UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

Informe de Memoria de Título para optar al título de

Ingeniero Civil en Telecomunicaciones

Arquitectura basada en microservicios para adquisición y procesamiento de datos de un sensor radiométrico de apoyo a la operación de fundiciones de cobre

por

Felipe Esteban Gasep Dueñas

Profesor patrocinante:

Dr. Sergio K. Sobarzo Guzmán

Concepción, Diciembre de 2024

Universidad de Concepción
Facultad de Ingeniería
Departamento de Ingeniería Eléctrica

Profesor Patrocinante:
Dr. Sergio K. Sobarzo Guzmán

Arquitectura basada en microservicios para adquisición y procesamiento de datos de un sensor radiométrico de apoyo a la operación de fundiciones de cobre

Felipe Esteban Gasep Dueñas

Informe de Memoria de Título
para optar al Título de

Ingeniero Civil en Telecomunicaciones

Diciembre, 2024

Resumen

La implementación de sistemas de adquisición de datos juega un papel de gran relevancia en entornos industriales, donde la recolección y análisis de mediciones es esencial, pues su obtención y monitoreo facilita la toma de decisiones informadas y la optimización de las operaciones. En el caso de las fundiciones de cobre, un sistema de sensor optoelectrónico especializado desempeña un rol crucial a la hora de entregar en tiempo real los indicadores clave de rendimiento (KPIs), proporcionando información valiosa sobre el rendimiento del proceso. No obstante, el diseño de este tipo de sistemas presenta desafíos técnicos importantes debido a la necesidad de procesar grandes volúmenes de datos de manera eficiente y garantizar su transmisión y almacenamiento seguro. Ante estos desafíos, se torna importante analizar y proponer arquitecturas que permitan manejar estos problemas de forma escalable y modular.

En el desarrollo de este trabajo se exploran distintas opciones de organización que puede adoptar un sistema informático, a la vez que se revisan dos importantes técnicas de comunicación interproceso, como lo son REST y RPC. Luego se analizan los principios de estructura del caso general de un sistema de adquisición de datos, para después profundizar en el caso particular de un sistema monolítico de sensor optoelectrónico ya en operación, diseñado específicamente para obtener KPIs basados en datos radiométricos provenientes de procesos de fundición. Con esto, se identifican los problemas específicos de la estructura del sistema y, posteriormente, se propone una arquitectura basada en microservicios que cumple con los requisitos necesarios para abordar dichos desafíos.

Para probar la eficacia del nuevo modelo, se finalizó con una prueba de concepto de un sistema modular simple de adquisición de datos que abarca las fases clave del proceso, en la cual se mide la distancia a un objeto mediante un sensor de ultrasonido controlado por Arduino, se transmiten los datos localmente usando gRPC y se despliegan en un gráfico que se actualiza en tiempo real durante un lapso del período de medición.

Índice

Índice de tablas

Índice de figuras

1. Introducción

En la era digital actual, donde la interconexión y el intercambio de información ya desempeñan un papel fundamental en nuestra vida cotidiana y son, también, una de las partes más importantes en el ámbito industrial, los sistemas distribuidos se han convertido en una parte esencial de la infraestructura tecnológica. Estos sistemas permiten la creación de entornos computacionales complejos y altamente eficientes que abordan los desafíos de escalabilidad y confiabilidad, que son principales constituyentes de los cimientos de la Industria 4.0.

Un sistema distribuido se refiere a una colección de computadores o componentes independientes que dan al usuario final la impresión de constituir un único sistema coherente [1]. A diferencia de los sistemas centralizados tradicionales, en los que una única máquina se encarga de todas las tareas, los sistemas distribuidos dividen las tareas en múltiples módulos de procesamiento que pueden ejecutarse en distintas máquinas, lo que permite operar de manera paralela y colaborativa y da la posibilidad de escalar sistemas previamente monolíticos.

Hoy en día, aplicaciones como Dropbox en el ámbito de la productividad o Netflix en el del entretenimiento, han adoptado una arquitectura enfocada en la modularidad [2, 3]. También lo han hecho empresas como eBay y Walmart en el sector comercial [4] y se ha incursionado en los beneficios que trae una migración de arquitectura como esta, tanto en un campo bancario con el sistema FX Core del Danske Bank [5, 6], como en el satelital con el programa de satélites fraccionados de DARPA [7].

En los últimos años y bajo el contexto de desarrollo de software industrial, se ha logrado idear, diseñar, construir y probar un prototipo de sensor optoelectrónico como instrumento altamente especializado para dar apoyo a las fundiciones de cobre, permitiendo contar en línea con KPIs de estimación de temperatura y ley de cobre.

