Taller 1

Jorge Franco
Kevin Acuña
David Barrios
Juan Carlos Peñaranda
Sergio Camilo Silva

1 Contenido

1. Descripción del problema

El objetivo es diseñar un software para una fábrica inteligente. La fábrica está compuesta por 3 líneas de producción y más de 10 sensores que dan datos sobre el estado de los dispositivos físicos.

El software debe contar con las siguientes funcionalidades:

- Un componente de visualización que muestre analíticas en tiempo real del proceso productivo y las órdenes de trabajo.

- Un módulo de órdenes de trabajo con asignaciones de órdenes por operario y máquinas que van a fabricar cada componente.

- Un centro de notificaciones (Cockpit) donde se reciben los datos de los sensores IoT y se visualizan las analíticas.

- La BBDD deberá almacenar además de las órdenes de trabajo el inventario del material existente.

- Los sensores IoT se clasifican en dos familias (sensores A y B), cada una de las cuales comparte cierta funcionalidad, pero dispone de ciertas características diferentes entre una familia y otra. La familia de sensores A, está compuesta por tres sensores en los que el primero envía información al segundo y éste al tercero.

- El software contará con dos algoritmos inteligentes predictivos: uno para optimizar el volumen de órdenes de trabajo y otro para predecir el fallo de una línea de trabajo y asignar recursos de otras líneas. La selección del algoritmo adecuado es uno de los desafíos de diseño a resolver.

- Los operarios de la factoría 4.0 deben estar permanentemente 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.

- Se deberá elegir un sistema de mensajería fiable en cuanto a mensajes enviados. Algunas alternativas son MQTT, Apache Kafka o similares. Por otra parte, si el número de intentos de conectar con un dispositivo que falla supera un umbral entonces se deberán suspender los intentos de conexión y considerar al dispositivo fuera de servicio.

2 Construcción de arquitectura

2.1 Requisitos funcionales

Req Funcional	Nombre	Descripción
RF1	Panel de control	Yo como jefe de producción quiero poder ver los datos analíticos del proceso productivo y las órdenes de trabajo en tiempo real así como los datos de los sensores de tal manera que pueda tomar decisiones sobre la esta.
RF2	Órdenes asignadas	Yo como operario quiero tener un módulo de administración de órdenes de trabajo con mis órdenes asignadas.
RF3	Almacenar inventario	Yo como jefe de producción quiero tener el inventario del material existente.
RF4	Optimización de órdenes	Yo como jefe de producción quiero optimizar el volumen de órdenes de trabajo.
RF5	Predicción de fallos	Yo como jefe de producción quiero predecir el fallo de las líneas de producción y que las otras líneas suplan el fallo.
RF6	Notificaciones	Yo como operario quiero estar notificado con un sistema de mensajería interna
RF7	Suscripciones	Yo como operario quiero poder suscribirme a diferentes eventos y notificaciones como el estado, fallos y sobrecargas de la producción.
RF8	Captura de información IoT	Yo como jefe de producción quiero poder capturar la información de 10 sensores IoT que miden el estado de los dispositivos en la fábrica.
RF9	Transmisión de información de sensores	Yo como jefe de producción quiero poder transmitir la información de los sensores a un sistema de almacenamiento
RF10	Almacenamiento de información de sensores	Yo como jefe de producción quiero poder tener la información de los sensores almacenada en un sistema
RF11	Configuración de secuencia	Yo como jefe de producción quiero poder configurar la secuencia entre tipologías de sensores
RF12	Interfaz configurable	Yo como operario quiero poder tener una interfaz configurable donde se encuentren los mecanismos de notificación de eventos.
RF13	Intentos de conexión	Yo como jefe del sistema quiero que el dispositivo del operario se considere como fuera de servicio si se supera el umbral de intentos de conexión fallida.

2.2 Problemas de diseño

- Adquirir datos de los sensores IoT (tiempo real)
- Configuración de secuencia de actuadores de las líneas de sensores de IoT
- Almacenar información de los
    - Inventarios
    - Ordenes
    - Operarios
    - Maquinas
    - Sensores, Tipos y Secuencias de operación
- Correlación entre sensores
- Generación de data correlacionada para visualización y toma de decisiones.
- Entrenamiento, Elección y Aplicación de modelo de Machine Learning para generación información predictiva (ordenes de trabajo y predicciones de fallo)
- Orquestación de mensajes de comunicación entre sistemas

2.3 Toma y captura de decisiones de diseño

Puede acceder al repositorio de GitHub con la captura de decisiones de diseño en el siguiente enlace: https://github.com/arq-nuevas-tecnologias-G5/taller1.

