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

ASS:

- SAMUEL RUSU
- SERGIO VILLAGARCÍA SÁNCHEZ

ASC:

- CARLOS HERNÁNDEZ HERNÁNDEZ
- MARÍA ESTEBAN SÁNCHEZ

ASJ:

- MARIO RECIO MONTERO
- JESÚS ORTIZ LOPO

2.ADMENTOR Y REQUISITOS FUNCIONALES

	Nombre	Descripción
RF1	**Centro de notificaciones**	Centro de notificaciones para el recibo de datos de los sensores y visualización de las analíticas. Desde este módulo se pueden gestionar todas las funcionalidades del software.
RF2	**Almacenamiento de inventario**	Añadir una base de datos SQL, que almacenará tanto las órdenes de trabajo, como el inventario de todo el material existente.
RF2.1	**Almacenamiento de datos de sensores**	Añadir una base de datos SQL, que almacenará todos los datos de los sensores
RF3	**Sistema de mensajería**	los operarios de la factoría 4.0 debe estar permanente notificados a través de un sistema de mensajería interno.
RF3.1	**Suscripción de los operadores**	Poderse suscribir a diferentes eventos y notificaciones como actualizaciones de la producción, fallos en los sensores o sobrecarga en la producción.
RF4	**Módulo de ordenes de trabajo**	Incluir un módulo de asignación de órdenes para operarios y máquinas que van a fabricar cada componente.
RF5	**Módulo de selección de algoritmos**	Dependiendo del momento de la producción elige el algoritmo que le corresponde.
RF5.1	**Algoritmo de optimización de volumen de trabajo**	Ya que se enviarán múltiples ordenes de trabajo, se requiere implementar un algoritmo que gestione el volumen y la gestión de dichas órdenes.
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 para detectarlos, y asignar los recursos necesarios desde otras líneas.
RF6	**Componente visual**	Incluir un componente de visualización para mostrar los datos en tiempo real del proceso productivo y las órdenes de trabajo.
RF7	**Medidas de seguridad**	Se requiere implementar medidas de seguridad para gestionar el acceso de los usuarios con el software.
RF7.1	**Seguridad en los mensajes**	A la hora de mandar y recibir mensajes, se deberá tener en cuenta que sea un sistema fiable, utilizando alternativas como Apache Kafka o MQTT.
RF7.2	**Límite de intentos de conexión**	Si el número de intentos supera los permitidos, se deberá suspender el intento de acceso al

		software y se considerará al dispositivo como fuera de servicio.
RF8	**Implementación de 3 familias de sensores**	Ya que los sensores IoT se clasifican en tres familias, cada una con ciertas funcionalidades características, se debe dar soporte a cada una de estas variantes.
RF8.1	**Comunicación de los 3 sensores**	Existen tres sensores de una determinada familia, donde el primero envía información al segundo y este al tercero que finalmente lo envía al centro de notificaciones.
RF10	**Componente Gateway**	Para poder invocar un simulador externo, el software incorporará un componente Gateway que sirva como pasarela de comunicación.

3. DECISIONES TOMADAS Y ARQUITECTURAS RESULTANTES

Iteración 1:

— Decisión 1: Se necesita un sistema de gestión y envío de notificaciones, que reaccione a los sensores, enviando las señales pertinentes.