A pesar de su buen funcionamiento, el producto presenta los problemas típicos de un sistema monolítico, como lo son un gran acoplamiento entre componentes y una escalabilidad limitada. Ante esto y la necesidad de elevarlo a una etapa comercial, en este trabajo se revisan las patrones arquitectónicos más importantes y dos de los principales estilos de comunicación usados en sistemas distribuidos. Esto, con el fin de proponer una nueva arquitectura general para el software de instrumentación científica subyacente, que esté preparada para la escalabilidad y que permita la ejecución sincronizada y segura de la serie de etapas en su cadena de datos.

2. Marco teórico

2.1. Ingeniería de software

El concepto de "sistema informático" ha sido una parte esencial del desarrollo de la informática desde sus inicios en la década de 1960. Durante este período, a medida que las primeras computadoras comenzaron a ser utilizadas más allá de los laboratorios y universidades, surgió la necesidad de estructurar y entender cómo estas máquinas procesaban la información.

No fue hasta 1968, en una conferencia organizada por la OTAN en Garmisch, que el término "ingeniería de software" fue acuñado en respuesta a lo que se denominó la "crisis del software". Esta crisis reflejaba los desafíos crecientes en el desarrollo de software debido a la complejidad cada vez mayor de los sistemas informáticos. Dicha conferencia marcó un punto de inflexión, donde la disciplina comenzó a ser tomada en serio, impulsando la regularización y estandarización de prácticas en el desarrollo de software [8].

A partir de este hito, los enfoques de la ingeniería de software y el desarrollo de aplicaciones han evolucionado significativamente. Ya no se limitan a simples procesos de programación, sino que han integrado nuevas metodologías y tecnologías para abordar la creciente complejidad de los sistemas. Una de las principales evoluciones ha sido el uso de sistemas distribuidos, un concepto que comenzó a tomar forma en los años 70. En esa época, surgió un interés creciente por transformar el enfoque de uso de recursos, pasando de la aplicabilidad, factibilidad y simplicidad hacia paradigmas más avanzados como los microservicios, el paralelismo y la computación distribuida.

Este cambio de paradigma trajo consigo nuevos desafíos, aumentando la complejidad tanto de los sistemas como de los problemas asociados, ante lo cual se han desarrollado las soluciones ingeniosas actuales que atacan estos problemas desde su raíz. Un ejemplo es el uso de contenedores, como Docker, y herramientas de orquestación, como Kubernetes, los cuales han transformado la forma en que se gestionan y despliegan las aplicaciones. Otro ejemplo sería la adopción de plataformas en la nube, el que ha revolucionado la capacidad para escalar y gestionar recursos, ofreciendo elasticidad y flexibilidad, y permitiendo a las organizaciones ajustar su infraestructura según las necesidades del sistema, reducir la carga de gestión y optimizar costos.

2.2. Sistemas monolíticos

Los sistemas monolíticos son una forma clásica de organización de software donde todos los componentes de una aplicación están integrados en un único ente de procesamiento. En este tipo de sistemas, todas las fases de una aplicación, las porciones de código, la lógica, la interfaz de usuario y la gestión de datos, están unidos en un solo bloque, funcionando como un ente indivisible. Esta arquitectura unificada permite una estrecha interrelación entre sus partes,

lo que facilita el desarrollo inicial y el despliegue. Sin embargo, al mismo tiempo, un estilo como este puede presentar desafíos significativos en términos de escalabilidad, mantenimiento y adaptación a nuevas tecnologías.

2.2.1. Arquitectura multicapas

Uno de los modelos más reconocidos y utilizados en la arquitectura de software es el modelo por capas. Surgido a principios de los años 70, ha sido ampliamente aplicado en diversas aplicaciones y sistemas de software debido a su naturaleza intuitiva y estructurada. Esta arquitectura se idea bajo la necesidad de separar los niveles de procesamiento dentro de un sistema, organizando el código en capas, cada una con roles específicos y objetos que interactúan únicamente dentro de su capa.

Las responsabilidades distribuidas dentro de cada capa están bien definidas y cada una se centra en sus funciones sin preocuparse por las de las otras capas. Los componentes de un nivel (j) dependen exclusivamente del comportamiento del nivel anterior $(j-1)$, siguiendo lo que se conoce como la Regla de Dependencia Incremental de Capas (ILD) [9].

