ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA DEL SOFTWARE

Ejercicio 1 - Diseño de una Arquitectura Software

Diseño y Arquitectura del Software

Integrantes
SAMUEL RUSU
MARÍA ESTEBAN SÁNCHEZ
SERGIO VILLAGARCÍA SÁNCHEZ
JESÚS ORTIZ LOPO
CARLOS HERNÁNDEZ HERNÁNDEZ
MARIO RECIO MONTERO

Portavoz
s.rusu.2019@alumnos.urjc.es

ÍNDICE

1.ROLES

1.1.ASS

- SAMUEL RUSU
- SERGIO VILLAGARCÍA SÁNCHEZ

1.2.ASC

- CARLOS HERNÁNDEZ HERNÁNDEZ
- MARÍA ESTEBAN SÁNCHEZ

1.3.ASJ

- MARIO RECIO MONTERO
- JESÚS ORTIZ LOPO

2.REQUISITOS

	Nombre	Descripción
RF1	**Comunicación entre módulos**	Se necesita una arquitectura con emisores de eventos (sistema de sensores IoT), consumidores (módulos y operarios) y canales para transmitir dichos eventos (pudiendo servirse de un sistema de mensajería interno).
RF1.1	**Comunicación de la información de los sensores**	El sistema comunicar la información producida por las diferentes familias de sensores a un módulo central de notificaciones, Cockpit.
RF1.2	**Comunicación de la información de las líneas de producción**	El sistema debe recoger información relativa a las líneas de producción, y enviarla al Cockpit para su tratamiento.
RF2	**Base de datos**	Debe incluirse en el sistema una base de datos NoSQL para almacenar todos los datos que requiere el sistema.
RF2.1	**Almacenamiento e identificación de órdenes**	La base de datos almacenará las órdenes de trabajo, Se debe incluir y almacenar para cada orden de trabajo, un identificador asociado y el nombre del operario u operarios designados a realizar dicha tarea.
RF2.2	**Almacenamiento e identificación de materiales**	La base de datos contiene la información de los materiales disponibles en la fábrica, y debe incluirse y almacenarse junto a cada material, su identificador y nombre asociados.
RF2.3	**Almacenamiento de datos de sensores**	La base de datos debe almacenar, además, todos los datos producidos por los sensores y cuál es su estado.
RF3	**Sistema de mensajería**	Debe incluirse un sistema de mensajería, que permita comunicar la información y los eventos, entre cada uno de los módulos de la arquitectura.
RF3.1	**Suscripción de los operadores**	El sistema de mensajería debe permitir a los operarios suscribirse a diferentes eventos y estar permanentemente conectados para recibir notificaciones como actualizaciones de la producción, fallos en los sensores o sobrecarga en la producción.
RF4	**Módulo de ordenes de trabajo**	Se debe incluir un módulo de asignación de órdenes de trabajo, que permita asignar a los

		operarios y las máquinas necesarias para fabricar los componentes.
RF5	**Módulo de selección de algoritmos**	El sistema debe contar con, al menos, dos algoritmos distintos, y dependiendo de la necesidad del sistema, debe poder seleccionarse el adecuado. El tipo de algoritmo se especificarán en los subapartados de este requisito.
RF5.1	**Algoritmo de optimización de volumen de trabajo**	Ya que se enviarán múltiples ordenes de trabajo, se requiere implementar un algoritmo predictivo que gestione el volumen y la gestión de dichas órdenes, con el objetivo de evitar saturaciones del sistema y de los operarios.
RF5.2	**Algoritmo de predicción de fallo**	Ya que es posible que se produzcan incidencias en las líneas de trabajo, es necesario incluir un algoritmo predictivo para detectarlos, y asignar los recursos necesarios desde otras líneas, hasta que pueda solucionarse el problema.
RF6	**Componente visual**	Los operarios deben poder visualizar los datos en tiempo real del proceso productivo, así como información de las órdenes de trabajo emitidas y las líneas de trabajo.
RF8	**Implementación de 3 familias de sensores**	Los sensores IoT se clasifican en tres familias, y cada una de estas tres familias, cuenta con funcionalidades características, de forma que se debe dar soporte a cada una de estas variantes.
RF8.1	**Comunicación de los 3 sensores**	Se especifica que existen tres sensores de una determinada familia, que siguen un patrón de comunicación especifico, donde el primero envía información al segundo y este al tercero, que finalmente lo envía al centro de notificaciones.
RF8.2	**Compatibilidad entre sensores**	Es necesario asegurar la compatibilidad entre los datos producidos por el sensor dos y el sensor tres de la familia especificada en el RF8.1. Por ello, es preciso incluir un mecanismo de adaptación de estos.
RF9	**Canal de comunicación**	El sistema será invocado desde un módulo de seguridad externo, basado en un servicio web, donde cada usuario (operario), debe identificarse con un nombre y contraseña.