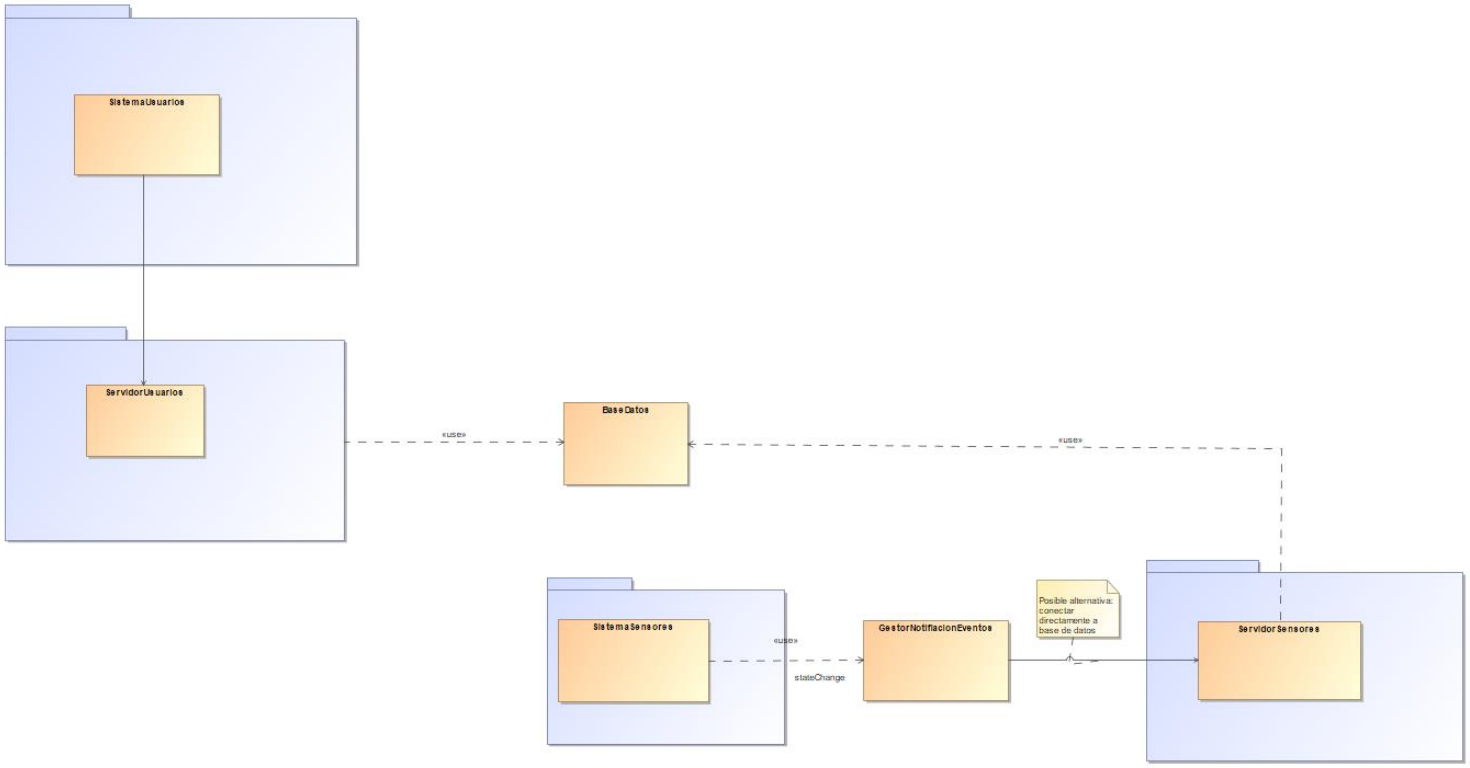
Solución: Arquitectura con emisores de eventos, consumidores y canales para transmitir los eventos, donde los consumidores son responsables de reaccionar a los eventos.

— Decisión 2: Se requiere una pasarela a través de la cual los operarios que quieran consumir ciertos eventos o notificaciones se den de alta en el sistema de notificaciones.

Solución: Arquitectura cliente servidor para soportar peticiones y poder persistir los cambios o actualizaciones de datos en el sistema.

Nota: La decisión para incluir la base de datos se lleva a cabo en la siguiente iteración, a pesar de estar presente en esta arquitectura, ya hubo que hacer cambios en los requisitos, y se formalizó la decisión en la iteración 2 (decisión 4).

Arquitectura resultante:



Iteración 2:

— Decisión 3: Se requiere un módulo de órdenes de trabajo para gestionar toda la operativa de cada trabajador y máquina.

Solución: Dentro de los usuarios que consumen las notificaciones enviadas por el sistema interno, habrá un apartado específico para gestionar las necesidades de cada operador.