Considerando el principio del modelo, cada una de estas capas debe estar diseñada de manera tal que las llamadas a recursos de capas anteriores sean evitadas y que los saltos de capas, en caso de ocurrir, estén bien definidos. Esto último lleva a referirse al tema de capas abiertas y capas cerradas. Tal como se muestra en la Figura 2.1, las peticiones de recursos se pueden realizar desde una capa superior a una inmediatamente inferior, para el caso de las capas cerradas, o pueden prescindir del paso por todas las capas intermedias y saltar directamente a la capa objetivo, para el caso de las capas abiertas.

Aunque esto puede ser beneficioso, pues evita la sobrecarga computacional debido a la comunicación innecesaria entre algunas capas, trae ciertos problemas a la hora de ordenar el equipo de trabajo y se vuelve imperioso tener una buena comunicación y una documentación clara. Es por esto que, generalmente, se sigue el principio de capas de aislamiento o capas aisladas, donde se opta por definir de una manera conveniente las tareas de cada capa, de manera que cambios en las capas intermedias no impacten en el funcionamiento del sistema y no se tenga una aplicación fuertemente acoplada.

En general, la arquitectura monolítica por capas es adecuada para sistemas pequeños a medianos, donde es prioridad la simplicidad y la facilidad de desarrollo. La organización inicial es lógica, lo que hace más sencilla la comprensión del sistema y el mantenimiento y actualización del código. Esto, sumado a que se tienen todos los componentes dentro de un único proceso principal, no es necesario lidiar con la complejidad de la comunicación entre servicios distribuidos. El traspaso de información entre capas tampoco incurre en latencias de red ni sobrecarga asociada a la serialización y deserialización de datos.

A pesar de que en un principio sea una buena opción trabajar con una arquitectura como

esta, a medida que la aplicación crece en tamaño y complejidad, comienzan a aparecer desafíos asociados a la escalabilidad, al rendimiento y a la dificultad de adoptar nuevas tecnologías. Mantener una separación clara entre la capas se vuelve más difícil, lo que puede llevar a un fuerte acoplamiento entre las diferentes partes del sistema y que cambios en algunas capas afecten al funcionamiento de las otras. Así, no se puede actualizar independientemente las capas de la aplicación sin afectar al resto del sistema y el crecimiento del sistema puede conducir a una "*big ball of mud*", donde entender y modificar el código se vuelve un gran inconveniente. Como resultado de lo anterior, conforme aumenta el tamaño del código base, los tiempos de compilación y carga pueden incrementarse considerablemente, afectando de manera negativa al desempeño general del sistema.

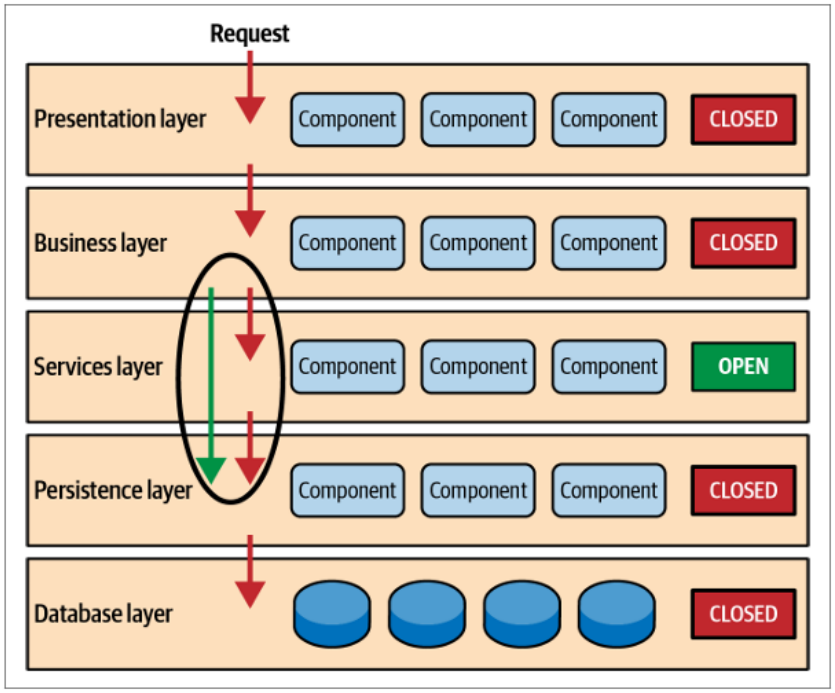


Figura 2.1: Flujo de una solicitud en un sistema organizado por capas abiertas y cerradas [10].

