**Identificador**

**Nombre Decisión**

**Contexto**

**Requisitos**

**Descripción**

**Argumentación**

**Alternativas**

**Pros y Cons**

{ADD-001}

{ADD-001.1}

* Nombre Decisión: [Estilo por Eventos]
* Estatus: [Propuesto]
* Fecha: [2022-11-08]

**Requisitos**

\*[RF-02] \*[RF-03] \*[RF-09]

**Descripción**

{Está compuesta por productores y consumidores de eventos. Una vez que se detecta un evento, este se transmite a una plataforma donde se procesa de manera asíncrona para procesar los eventos. La plataforma de procesamiento ejecutará la respuesta adecuada para el evento y enviará la actividad a los consumidores correspondientes.}

**Alternativas**

* {Estilo por Capas}
* {Estilo Pipe and Filter}

**Argumentación**

{Hemos elegido el Estilo por Eventos porque según como está escrito nuestro problema es el que más cuadra. Ya que, tenemos el productor de eventos, en este caso serían los sensores. Los sensores envían datos al centro de notificaciones (Cockpit), funcionando como gestor de eventos. Finalmente, el Cockpit le envía órdenes a las diferentes máquinas y trabajadores.}

**Consecuencias Positivas**

* **Escalabilidad**: La escalabilidad es uno de los puntos más fuertes de esta arquitectura, pues permite que cada consumidor (Event Consummer) puede escalar de forma independiente y reduce al máximo el acoplamiento entre los componentes.
* **Despliegue**: Debido al bajo acoplamiento entre los componentes, es posible el despliegue sin preocuparse por dependencias o precondiciones, al final, los componentes solamente se suscriben para recibir eventos y reaccionar ante ellos.
* **Performance**: Esta es que ventaja cuestionable, ya que al igual que en REST o SOA, EDA necesita pasar por una serie de pasos para completar una tarea, agregando retrasos en cada paso, desde colocar el Evento por parte del productor, esperar a que el consumidor lo tome, y generar nuevos Eventos, sin embargo, la naturaleza Asíncrona de EDA hace que esta desventaja se supere mediante el procesamiento en paralelo.
* **Flexibilidad**: EDA permite responder rápidamente a un entorno cambiante, debido a que cada componente procesador de eventos tiene una sola responsabilidad y está completamente desacoplado de los demás, de esta forma, si ocurre un cambio, se puede aislar en un solo componente sin afectar al resto, además, si un nuevo requerimiento es requerido, solo es necesario regresar un nuevo tipo de procesador de eventos que escuche un determinado tipo de evento.

**Consecuencias Negativas**

* **Testabilidad**: Una arquitectura distribuida y asíncrona agrega cierta complejidad a las pruebas, pues no es posible generar un evento y esperar un resultado para validar el resultado, en su lugar, es necesario crear pruebas más sofisticadas que creen los eventos y esperen la finalización del procesamiento del evento inicial más toda la serie de eventos que se podrían generar como consecuencia del evento inicial para validar el resultado final.
* **Desarrollo**: Codificar soluciones asíncronas es complicado, pero aún más, es la necesidad de crear manejadores de errores más avanzados que permitan recuperarse en una arquitectura asíncrona, donde la falla en un procesador no significa la falla en el resto, lo que puede ocasionar la inconsistencia entre los sistemas.

{ADD-001.3}

* Nombre Decisión: [Estilo Pipe and Filter]
* Estatus: [Propuesto]
* Fecha: [2022-11-09]

**Requisitos**

\*[RF-02] \*[RF-03] \*[RF-09]

**Descripción**

{Provee una estructura para los sistemas que procesan un flujo de datos. Cada paso de procesamiento se encapsula en un componente de filtro. Los datos pasan a través de tuberías entre filtros adyacentes. Recombinar filtros permite construir familias de filtros relacionados}

**Argumentación**

{Hemos pensado que este estilo podría funcionar, ya que los diferentes sistemas de nuestro software se pueden comportar como filtros, como pueden ser los sensores o el centro de notificaciones (Cockpit). Ya que, los filtros leen un flujo de datos de su entrada y generan flujos de datos a sus salidas. Transmitiéndose así el flujo de datos entre los diferentes filtros.}

**Consecuencias Positivas**

* Descomposición del problema en pasos independientes.
* Soporta la reutilización de los filtros.
* Sistemas fáciles de mantener y mejorar.
* Soportan la ejecución concurrente de filtros.
* Escalables.