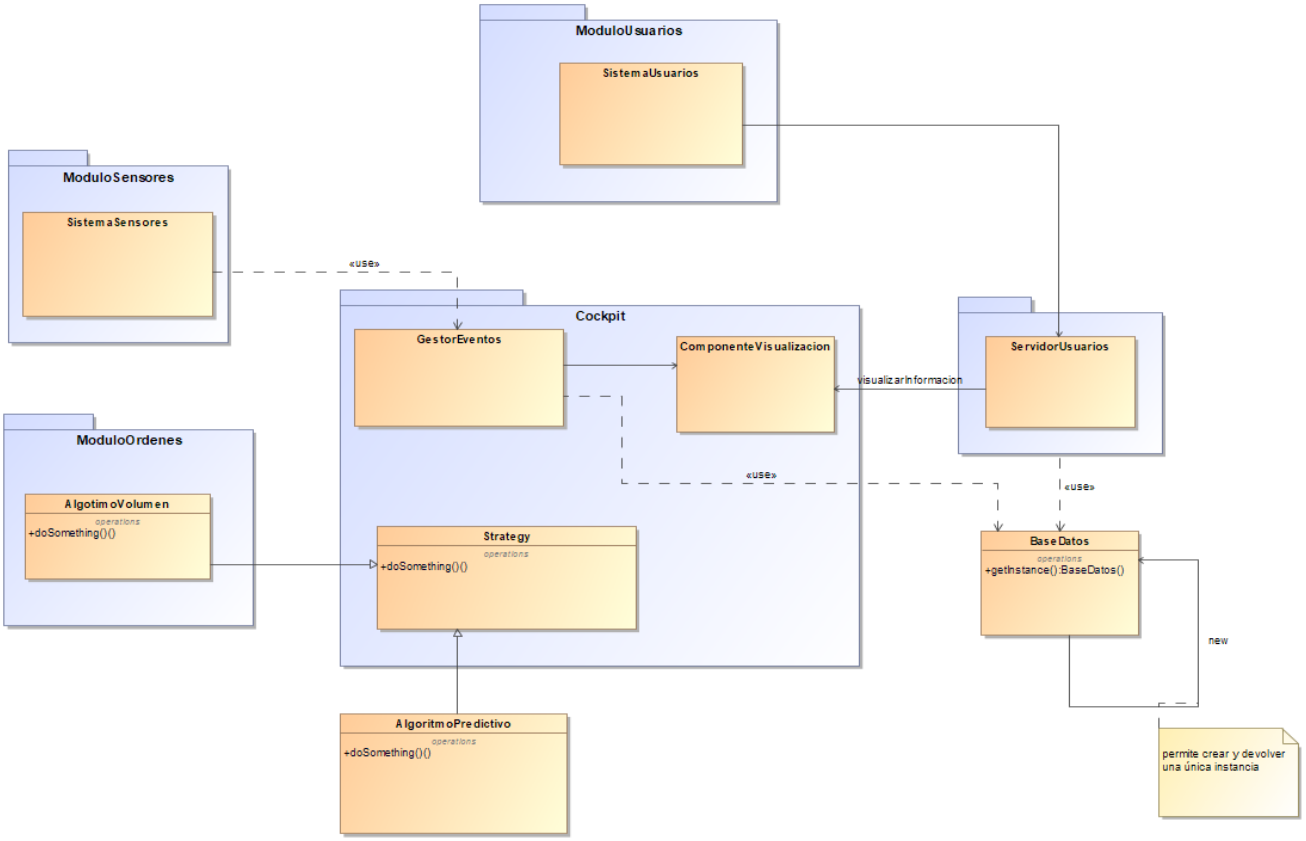
— Decisión 4: Es necesaria una base de datos SQL para gestionar todo el espacio de almacenamiento de los datos que producen los sensores.

Solución: Se utilizará el patrón Singleton para asegurar que la base de datos tenga una única instancia.

— Decisión 5: Se requiere un patrón para seleccionar el algoritmo inteligente predictivo óptimo en cada situación.

Solución: El patrón Strategy permitirá cambiar el algoritmo seleccionado dependiendo de las necesidades de nuestro sistema.

Arquitectura resultante:



Iteración 3:

— Decisión 6: Se requiere un sistema que gestione las subscripciones de usuarios a eventos de interés.

Solución: Se usará un patrón Publish-Subscribe para permitir la subscripción a eventos de su interés. Para ello, estableceremos un "middleware" que recibirá las notificaciones relevantes del gestor de eventos.

— Decisión 7: Se requiere establecer un límite de intentos en las conexiones con los dispositivos.

Solución: Estableceremos un control de intentos de conexión en el middleware.