2.2.2. Arquitectura microkernel

En respuesta a los rápidos cambios que sufría el mundo computacional y de la ingeniería de software, la tecnología de microkernel se viene desarrollando desde mediados de los años 70 [11]. Situado en el espectro entre un sistema monolítico y uno distribuido, su filosofía se basa en tener un núcleo central que contenga todas las funcionalidades básicas y fundamentales para que un sistema opere, al cual se le agregan funcionalidades específicas en forma de *plug-ins* dependiendo del contexto y el uso que se le vaya a dar (Figura 2.2).

La manera en la que los *plug-ins* se ejecutan es como módulos *standalone*, independientes entre ellos, y se agregan o instalan en forma de librerías, archivos `.dll`, `.jar` u otro sistema de paquetes que se esté utilizando. Estos módulos no forman parte de los servicios básicos

que ofrece el sistema, sino que actúan como extensiones opcionales que pueden ser añadidas o removidas según las necesidades específicas de la aplicación, proporcionando un grado de flexibilidad y extensibilidad al sistema. Para que el sistema funcione de manera eficiente, es esencial que este conozca qué extensiones están disponibles, permitiendo así su correcta integración y administración.

A fin de establecer el traspaso de información entre los *plug-ins* y el núcleo, este último se encarga de gestionar la comunicación entre ellos mediante mecanismos de comunicación interproceso que pueden incluir *message passing*, en el que cada *plug-in* o servicio puede enviar y recibir mensajes que el microkernel encola y distribuye según corresponda; canales de comunicación, que corresponden a tuberías o sockets específicos que algunos microkernels implementan, con el propósito de tener una comunicación más estructurada y predecible; memoria compartida, en el caso de servicios que necesitan un alto rendimiento, pues permite que los *plug-ins* lean y escriban datos directamente en un área común, sin necesidad de copiar los datos a través de mensajes individuales; entre otros mecanismos específicos para cada diseño.

La arquitectura microkernel es especialmente adecuada para entornos donde la seguridad y la capacidad de adaptación son prioritarias [12]. Tiene la capacidad de aislar fallos y el hecho de permitir actualizaciones modulares lo hace ideal para sistemas que requieren un alto grado de confiabilidad. No obstante, como todo el procesamiento sigue ejecutándose en el sistema central, el incremento en la cantidad de servicios y en la complejidad del sistema puede aumentar las exigencias sobre la infraestructura a la hora de hacer las peticiones, afectando potencialmente el rendimiento del sistema con cuellos de botella y, por ende, limitando la escalabilidad.

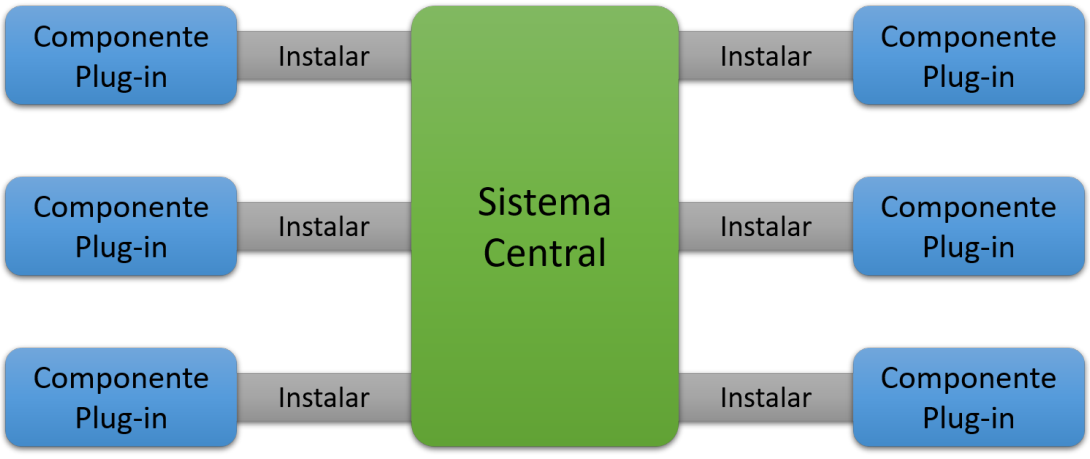


Figura 2.2: Estructura general de la arquitectura microkernel.