2.3.1 Estilo de la Arquitectura General

ADR: Estilo Arquitectura - Event-Diven

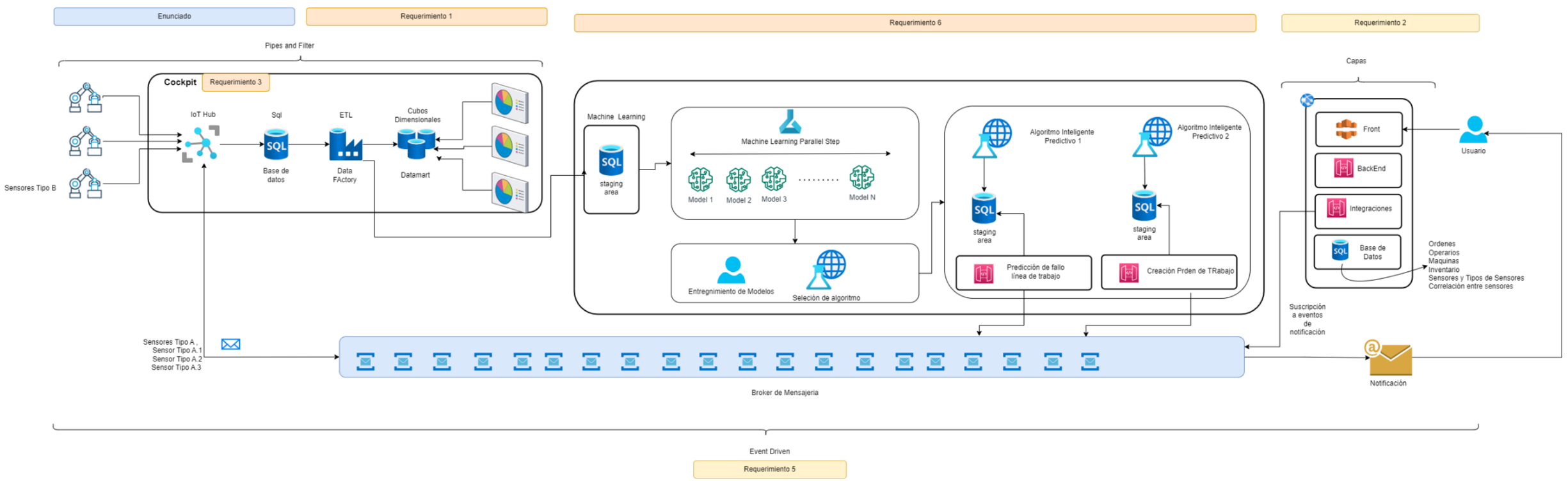
El problema abordado con esta arquitectura requiere grupos de componentes funcionales que generan, transforman, visualizan y/o publican información e intercambian datos entre ellos de forma asíncrona para la realización de acciones ante eventos programados.

Se visualiza un conjunto de interacciones entre grandes 5 grandes componentes de la solución, donde el mecanismo de comunicación es el intercambio de mensajes.

- Agrupación 1 Cockpit:
  - Interacción directa con dispositivos IoT, para la captura, almacenamiento, transformación y visualización de datos.
- Agrupación 2 analítica Avanzada:
  - Preparación de datos,
  - Validación de modelos de ML, entrenamiento y selección de modelo.
  - Aplicación de modelo.
  - Generación de datos para toma de decisiones
- Agrupación 3 Solución Web:
  - Solución de Software que permita la interacción con usuario final.
  - Almacén de datos paramétricas, inventario y configuración de la plataforma.
- Agrupación 4 Broker de mensajería
  - Componente para intercambio de información entre las componentes actuales y futuros de la solución.
- Agrupación 5 Notificación:
  - Componente para generación de notificaciones.

En marcado en esta decisión estos 4 grupos interactúan entre ellos mediante el intercambio de mensajes por eventos, como se presenta en la siguiente imagen.

Las decisiones de arquitectura subsecuentes que componen cada una de estas agrupaciones, presentan estilos propios detallados en cada ADR



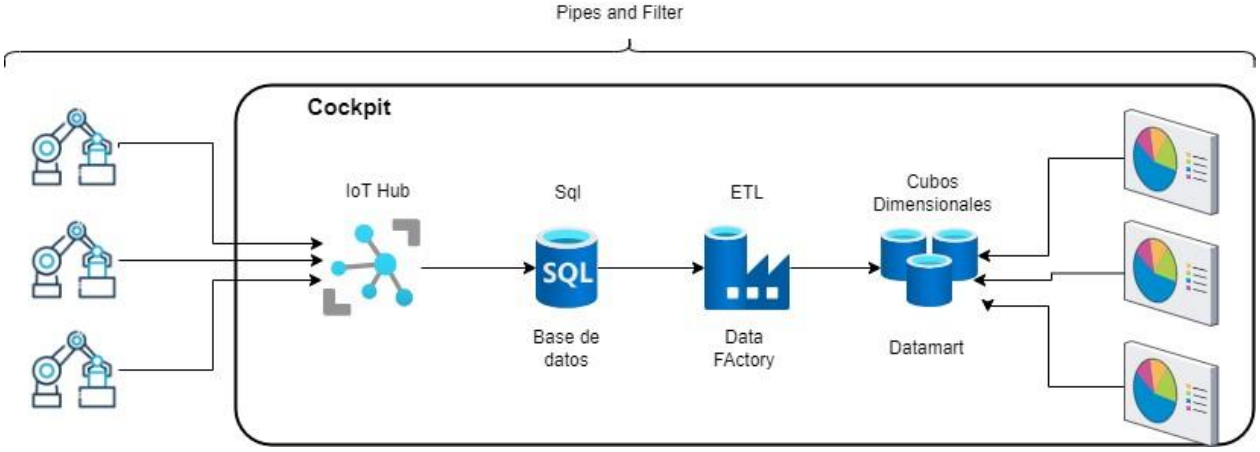
El estilo arquitectónico "Event-Driven" es un patrón de diseño que se basa en la producción, detección y reacción a los eventos que ocurren en un sistema. Este estilo ofrece varias ventajas para dar solución a un problema en arquitectura de software, tales como:

- Permite un bajo acoplamiento entre los componentes del sistema, ya que los productores de eventos no conocen a los consumidores ni las acciones que estos realizan. Por ejemplo, en una aplicación de comercio electrónico, el evento de realizar un pedido puede ser consumido por diferentes componentes, como el de facturación, el de envío o el de recomendación, sin que el componente que genera el evento tenga que saber cómo se procesa cada uno.
-
- Facilita el desarrollo de aplicaciones distribuidas y modernas, que requieren una comunicación asíncrona y dinámica entre los componentes. Por ejemplo, en una solución de IoT, los eventos generados por los sensores pueden ser transmitidos y procesados por diferentes plataformas y servicios en la nube, sin depender de una conexión permanente o sincronizada.
-
- Proporciona una mayor escalabilidad y rendimiento, al permitir el procesamiento paralelo y concurrente de los eventos. Por ejemplo, en una aplicación de Big Data, los eventos pueden ser almacenados y analizados por diferentes nodos o clusters, aprovechando la distribución y la replicación de los datos.
-
- Soporta diferentes modelos de arquitectura, como el pub/sub o el streaming de eventos, según las necesidades del problema a resolver. Por ejemplo, en una aplicación de streaming de vídeo, se puede usar un modelo pub/sub para enviar notificaciones a los usuarios cuando hay nuevos contenidos disponibles, y un modelo de streaming de eventos para transmitir los datos del vídeo en tiempo real.

Por estas razones, el estilo arquitectónico "Event-Driven" es una opción adecuada para diseñar soluciones de software que se adapten a contextos cambiantes y complejos, como los que se presentan en el ámbito de IoT, Big Data , analítica avanzada hasta machine learning.

2.3.2 Estilo Arquitectura Componente IoT

ADR: Estilo Arquitectura PIPES AND FILTERS - Componente IoT



El componente Cockpit, es implementado pajo del Estilo Pipes and Filter, con de mediante una secuencia de pasos y transformaciones se obtienen diferentes resultados así:

- Agrupación 1 Cockpit, descripción de aplicación:
  - Ingesta de datos realizada por un componente especializado que permite captura de sensores IoT en tiempo real sin pérdida de información.
  - Almacenamiento en base de datos relacional para garantizar durabilidad.
  - componente ETL
    - Transformación
    - Limpieza
    - completado
    - calidad
  - Almacenamiento en múltiples bases de datos relacionales con información dimensional por dominio de negocio.
  - Componentes especializados para visualización e interacción con los datos.

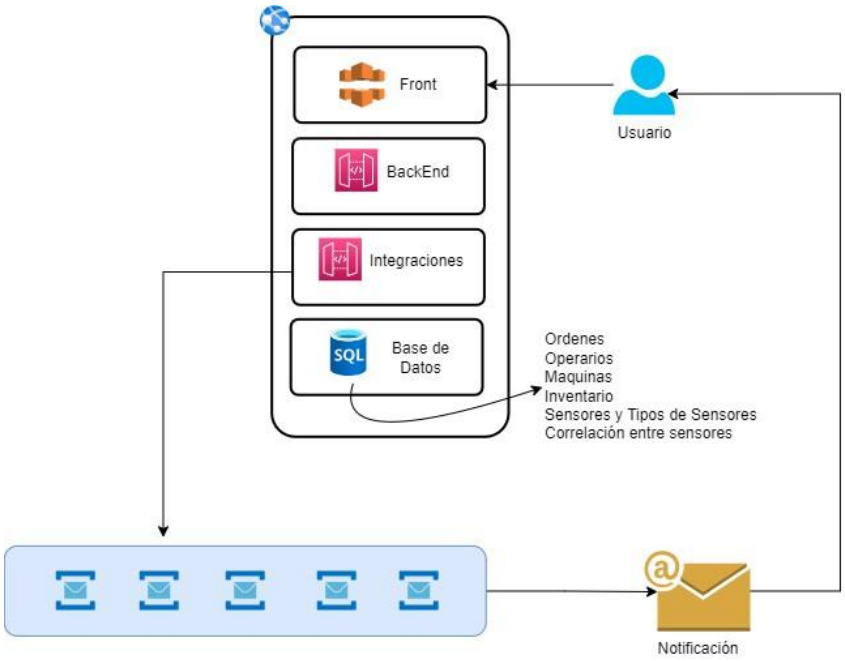
En conclusión, el estilo arquitectónico "Pipe and Filter" ofrece varias ventajas para dar solución a un problema en arquitectura de software. Algunas de estas ventajas son:

- Permite la reutilización de componentes, ya que los filtros pueden ser usados en diferentes contextos y combinados de distintas formas.
- Facilita el desarrollo paralelo y la distribución de la carga de trabajo, ya que los filtros pueden ser implementados y ejecutados de forma independiente.
- Mejora la modularidad y la mantenibilidad del sistema, ya que los filtros tienen una interfaz simple y bien definida, y se pueden modificar o reemplazar sin afectar al resto del sistema.
- Favorece la escalabilidad y el rendimiento del sistema, ya que los filtros pueden procesar los datos de forma concurrente y aprovechar los recursos disponibles.