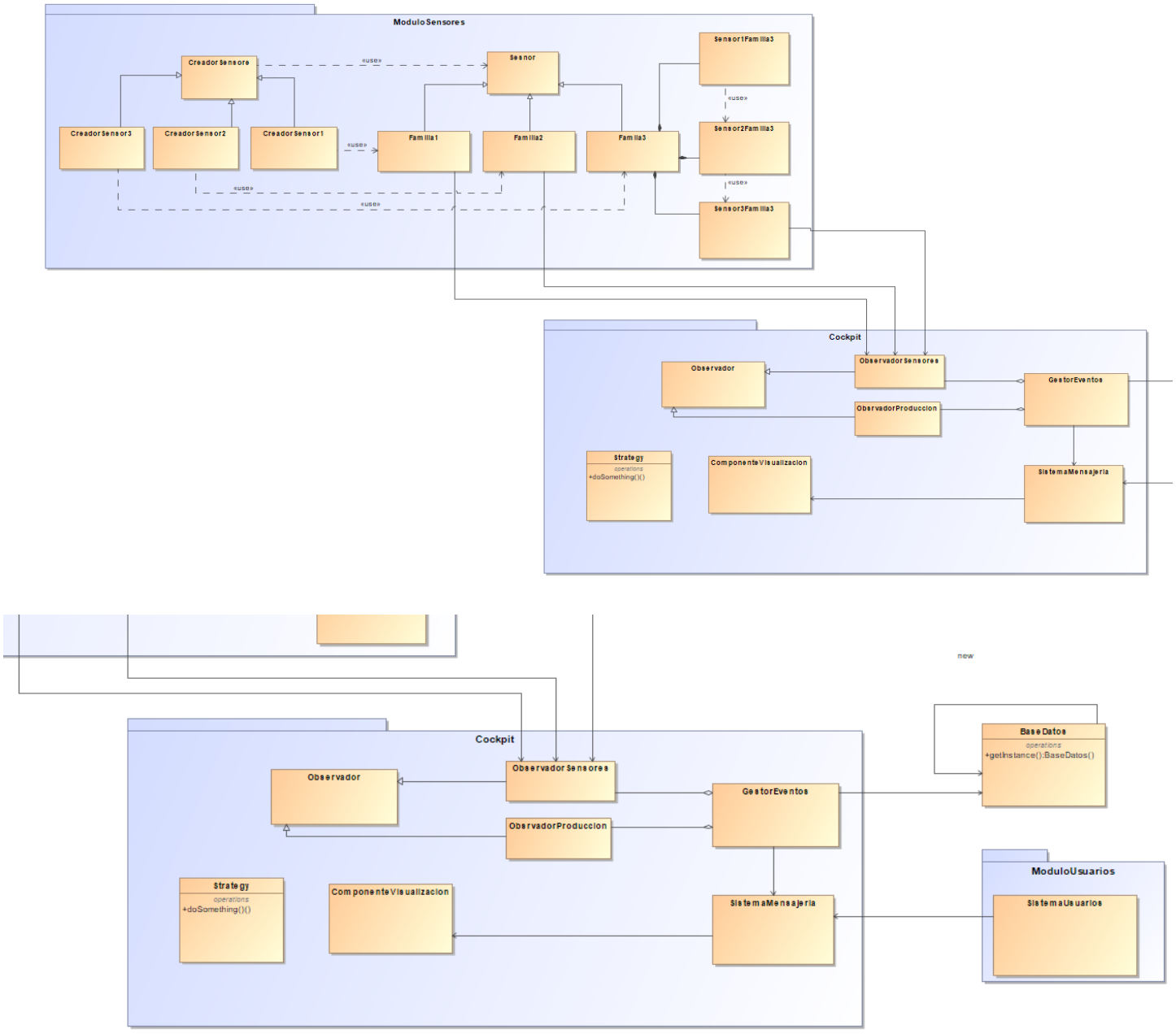
— Decisión 8: Se necesita implementar una comunicación entre 3 sensores, de forma que el primero envía información al segundo y este al tercero que finalmente lo envía al centro de notificaciones.

Solución: Ya que los sensores se conectan solo con otro o con el centro de notificaciones, en una única dirección, puede verse como un intercambio de datos de desde capas superiores a inferiores.

— Decisión 9: 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 la creación de cada una de ellas.

Arquitectura resultante:



Iteración 4:

— Decisión 10: Se necesita representar las líneas de trabajo como un productor de eventos, para recoger información de la línea.

Solución: Ya que es necesario mantener informados a los operarios, se incluirá un observador que capte los cambios en la línea de trabajo.