2.3. Sistemas distribuidos

Los sistemas distribuidos se caracterizan por dividir las funciones de una aplicación en múltiples componentes que se ejecutan en diferentes máquinas o nodos dentro de una red. A

diferencia de los sistemas monolíticos, donde todo el software reside en una sola unidad, en los sistemas distribuidos las tareas están repartidas entre varios procesos autónomos que cooperan para lograr un objetivo común. Esta organización permite una mayor escalabilidad y flexibilidad, ya que cada componente puede ser desarrollado, desplegado y mantenido de forma independiente. Sin embargo, esta independencia también introduce complejidades adicionales, como la necesidad de gestionar y sincronizar la transferencia de datos mediante diferentes tipos de comunicación entre los procesos.

2.3.1. Arquitectura dirigida por eventos

La arquitectura dirigida por eventos se encuentra dentro de los modelos más destacados en el diseño de sistemas distribuidos. Este enfoque, centrado en la asíncronía y la reactividad, se basa en la idea de que los componentes del sistema reaccionan a eventos que ocurren en su entorno o dentro de la aplicación. A diferencia de las arquitecturas tradicionales, donde el flujo de control es lineal y predecible, en un tipo de arquitectura como lo es la dirigida por eventos, el flujo va determinado por la ocurrencia y el manejo de eventos, lo que permite una mayor flexibilidad y adaptabilidad en sistemas dinámicos y altamente interactivos.

Este modelo se compone de productores de eventos, que son los que generan y emiten notificaciones sobre cambios de estado o sucesos relevantes, y consumidores de eventos, que están diseñados para responder a estos eventos de una manera específica. Entre ambos, se tiene un componente llamado *event broker*, el cual es un sistema de mensajería que actúa como intermediario, garantizando que dichos eventos sean entregados a los consumidores correspondientes.

Para implementar esta arquitectura, se suelen incorporar sistemas de mensajería robustos y confiables, como Apache Kafka, RabbitMQ, ActiveMQ, y AWS SQS. Estos sistemas actúan como *event brokers*, gestionando la publicación, enrutamiento y entrega de eventos entre los participantes. La elección del sistema de mensajería dependerá de los requisitos específicos del sistema, como la necesidad de alta disponibilidad, persistencia de mensajes, soporte para diferentes protocolos o la capacidad de escalar horizontalmente para manejar grandes volúmenes de eventos.

El enfoque que sigue esta arquitectura permite desacoplar los componentes del sistema, facilitando su escalabilidad, extensibilidad y mantenimiento. En ella, es posible agregar distintos servicios o componentes de manera flexible, ya que los productores y consumidores de eventos no están directamente acoplados entre sí. Este desacoplamiento permite que los servicios puedan ser actualizados, reemplazados, o escalados de forma independiente, sin interrumpir el funcionamiento general del sistema.

Un ejemplo de adopción de este planteamiento es ilustrado por el esquema de LISA (Lightweight Infrastructure for Service Automation) de la Figura 2.3, en el que se ve cómo

esta herramienta de automatización de infraestructura constituye una arquitectura dirigida por eventos, preparada para gestionar y orquestar servicios en respuesta a eventos generados en el entorno.

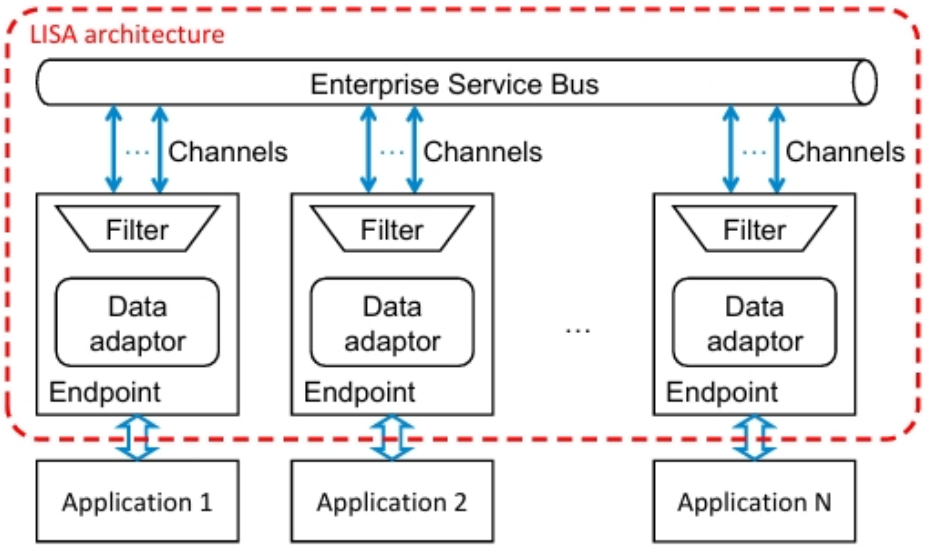


Figura 2.3: Vista general de la arquitectura de comunicación de LISA [13].