Tabla 1: Tabla de requisitos

3.DECISIONES TOMADAS Y ARQUITECTURAS RESULTANTES

3.1 ITERACIÓN 1

3.1.1 DECISIONES

— **Decisión 1/ADR1-01:** Se necesita un sistema de gestión y envío de notificaciones, que reaccione a las modificaciones detectadas por los sensores, enviando las señales pertinentes.

Solución: Arquitectura basada en eventos, que contará con tres módulos conectados por un sistema de mensajería interno. Los módulos se separarán en: productores (sensores), un módulo central manejador (Cockpit) y consumidores (usuarios suscritos al sistema).

— **Decisión 2/ADR1-03:** Se necesita implementar una familia de 3 tipos de sensores, de forma que compartan funcionalidades, pero se diferencien por otras propias.

Solución: Ya que se necesitan 3 familias de sensores, declaramos interfaces abstractas para instanciar cada una de ellas, e implementaremos un patrón Abstract Factory para su creación.

— **Decisión 3/ADR1-07:** Se necesita implementar una familia de 3 tipos de sensores que se comuniquen entre sí, de forma que el primero envía información al segundo y este al tercero que finalmente lo envía al centro de notificaciones.

Solución: Ya que se trata de una comunicación lineal de los sensores, y no se comparten recursos entre los distintos sensores, no es necesario implementar ningún patrón especifico que complique más la solución, y un patrón por capas sería incorrecto. La comunicación se hará por tanto de forma simple y directa, sin un estilo específico.

— **Decisión 4/ADR1-10**: Se requiere un módulo de órdenes de trabajo para gestionar toda la operativa de cada trabajador y máquina, así como un componente de visualización para mostrar dichas órdenes, y otra información analítica.

Solución: Ya que debe existir un componente de visualización, encargado de mostrar las órdenes de trabajo, y las analíticas de los datos, se incluirá dicho elemento junto al módulo de órdenes, para representar esta relación.

— **Decisión 5/ADR1-11:** Los operarios de la factoría 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.

Solución: Se incluirá el módulo de comunicación interno como mecanismo de traspaso de mensajes y eventos entre los distintos módulos, que permita además a los operarios conectarse y consultar información de relevancia.

— **Decisión 6/ADR1-13:** Los operarios de la factoría 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.

Solución: Se utilizará un patrón Publish Subscribe para que los operarios puedan suscribirse a distintos tópicos de interés, permitiendo que sean notificados de estos a través del sistema de mensajería integrado.

— **Decisión 7/ADR1-15:** Se requiere que el sistema sea capaz de procesar los eventos que hayan sido captados por los sensores, y posteriormente transmitir dichos datos al módulo Cockpit.

Solución: Se plantea incluir un sistema que procese los eventos recogidos por diferentes familias de sensores y los transmita a un centro de notificaciones denominado Cockpit.