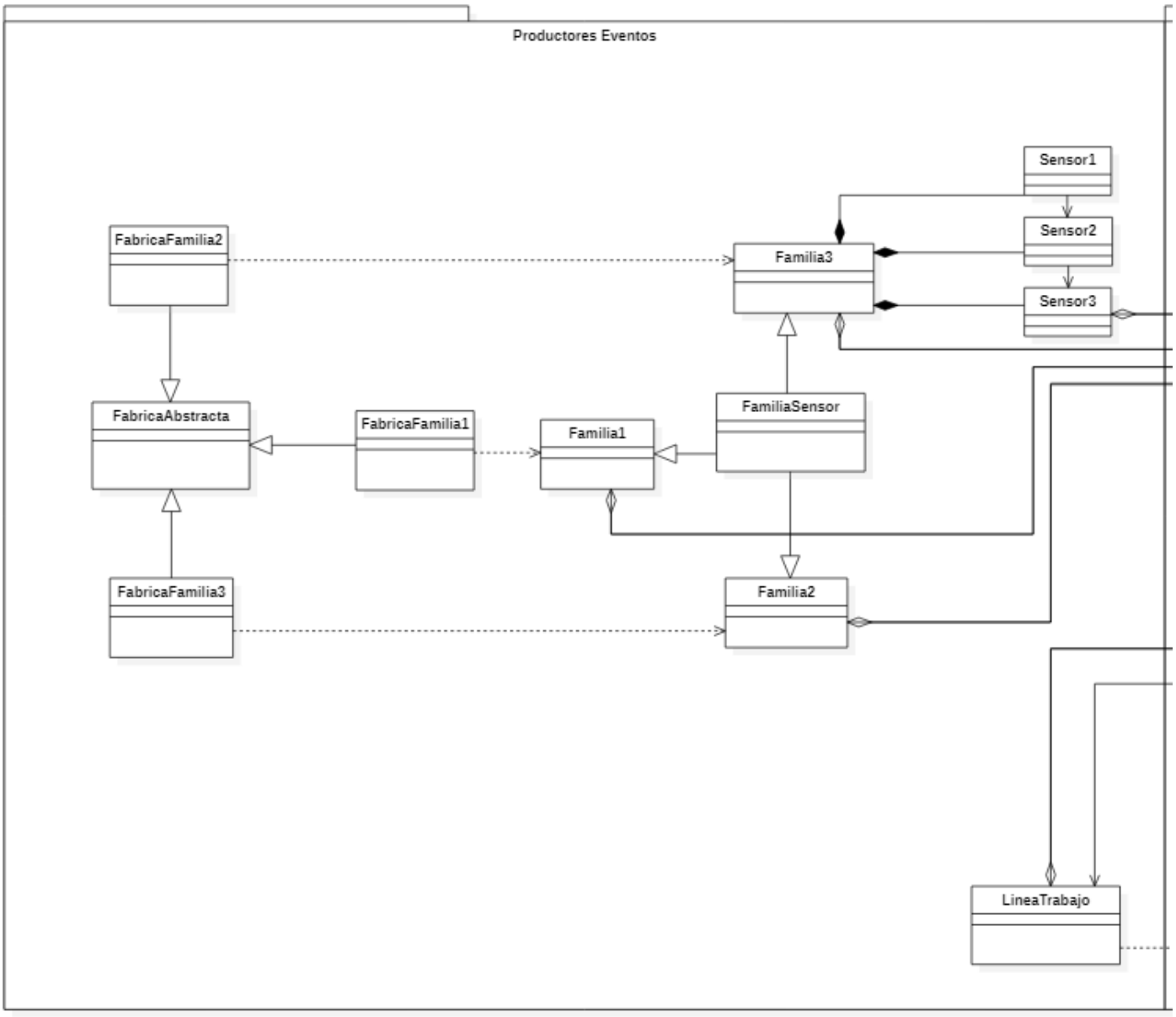
— Decisión 11: Se necesita de una pasarela que sirva de pasarela de comunicación.