2.3.2. Arquitectura de microservicios

La arquitectura de microservicios se ha convertido en un enfoque popular para diseñar sistemas distribuidos. En contraste con cualquier arquitectura clásica, donde todas las funcionalidades de la aplicación están empaquetadas en un solo bloque de código, la arquitectura de microservicios descompone la aplicación en múltiples servicios pequeños e independientes. Cada uno de estos microservicios se encarga de una función específica dentro de la aplicación y se comunica con otros servicios a través de *endpoints* o puntos de contacto bien definidos por una *API gateway* (Figura 2.4).

La *API gateway* actúa como un intermediario que enruta las solicitudes a los microservicios correspondientes, gestiona la autenticación y autorización, y puede realizar tareas adicionales como balanceo de carga o transformación de datos para asegurar la compatibilidad entre los componentes del sistema. Al centralizar estas funciones en la *API gateway*, se simplifica la comunicación entre los microservicios y se ofrece un punto único de entrada para las solicitudes del cliente. Esto permite que cada microservicio se enfoque en su propia lógica y no necesite preocuparse por las complejidades asociadas a la transferencia de información a lo largo del sistema.

La principal ventaja de este enfoque radica en la independencia de los microservicios. Cada servicio puede ser desarrollado, desplegado y escalado de manera autónoma, lo que permite a los equipos de desarrollo trabajar en diferentes partes de la aplicación simultáneamente sin

interferir en el progreso de los demás. Este grado de mantenimiento no solo facilita el desarrollo a nivel social, sino que también abre las puertas a la adopción de nuevas tecnologías, ya que cada microservicio puede ser implementado en el lenguaje o plataforma que mejor se adapte a sus necesidades, sin afectar al resto del sistema.

Sin embargo, la arquitectura de microservicios también presenta desafíos. La gestión de múltiples servicios distribuidos introduce una mayor complejidad en la comunicación y la orquestación de los componentes. Se requiere una infraestructura sólida para manejar el descubrimiento de servicios, el balanceo de carga, la tolerancia a fallos y la seguridad. Además, la latencia y la sobrecarga de red pueden convertirse en problemas críticos, especialmente en sistemas donde los microservicios necesitan interactuar con frecuencia. Ante esto, se vuelve imperioso usar técnicas de comunicación interproceso que estén acondicionadas para desafíos como estos.

A pesar de esto, la arquitectura de microservicios es altamente valorada en entornos donde la agilidad y la escalabilidad son prioridades. Tiene una gran capacidad para adaptarse a cambios y evolucionar frente a las demandas operacionales del contexto en el que se usa, lo que la hace ideal para entornos dinámicos. Al permitir un desarrollo y despliegue más ágil, los microservicios ofrecen una solución robusta para la construcción de sistemas distribuidos que requieren una alta disponibilidad, rendimiento y capacidad de adaptación.

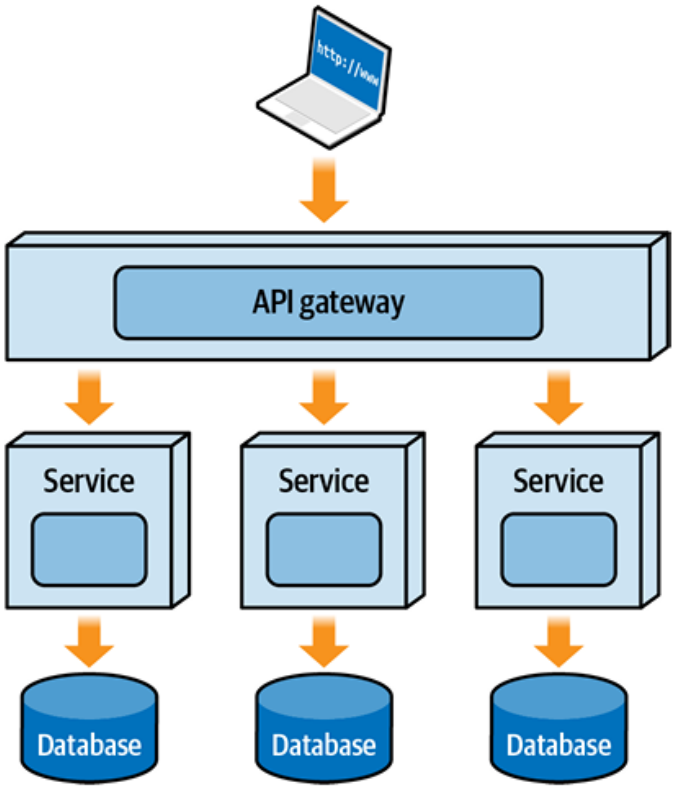


Figura 2.4: Topología básica de una arquitectura de microservicios [10].