3.1.2 ARQUITECTURA RESULTANTE

Sistema de mensajería:

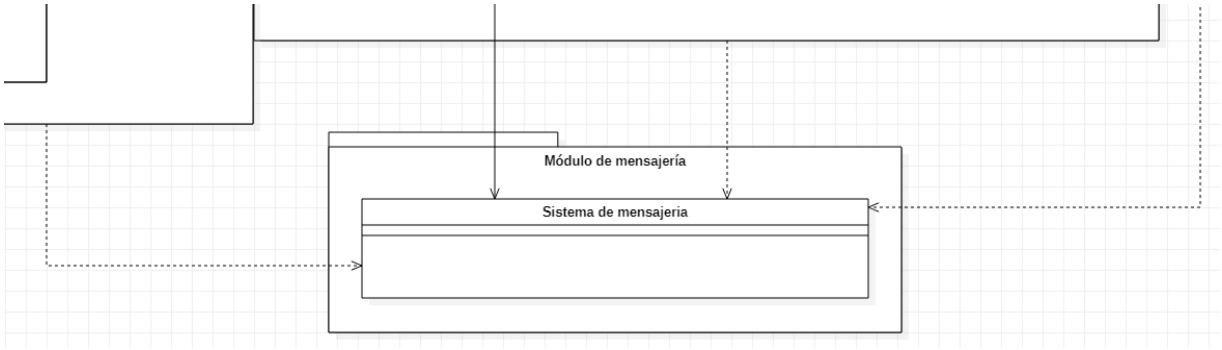

Imagen 1.1: Sistema de mensajería

Módulo consumidor de eventos:

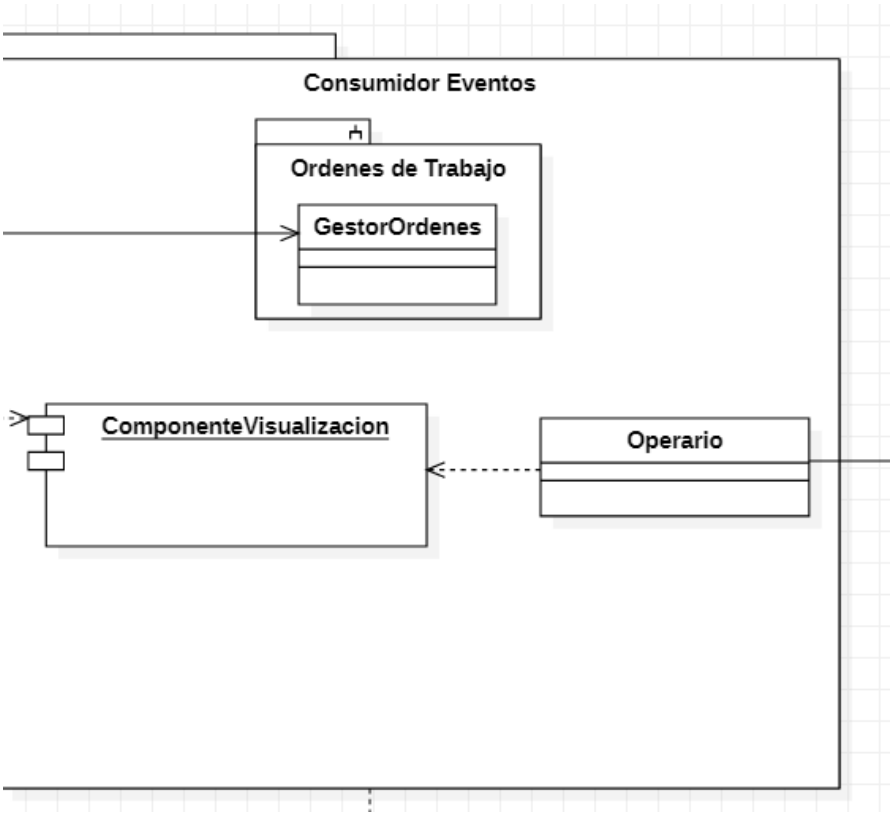

Imagen 1.2: Módulo consumidor de eventos

Módulo cockpit:

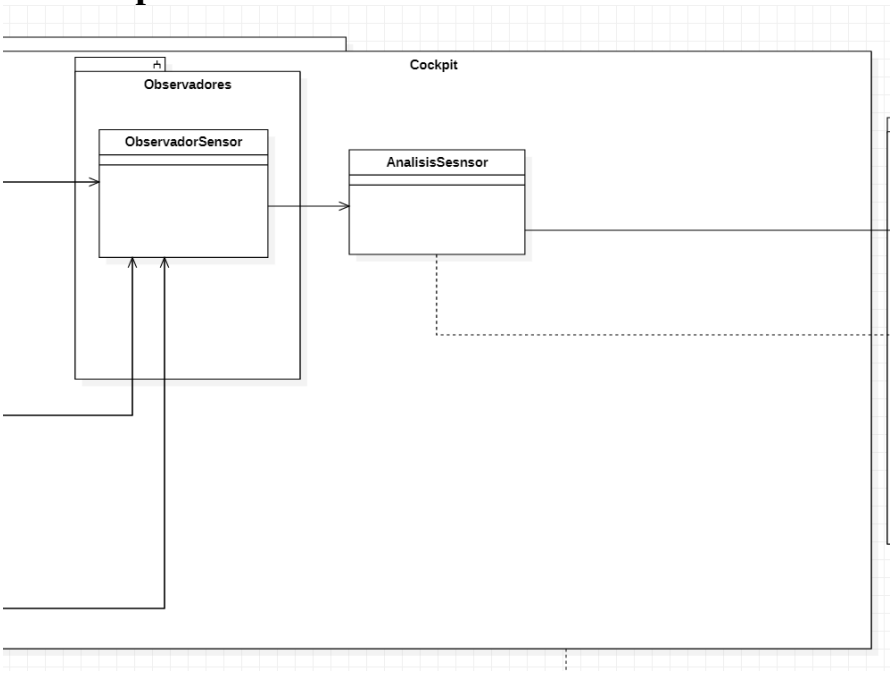

Imagen 1.3: Módulo cockpit

Módulo productor de eventos:

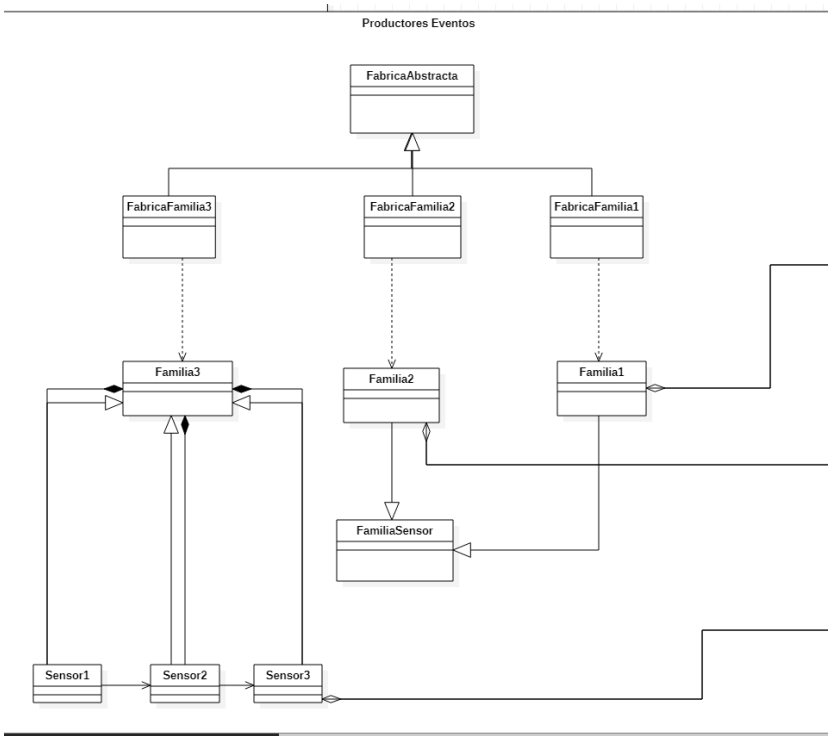

Imagen 1.4: Módulo productor de eventos

3.2 ITERACIÓN 2

3.2.1 DECISIONES

— **Decisión 1/ADR2-01:** Se necesita una base de datos NoSQL, que va a almacenar toda la información relacionada con los agentes que operan en la aplicación, la propia factoría y sus correspondientes productos.