Por estas razones, el estilo arquitectónico "Pipe and Filter" es una buena opción para resolver problemas que requieren un procesamiento secuencial y transformación de datos, como por ejemplo el caso de estudio.

2.3.3 Estilo Arquitectura Capas

ADR: Estilo Arquitectura Capas



- Estilo de la arquitectura Capas

La solución de software con la que interactuará el usuario tendrá una arquitectura de 4 capas:

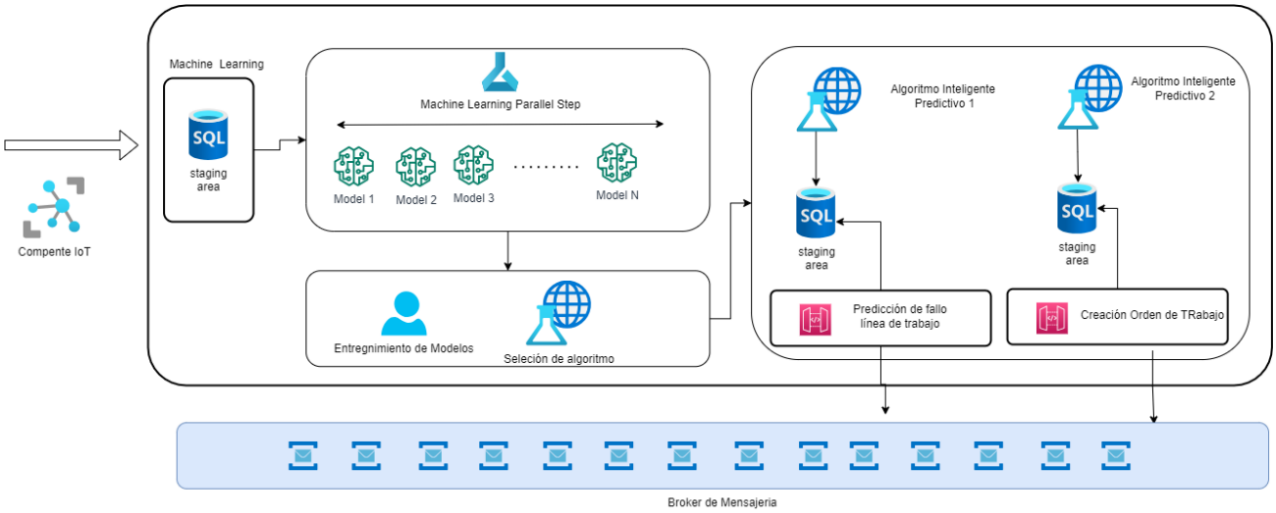
- Front: Capa dedicada a realizar representaciones gráficas en navegadores web para interacción directa con el usuario.
- Backend: Capa que contendrá toda la lógica de negocio requerida para el cumplir el objetivo de la solución.
- Integraciones: Capa que se servirá de disponibilizar la lógica de negocio e interactuar con el mediador de comunicación con el resto de la solución "Broker de Mensajeria"
- Base de datos: Repositorio de información

Se toma la decisión de implementar una vez consideradas sus ventajas

- Permite mejorar la modularidad, el mantenimiento y la reutilización al encapsular la lógica y los datos de cada capa.
- Facilita la adaptabilidad y la escalabilidad al permitir cambiar o reemplazar las implementaciones de cada capa según las necesidades.
- Favorece la seguridad y el rendimiento al distribuir las capas en diferentes niveles físicos y controlar el acceso a cada una.

2.3.4 Estilo Arquitectura Componente Machine Learning

ADR: 4. Estilo Arquitectura Componente Machine Learning



El componente Machine Learning, es implementado bajo el Estilo Pipes and Filter, donde mediante una secuencia de pasos y transformaciones se obtienen diferentes resultados así:

- El proceso de ingesta es realizado en componentes previos de la solución (Componente Cockpit).
- Estos son almacenados en una base de datos de staging
- Aquí pasan a una implementación de machine learning en donde se evalúan diferentes modelos genéricos de industria y personalizaciones específicas.
- Estos resultados son analizados y nuevamente procesados para elección y refinamiento del algoritmo.
- Una vez elegidos los algoritmos de predicción de fallos y optimización de ordenes de trabajo son publicados en el componente de procesamiento.
- Estos algoritmos son nuevamente alimentados con a data que llega en tiempo real a la base de datos de staging para la generación de resultados; todos estos resultado son entregados por medio una capa de integración a manejador de eventos para que sean consumidos por los demás componentes de la solución.

Es importante implementar arquitecturas de machine learning para obtener predicción sobre los datos de la organización. El machine learning es una técnica de inteligencia artificial que usa algoritmos matemáticos para crear modelos predictivos a partir de datos. Estos modelos se usan para realizar predicciones o tomar decisiones fundamentadas sobre nuevos datos.

Al implementar arquitecturas de machine learning en la organización, se pueden obtener beneficios como:

- Mejorar la eficiencia y la productividad de los procesos internos y externos.
- Optimizar el uso de los recursos y reducir los costes operativos.