2.4. Técnicas de comunicación interproceso

Las técnicas de comunicación interproceso son fundamentales para permitir que los distintos componentes o servicios de un sistema interactúen y se traspasen la información. Estas técnicas facilitan la transferencia de datos y la coordinación entre procesos que pueden estar en la misma máquina o distribuidos en diferentes ubicaciones a través de una red. Además, pueden variar en complejidad y enfoque, abarcando desde mecanismos de intercambio de mensajes hasta sofisticados métodos basados en protocolos y formatos de datos específicos. Entre las opciones más comunes se encuentran el enfoque de Representational State Transfer (REST) y las llamadas a procedimiento remoto (RPC).

2.4.1. REST

REST es usado como un estilo de comunicación para diseñar servicios web, que permite la comunicación entre clientes y servidores a través del protocolo HTTP. En un sistema RESTful, las operaciones se realizan utilizando los métodos HTTP estándar: GET, POST, PUT y DELETE. Estos métodos corresponden a las operaciones de lectura, creación, actualización y eliminación de recursos en el servidor, respectivamente. Los recursos se representan generalmente en formato JSON o XML, y cada recurso se identifica mediante una URI (Uniform Resource Identifier) única. Una interacción de este estilo es el que se ve en la Figura 2.5.

En REST, los recursos se definen en un formato estandarizado y pueden ser accedidos a través de los métodos HTTP. Los clientes envían solicitudes al servidor especificando la URI del recurso deseado y el método HTTP que corresponde a la operación que desean realizar. El servidor procesa la petición, realiza la operación en cuestión y devuelve una respuesta con el estado del recurso solicitado. Este enfoque basado en recursos y métodos HTTP proporciona una comunicación clara entre el cliente y el servidor.

Una de las ventajas clave de REST es su compatibilidad con el protocolo HTTP/1.1, que es ampliamente soportado por los navegadores y servidores web. REST no requiere una capa adicional de transporte o serialización, ya que utiliza el propio protocolo HTTP para la comunicación, lo que simplifica su implementación e integración. Los datos en REST pueden ser enviados en diferentes formatos, como XML y JSON, siendo este último el más común debido a su ligereza y facilidad de uso en aplicaciones web.

2.4.2. RPC (gRPC)

Las RPC son una técnica de comunicación interproceso que permite a un programa ejecutar funciones o procesos en otro programa que puede estar en una máquina diferente. En una RPC, el programa cliente envía una solicitud al programa servidor para ejecutar una función específica. Cuando el servidor recibe esta solicitud, ejecuta la función solicitada y luego envía la respuesta

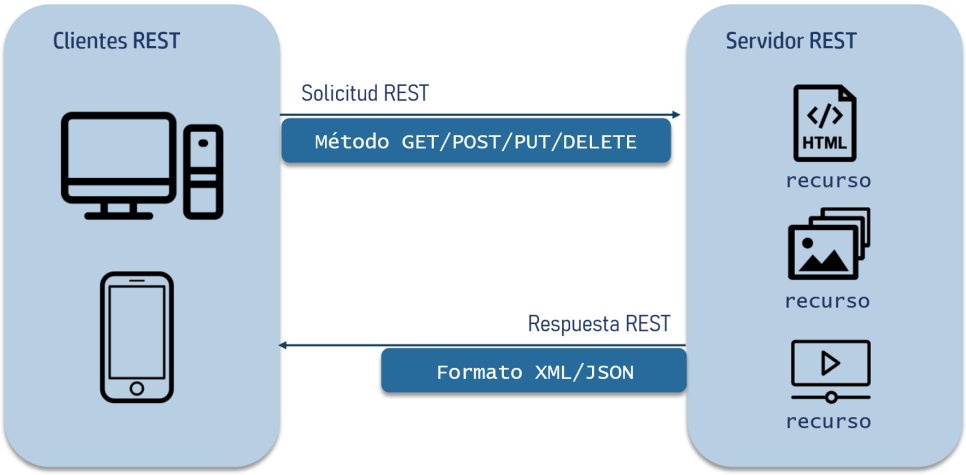


Figura 2.5: Interacción entre un cliente y un servidor REST.

de vuelta al cliente. Este proceso es transparente para el desarrollador o el usuario, ya que es similar a hacer una llamada a una función local, pero involucra la transmisión de datos a través de una red.