Solución: Se incluirá un componente en la arquitectura, que represente la base de datos requerida, y se conecte con el módulo cockpit para recibir la información procesada y almacenarla.

— **Decisión 2/ADR2-03:** Se necesita un sistema que detecte cambios en los sensores o información de las líneas de trabajo, las capture y comunique al módulo cockpit para el tratamiento de dichos datos.

Solución: Se utilizará un patrón observador, con un observador para los sensores, y otro para las líneas de trabajo. Estos observadores transmitirán los datos al módulo central para su tratamiento.

— **Decisión 3/ADR2-05:** Se deben invocar los servicios de un módulo de seguridad externo para acceder al sistema, por lo que se debe proporcionar un canal de comunicación adecuado entre ambos sistemas.

Solución: Mediante el uso de un componente gateway, se puede modelizar esta relación entre sistemas, u otras futuras, proporcionando un canal adecuado.

— **Decisión 4/ADR2-07:** El sistema hará uso de distintos algoritmos predictivos para mejorar el rendimiento general de la factoría, pero se debe emplear el algoritmo adecuado en el lugar y momentos necesarios.

Solución: Se utilizará el patrón Strategy, para definir una interfaz común a todos los algoritmos que se desee implementar, siendo solo necesario definir las diferencias y funcionamientos específicos de cada uno, e incluirlos en el módulo correspondiente.

— **Decisión 5/ADR2-09:** La información acerca de los sensores, líneas de trabajo y órdenes de trabajo, debe ser accesible a los operarios, a traves del componente de visualización, por lo que es necesario analizar dicha información, y transmitirla en un formato organizado, que sea comprensible por los empleados.

Solución: Para ello, la información captada en los observadores se envía a un módulo de análisis y tratamiento de información en el cockpit, que contendrá las herramientas necesarias para tratar dicha información, y transmitirla al componente de visualización.

— **Decisión 6/ADR2-11:** Es necesario asegurar la compatibilidad entre los datos producidos por el sensor dos y el sensor tres de la familia especificada en el RF8.1. Por ello, es preciso incluir un mecanismo intermedio entre ambos, que sirva de adaptación de estos.

Solución: El patrón Adapter permite utilizar una interfaz, que sirva de intermediario entre ambos sensores, adaptando la comunicación entre estos, y asegurando la compatibilidad.

3.2.2 ARQUITECTURA RESULTANTE

Sistema de mensajería:

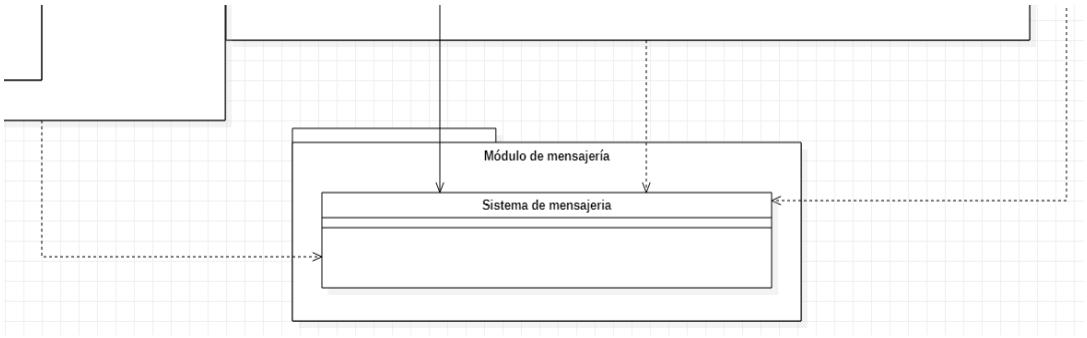

Imagen 2.1: Sistema de mensajería

Módulo consumidor de eventos:

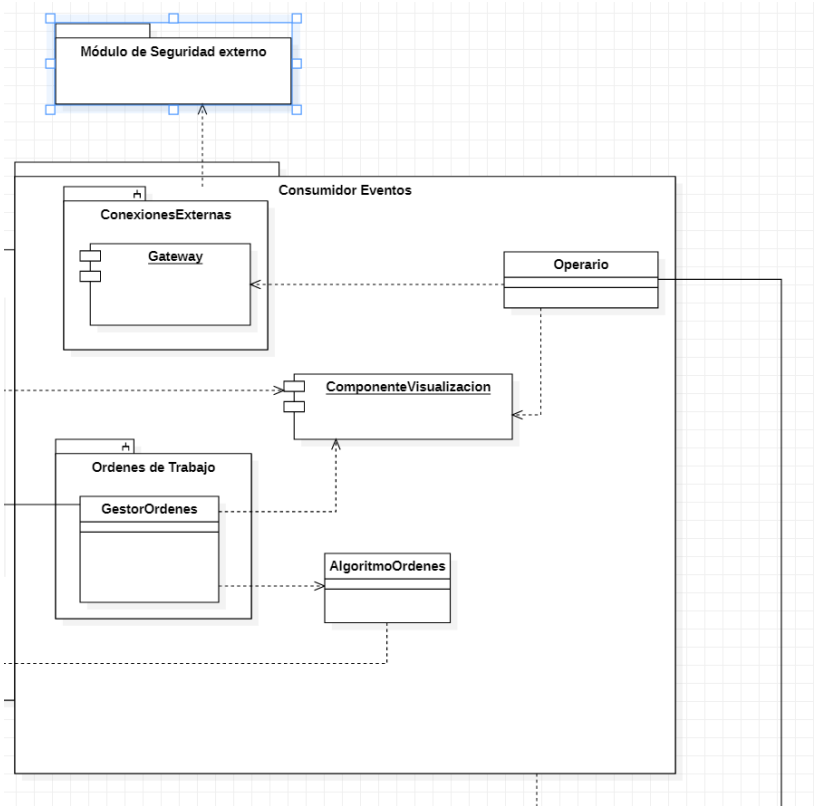

Imagen 2.2: Módulo consumidor de eventos

Módulo cockpit:

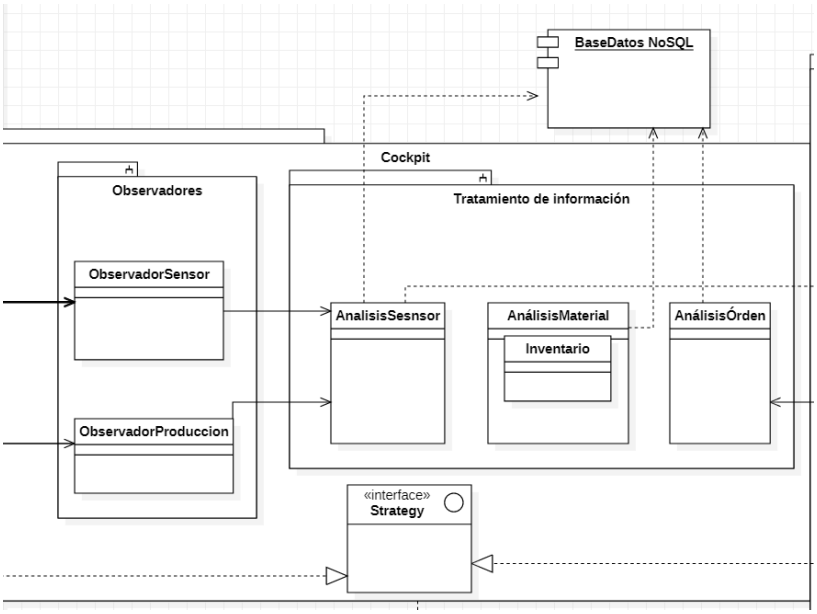

Imagen 2.3: Módulo cockpit

Módulo productor de eventos:

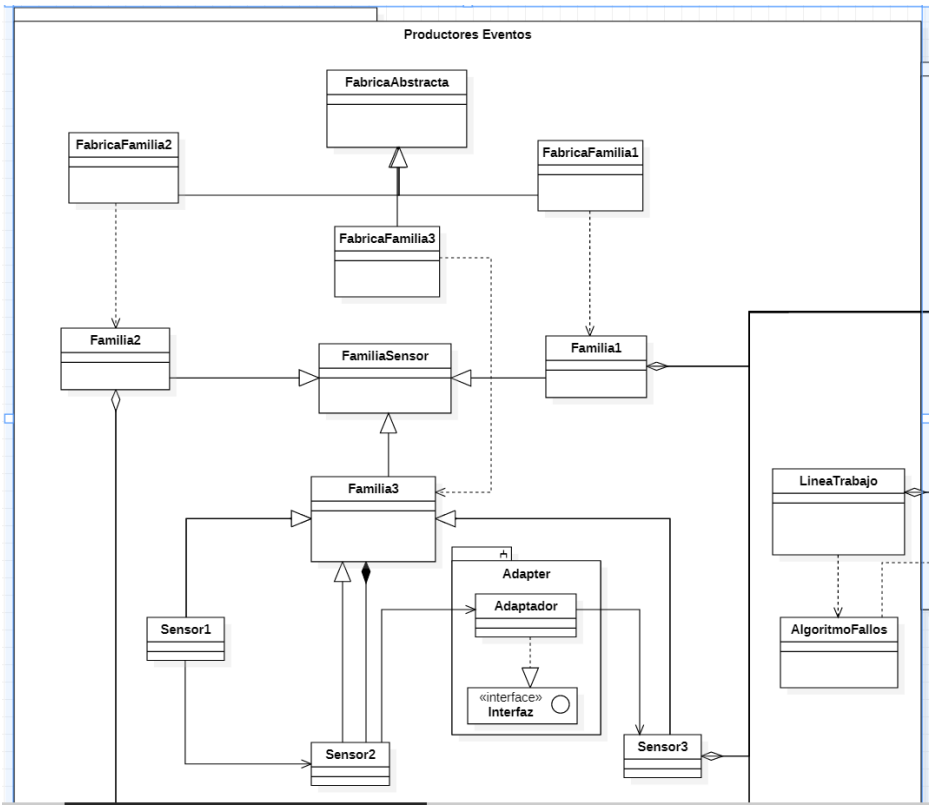

Imagen 2.4: Módulo productor de eventos

4.CONCLUSIONES

Al ser un ámbito desconocido, al principio nos resultó complicado tomar las decisiones más acertadas con relación al problema planteado, pero a medida que nos íbamos familiarizando con el trabajo pudimos resolver los contratiempos de forma más rápida y sencilla.

Dificultad a la hora de cohesionar todos los módulos ya que nunca habíamos enfrentado un diseño de este calibre.

En la tercera iteración, tuvimos un desacuerdo notable a la hora de decidir que patrón utilizábamos en la decisión 6. Al final llegamos a la conclusión de que la mejor solución era utilizar el patrón publish-subscribe.

5.BIBLIOGRAFÍA

https://es.wikipedia.org/wiki/Cliente-servidor#:~:text=La%20arquitectura%20cliente%2Dservidor%20es,servidor%2C%20quien%20le%20da%20respuesta.

https://refactoring.guru/es

https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b

https://somospnt.com/blog/118-arquitectura-de-capas

https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/capas

https://cloud.google.com/solutions/event-driven-architecture-pubsub?hl=es-419

https://medium.com/@Apoorv_Saxena/introducing-asyncresolver-js-resolve-async-subscribed-decisions-bead640b1352

https://refactoring.guru/es/design-patterns/behavioral-patterns

https://refactoring.guru/es/design-patterns/structural-patterns

https://learn.microsoft.com/es-es/azure/architecture/patterns/index-table

https://programacion.net/articulo/patrones_de_diseno_xix_patrones_de_comportamiento_mediator_1022

6.TABLA DE TIEMPOS

Week	Iteration	Time in ADD (ASS)	Reflection time (ASS-ASC)	Time in refined ADD (ASS)	Design ADD time (ASJ)
1	**1**	130	65	190	55
2	**2**	110	55	170	85

Tabla 2: Tabla de tiempos