**Consecuencias Negativas**

* No adecuados para procesamiento interactivo.
* Problemas de rendimiento ya que los datos se transmiten en forma completa entre filtros.
* Acordar cuál es el formato de los datos.
* Tiene que ser genérico si las componentes son reusadas.
* Dificulta el manejo de errores.

{ADD-002}

* Nombre Decisión: [Factory Method]
* Estatus: [Propuesto]
* Fecha: [2022-11-08]

**Requisitos**

\*[RF-05]

**Descripción**

{Es un patrón de diseño creacional que proporciona una interfaz para crear objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.}

**Alternativas**

* {Abstract Factory}

**Argumentación**

{}

**Consecuencias Positivas**

**Consecuencias Negativas**

{ADD-003}

* Nombre Decisión: [Observer]
* Estatus: [Propuesto]
* Fecha: [2022-11-08]

**Requisitos**

\*[RF-06] \*[RF-07]

**Descripción**

{Es un patrón de diseño de comportamiento que te permite definir un mecanismo de suscripción para notificar a varios objetos sobre cualquier evento que le suceda al objeto que están observando.}

**Alternativas**

* {Publish-Suscribe}

**Argumentación**

{}

**Consecuencias Positivas**

**Consecuencias Negativas**

{ADD-004}

* Nombre Decisión: [Estilo por Capas]
* Estatus: [Propuesto]
* Fecha: [2022-11-08]

**Requisitos**

\*[RF-06] \*[RF-07]

**Descripción**

{Todas las capas se colocan de forma horizontal, de tal forma que cada capa solo puede comunicarse con la capa que está inmediatamente por debajo, por lo que, si una capa quiere comunicarse con otras que están mucho más abajo, tendrán que hacerlo mediante la capa que está inmediatamente por debajo}

**Alternativas**

* {Estilo Pipe and Filter}
* **Chain of Responsibility**

**Argumentación**

{}

**Consecuencias Positivas**

**Consecuencias Negativas**

ADD-005

Elegir entre 2 algoritmos:

**State**

**Strategy**

Estilo Principal.md

* Estatus: [propuesto]
* : [ASS Grupo 1]
* Fecha: [2022-11-08]

**Contexto del problema**

**Requisitos**

[RF-02], [RF-09], [RF-03]

**Opciones Consideradas**

* Estilo por Capas
* Estilo Pipe and Filter
* Estilo por Eventos
* Estilo por Microservicios

**Decisión tomada**

**Consecuencias Positivas**

**Consecuencias Negativas**

Título

Estatus: Pendiente

Identificador: ADD-005.md

Problema

Existe una familia de dispositivos IoT compuesta por tres sensores en los que el primero envía información al segundo y este al tercero que finalmente lo envía al centro de notificaciones.

Requisitos

RF-08

Opciones

Estilo por capas

Estilo Pipe and Filter

Decisión Tomada

Estilo por capas

Consecuencias Positivas

Consecuencias Negativas

Identificador

ADD-004

Problema

Existen tres familias de sensores, cada una de las cuales comparte cierta funcionalidad, pero dispone de otras diferentes entre una familia y otra.

Requisitos

RF-05

Decisión

Factory Method

Estatus

Pendiente

Alternativas

Abstract Factory

Consecuencias Positivas

Consecuencias Negativas

Identificador

ADD-007

Problema

Además, los operarios de la factoria 4.0 debe estar permanente notificados a través de un sistema de mensajería interno y deben poderse suscribir a diferentes eventos y notificaciones como actualizaciones de la producción, fallos en los sensores o sobrecarga en la producción.

Requisitos

RF-06, RF-07

Decisión

Observer

Estatus

Pendiente

Alternativas

Publish-Suscribe

Consecuencias Positivas

Consecuencias Negativas