- Aumentar la satisfacción y la fidelización de los clientes y usuarios.
- Generar ventajas competitivas y diferenciarse de la competencia.
- Innovar y crear nuevos productos y servicios.

Para implementar arquitecturas de machine learning, se requiere contar con un equipo multidisciplinar que tenga conocimientos de matemáticas, estadística, programación, ingeniería de datos y dominio del negocio. También se necesita una plataforma que facilite el desarrollo, el entrenamiento, la validación, la implementación y el monitoreo de los modelos de machine learning. Una opción es usar la plataforma Azure AI, que ofrece servicios integrados y escalables para el machine learning a escala.

En conclusión, implementar arquitecturas de machine learning es una forma de aprovechar el potencial de los datos para obtener predicción y mejorar el rendimiento de la organización.

2.3.5 Adquirir datos de los sensores

ADR: Adquisición de datos

En una primera iteración se pensó que los sensores al ser muchos necesitarían un estilo de event driven para poder comunicar sus datos. Sin embargo, revisando la literatura no es necesario tener múltiples suscriptores y en cambio la complejidad radica en las diferentes formas en las que los sensores entregan los datos. Es un problema de compatibilidad entre las tecnologías hardware del sensor y el software que hace la adquisición de la información.

La solución a este problema se encuentra en el estilo pipe and Filter. Esto permite tener una serie de módulos que adapten el tipo de comunicación, protocolo, incluso terminal física para poder hacer la adquisición de datos, guardarlos y consultarlos desde la base de datos.

En posteriores iteraciones se encontró que la cantidad de componentes del estilo podría aumentar para dar más prestaciones. Por ello se agregaron las ETLs y. el DataMart que permiten realizar analíticas sobre la data de los sensores con mayor libertad en la estructura y forma de la información. Finalmente, dada la complejidad de este estilo se optó por adquirir una solución en el mercado, esto incrementa costos, pero reduce tiempos de implementación agregando valor y soporte a la solución final.

2.3.6 Almacenar información de los sensores y los inventarios

ADR: Database

Para seleccionar la base de datos se tenía una restricción fuerte mencionada por el cliente en el cual mencionaba que la información de los sensores debe almacenarse en una base SQL. Sin embargo, las bases de datos relacionales, pese a que tienen ventajas para conservar la integridad de los datos, no son buenas para manejar multiples nodos de escritura y lectura.

Por ello en la segunda iteración de la selección de base de datos se consideró Cassandra. Pese a que es una base no SQL tiene una estructura de tablas que puede emular ser SQL y está diseñada para aplicaciones que necesiten tratar con grandes volumenes de datos. Los sensores IoT se caracterizan por producir mucha información constantemente y por eso se aceptó la base de datos Cassandra.

Se decidió finalmente tener un sistema hibrido de almacenamiento entre MySQL y Cassandra que permita tener los beneficios de ambas dependiendo la aplicación funcional o analítica que se le quiera dar.

2.3.7 Configuración de secuencia de sensores.

ADR: BackIoT

Para seleccionar el mejor back end para la configuración de la secuencia de los sensores se tenía una restricción fuerte asociada al tema de flexibilidad debido a la cantidad de diferentes sensores que hay. Por lo tanto, en la primera iteración se prefiere una clase construida desde 0 vs un componente llamado Azure IoT Central. Esto es debido a que la clase construida desde 0 es una nueva clase totalmente flexible y configurable a los intereses del negocio y a las particularidades de la problemática.

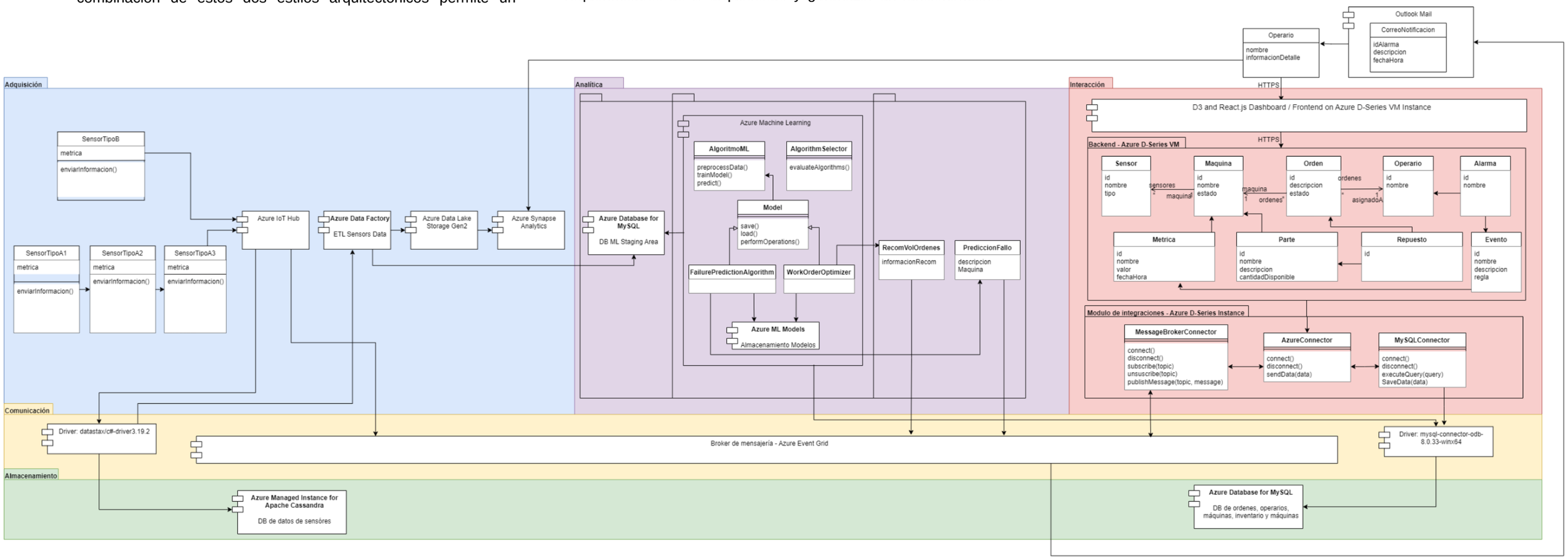
En la segunda iteración de la selección del back end para la configuración de la secuencia de los sensores se le agregaron a la clase nuevos atributos y métodos que entregan mayor información de valor para el usuario final. Cabe resaltar que la clase permite flexibilidad en la comunicación con componentes externos e incluir parámetros de seguridad para cada uno de los sensores.