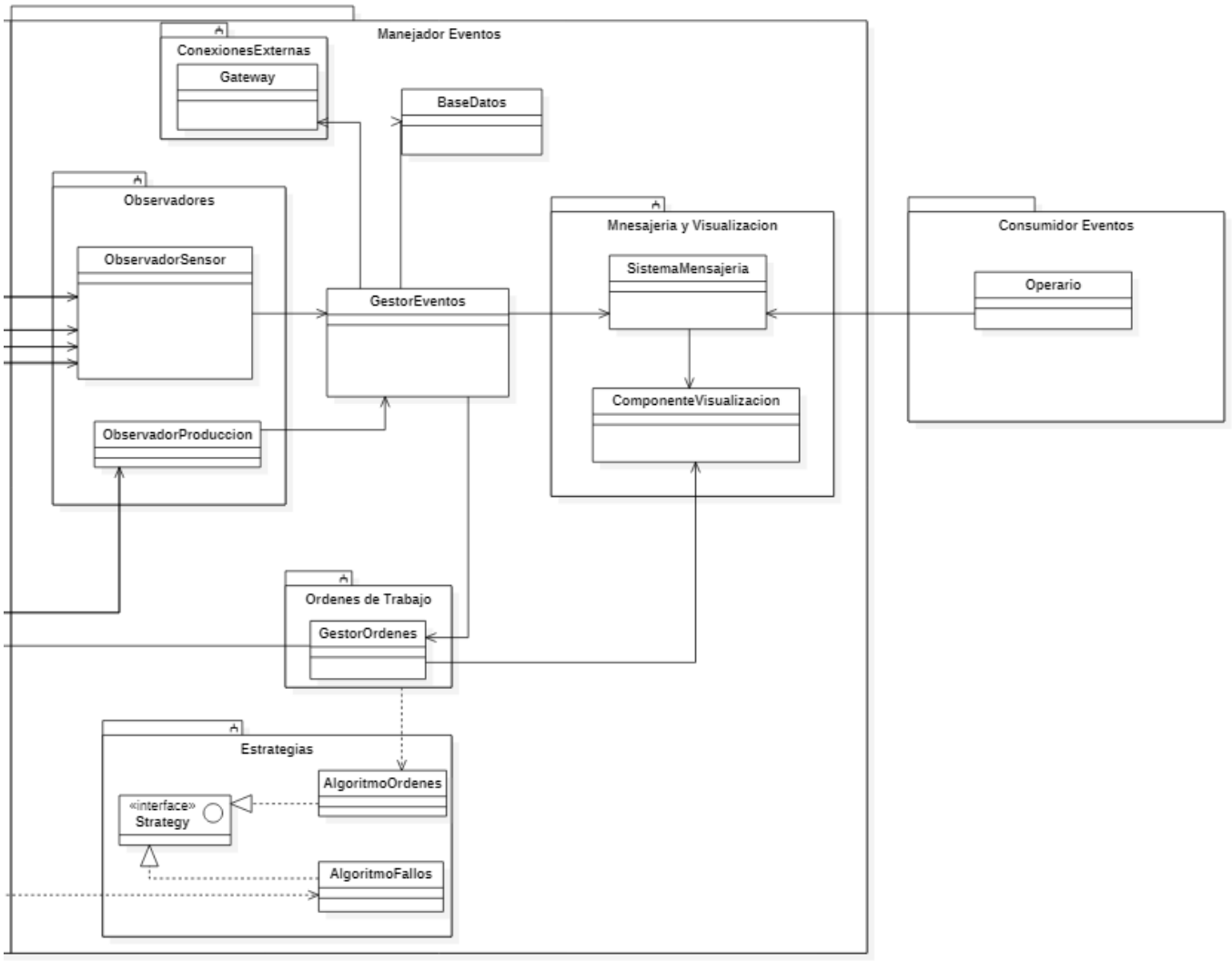
Solución: Utilizar un componente Gateway permite interconectar arquitecturas diferentes a todos los niveles de comunicación.

— Decisión 12: Se requiere un módulo de órdenes de trabajo para gestionar toda la operativa de cada trabajador y máquina.

Solución: Se incluirá un módulo de gestión para las órdenes de trabajo, que estará comunicado con la línea de trabajo y el componente de visualización.

Arquitectura resultante:


Productores Eventos
Sensor1
Sensor2
Sensor3
FabricaFamilia2
Familia3
FabricaAbstracta
FabricaFamilia1
Familia1
FamiliaSensor
FabricaFamilia3
Familia2
LineaTrabajo



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

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**	120’	90’	30’	30’
1	**1.1**	60’	45’	20’	20’
2	**2**	70’	60’	20’	30’
3	**3**	90’	70’	35’	40’
4	**4**	45’	30’	15’	25’

Tabla 1: Tabla de tiempos