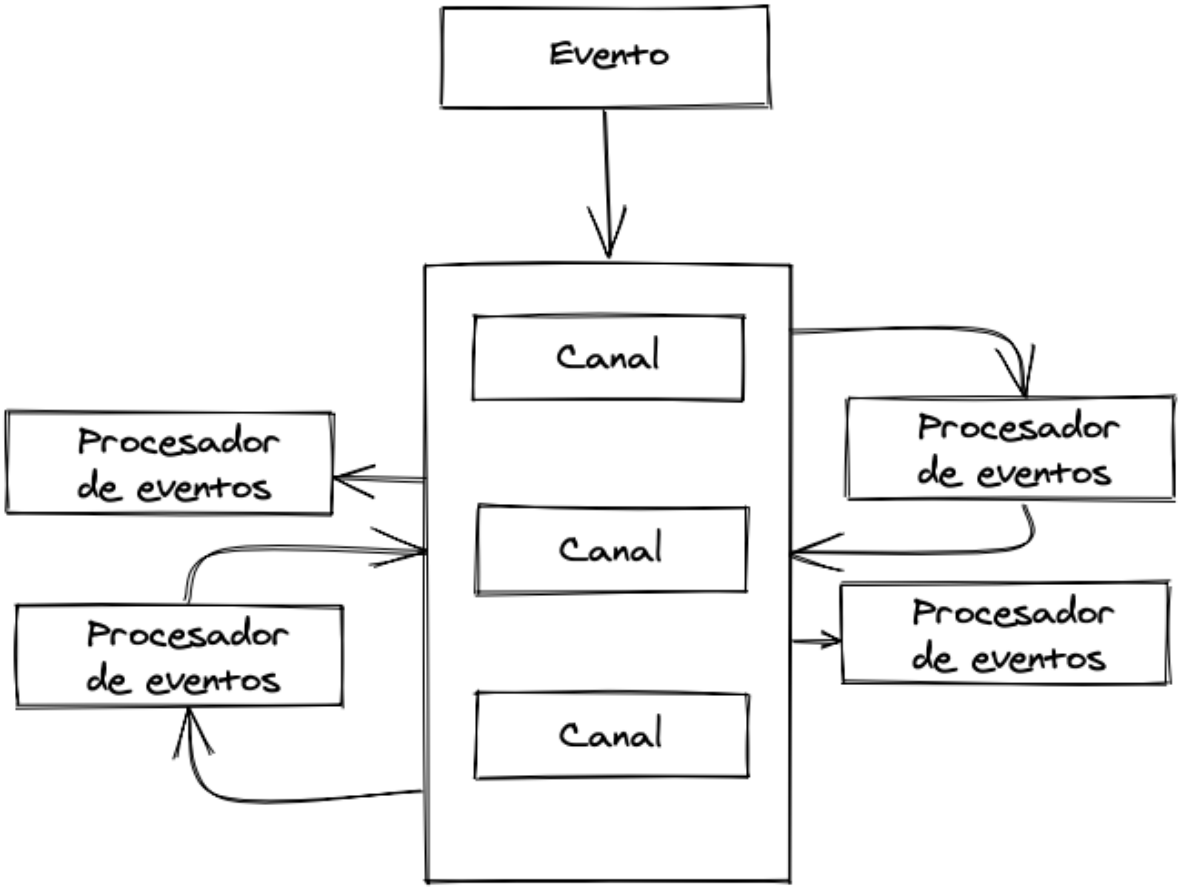


- El cliente envia un evento que es recibido por la cola de eventos
- La cola lo transfiere al mediador
- EL mediador recibe el evento y lo orquesta.
- En este paso envia una serie de eventos asíncronos a varios canales que se encargan de ejecutar los pasos necesarios.
- El procesador de eventos recoge el evento del canal y aplica la lógica de negocio asociada.
- La cola de eventos puede ser implementada de distintas formas:
  - Cola de mensajería
  - Componente Web Service
  - etc.
- Existen dos tipos de eventos disponibles en este patrón
  - Evento inicial: El evento original del mediador

    - Evento de procesamiento: Eventos generados por el mediador y enviados a los componentes de procesamiento.
- Los canales de eventos pueden aparecer en forma de colas o topics.
- La lógica de la aplicación está altamente acoplada ya que están asociadas a tareas específicas del sistema

2.6.2. Patrón de topología de broker de eventos

- Se usa cuando el flujo de eventos es relativamente simple
- No es necesario un orquestador central
- Los componentes principales son:
    - Broker
    - Procesador de eventos



- El componente de broker contiene todos los canales de eventos y pueden estar designados de forma centralizada o federada
- Podemos observar la ausencia del mediador
- El procesador de eventos posee el rol de procesar y publicar cada evento.
- EL broker puede contener todos los canales y pueden gestionar colas, topics o combinaciones de ambas.