3 Arquitectura

A continuación, se presenta el diagrama UML de la arquitectura planteada. Se puede observar que se diseñó una arquitectura con una mezcla entre los estilos Event-Driven y por capas. El enfoque Event-Driven se utiliza para la integración de los distintos componentes del sistema, permitiendo una comunicación asincrónica basada en eventos. Esto facilita la escalabilidad y la flexibilidad del sistema, ya que los componentes pueden responder dinámicamente a los eventos generados.

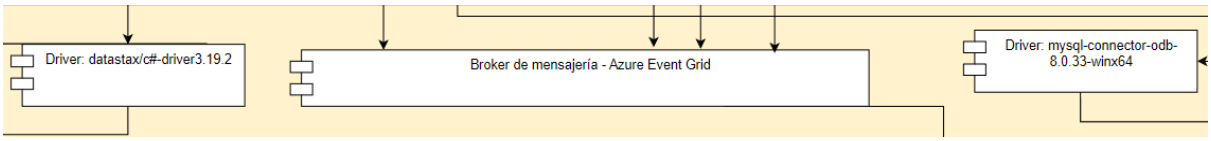
Por otro lado, la arquitectura por capas se emplea para organizar lógicamente los diferentes aspectos funcionales del sistema, como la adquisición de datos de las máquinas, los procesos de analítica, el almacenamiento transversal de datos, y la gestión de órdenes de trabajo y la interacción con los operarios. Esta separación en capas proporciona una estructura modular y facilita el mantenimiento y la evolución independiente de cada componente. En conjunto, la combinación de estos dos estilos arquitectónicos permite un sistema robusto, flexible y altamente escalable para el monitoreo y control del proceso productivo y de mantenimiento en la fábrica.

En general, se decidió utilizar múltiples componentes de Microsoft Azure para aprovechar la amplia gama de servicios y herramientas disponibles en la plataforma. Estos componentes proporcionan soluciones integradas y escalables para satisfacer los diversos requerimientos de la arquitectura y garantizar un funcionamiento eficiente del sistema. Al aprovechar las capacidades de Microsoft Azure, se logra una infraestructura robusta, confiable y altamente disponible para respaldar las funcionalidades de monitoreo, análisis, almacenamiento de datos, integración y comunicación del sistema de IoT en la fábrica. Para observar la imagen con mayor nitidez y detalle, esta se puede encontrar en la raíz del repositorio de GitHub

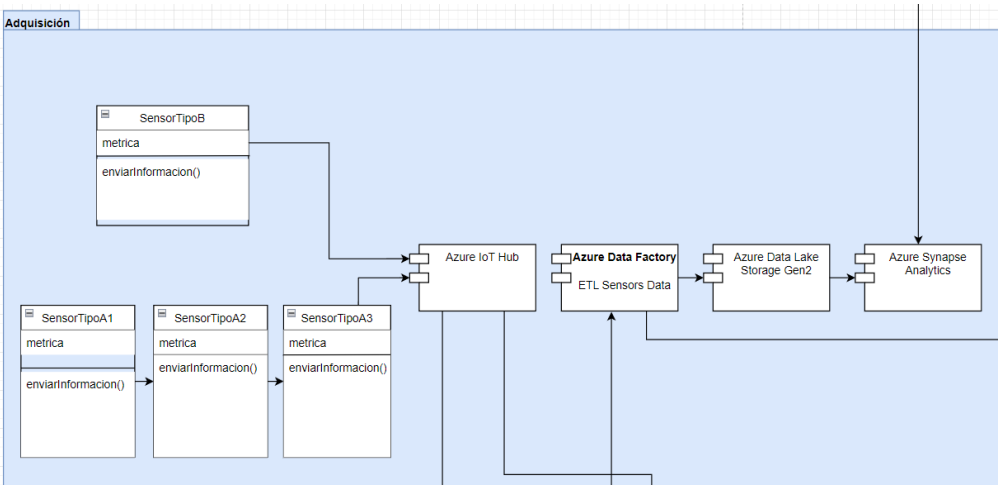


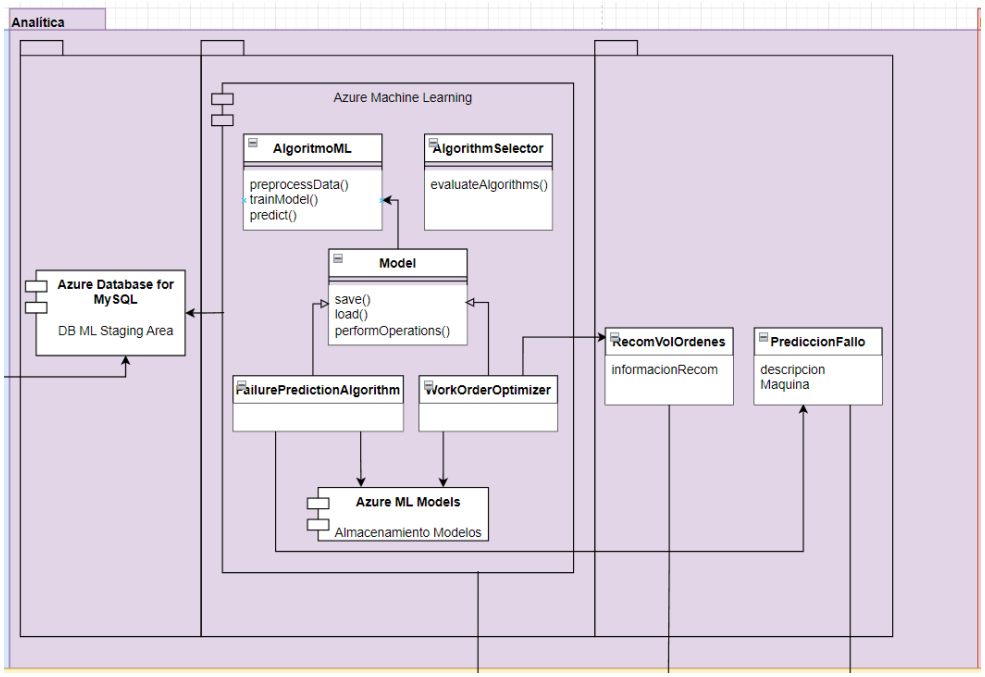
Se tienen 3 capas verticales principales: (1) almacenamiento, (2) Comunicación, y (3) Aplicación. En la capa de almacenamiento se tienen los componentes de bases de datos escogidos que son transversales a todo el sistema: Cassandra para datos de sensores y MySql para inventarios, operarios y ordenes, entre otra información relacionada.

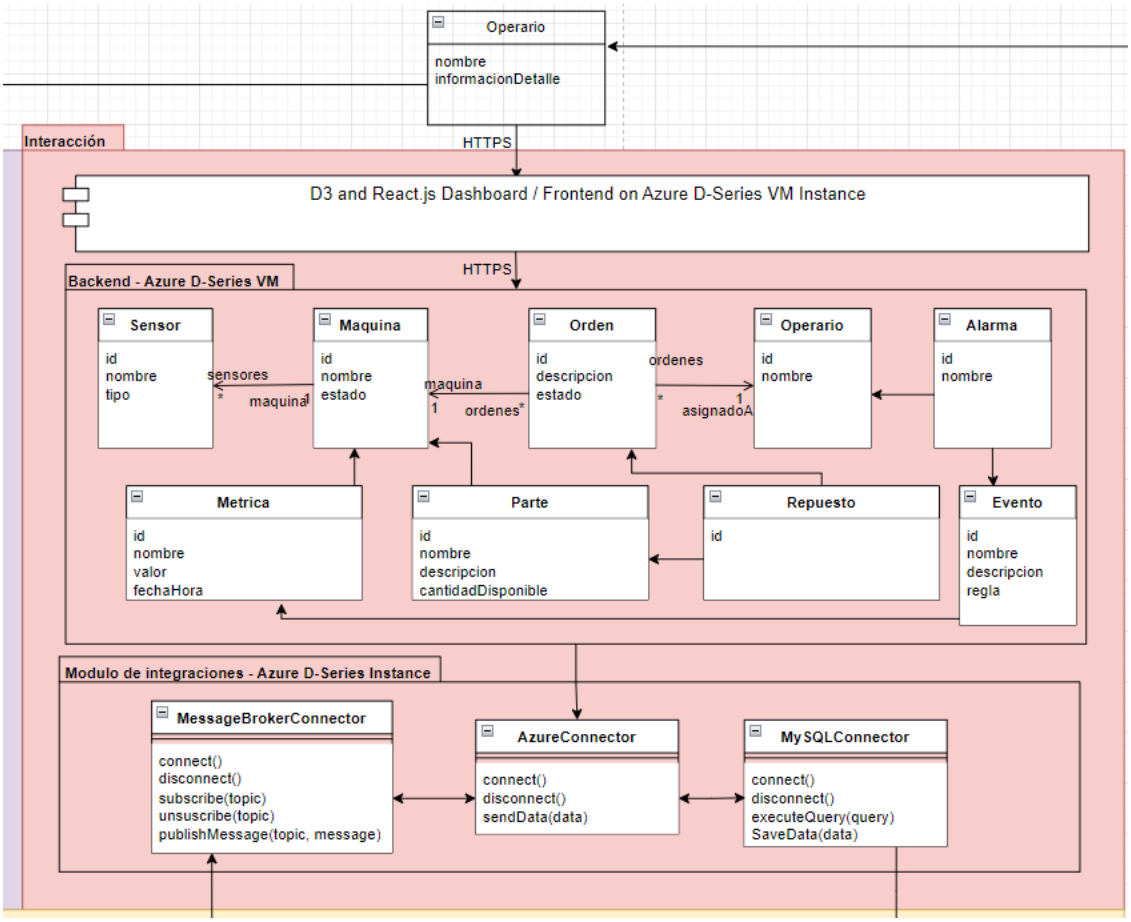
En la capa de comunicación se tienen los drivers que permiten la comunicación de los sistemas con las bases de datos descritas. Adicionalmente, se tiene el broker de mensajes, con el cual se implementa el patrón Event-Driven.



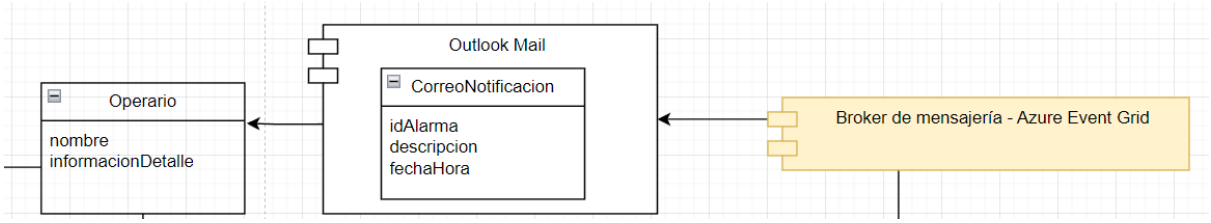
Finalmente, la capa superior está a su vez dividida en 3 capas horizontales: Aquisición, Analítica y Adquisición. En la primera capa, se tiene un paquete que sigue el patrón pipe and filter para obtener, transformar y almacenar los datos de los sensores de la fábrica. La segunda capa, contiene la lógica, almacenamiento y resultado de los algoritmos inteligentes predictivos. Finalmente, la última capa contiene una aplicación backend y frontend para la visualización e interacción con datos por parte de los operarios de la fábrica.







Adicionalmente, para mantener notificados a los operarios según las alarmas o suscripciones a eventos que estos tengan, el broker de mensajería se conecta con Outlook. De esta manera, al darse un evento que deba ser notificado, el operario será avisado mediante su correo.



4 Conclusiones

- Considerar diferentes opciones para un diseño permiten robustecerlo. Normalmente cuando una opción es suficientemente buena se detiene el proceso iterativo de diseño, pero tener un numero de fases mínimo obliga a considerar opciones que aportan a la decisión final una solución más robusta o al menos aportan consideraciones nuevas.

- Iterar sobre una misma solución permite que, aunque esta sea la correcta, considerar otras opciones que incrementen sus prestaciones, o que cambien completamente la solución (por ejemplo, de hacer un desarrollo in-house a adquirir un módulo de un proveedor externo).

- En conclusión, el estilo arquitectónico "Pipe and Filter" ofrece varias ventajas para dar solución a un problema en arquitectura de software. Este estilo permite descomponer una tarea compleja en una serie de componentes independientes que realizan una sola función. Estos componentes se conectan mediante tuberías que transmiten los datos en un formato estándar. De esta forma, se facilita la reutilización, la escalabilidad y la modificación de los componentes según los requisitos del problema. Además, este estilo es adecuado para los sistemas que procesan flujos de datos de forma continua y concurrente como es nuestra propuesta.

5 Bibliografía

Jena, S. A. (s.f.). *Architecture of Internet of Things (IoT)*. Obtenido de GeeksForGeeks.org: https://www.geeksforgeeks.org/architecture-of-internet-of-things-iot/

Microsoft. (2023). *Azure*. Obtenido de Azure: https://azure.microsoft.com/es-es/

Oracle. (2023). *MySQL*. Obtenido de MySQL: https://www.mysql.com/

Parashar, N. (6 de Julio de 2022). *Top 10 Open Source Cloud Computing Platform Databases*. Obtenido de Medium: https://medium.com/@niitwork0921/top-10-open-source-cloud-computing-platform-databases-6081f28ac55e

SQLite Consortium. (2023). *SQLite*. Obtenido de SQLite: https://www.sqlite.org/index.html

The Apache Software Foundation. (2023). *Cassandra: Open Source NoSQL Database*. Obtenido de Cassandra: https://cassandra.apache.org/_/index.html

Yumi Nakagawa, E., Oliveira Antonino, P., Schnicke, F., Capilla, R., Kuhn, T., & Liggesmeyer, P. (2021). Industry 4.0 reference architectures: State of the art and future trends. *Computers & Industrial Engineering, 156*(107241). doi:https://doi.org/10.1016/j.cie.2021.107241