Las RPC son ampliamente utilizadas en sistemas distribuidos para permitir la comunicación entre componentes que están distribuidos en diferentes ubicaciones. Una implementación moderna de esto es la desarrollada por Google en 2016, llamada gRPC, la cual proporciona un framework eficiente y flexible para construir servicios RPC utilizando HTTP/2 y Protocol Buffers.

El uso del protocolo HTTP/2 para la transmisión de datos ofrece ventajas significativas sobre su versión anterior HTTP/1.1, como la multiplexación de solicitudes y respuestas sobre una sola conexión, la compresión de encabezados y la capacidad de mantener conexiones persistentes. Esto mejora la eficiencia y el rendimiento en la comunicación entre cliente y servidor.

Por otro lado, Protocol Buffers es un sistema de serialización de datos que utiliza un formato binario compacto y eficiente para representar estructuras de datos. Este formato ocupa menos espacio en comparación con el texto usado en otros formatos como JSON o XML, lo que resulta en menos ancho de banda utilizado y tiempos de transferencia más rápidos. Con esto, gRPC está preparado para tolerar grandes cargas de datos y levantar sistemas de alto rendimiento.

El funcionamiento general de gRPC se basa en distribuir a todos los componentes del sistema un archivo de esquema `.proto`, donde se describen la estructura de los datos y los servicios de manera abstracta, y se utilizan para generar automáticamente el código necesario en varios lenguajes. El código generado maneja la conversión entre el formato binario y los objetos de datos en el lenguaje de programación correspondiente, promoviendo totalmente la independencia de los módulos involucrados y permitiendo una comunicación agnóstica con respecto al lenguaje, tal como se ve en la Figura 2.6.

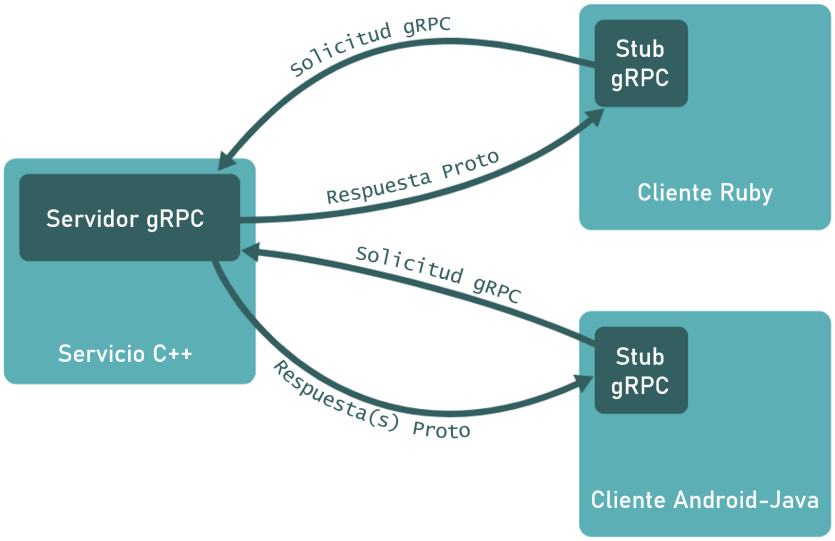


Figura 2.6: Ejemplificación del agnosticismo de gRPC frente al lenguaje de programación.

3. Definición del problema

3.1. Hipótesis de Trabajo

El cambio de arquitectura de un sistema de adquisición de datos de procesos industriales, desde una monolítica a una distribuida, permitirá una mayor adaptabilidad general a nuevas tecnologías y requerimientos operativos.

3.2. Objetivos

3.2.1. Objetivo General

Diseñar y proponer una arquitectura de un sistema de adquisición de datos basada en microservicios, mediante la implementación de un sistema de prueba que permita la distribución segura de la información y admita la ejecución independiente de cada una de las etapas de la cadena de datos.

3.2.2. Objetivos Específicos

- Explorar y estudiar distintos tipos de arquitectura y de comunicación interproceso dentro de un sistema de software.
- Analizar un sistema de adquisición de datos radiométricos operativo.
- Proponer un rediseño de la arquitectura de software del sistema.
- Realizar una prueba de concepto con un sistema de adquisición de datos propio.

3.3. Metodología

El trabajo comienza con un estudio teórico de los tipos más relevantes de estructura interna de un sistema de software, explorando las técnicas de comunicación interproceso entre sus componentes clave y las características que ofrece cada una.

Luego, se explica el principio común que siguen los sistemas de adquisición de datos de procesos y se compara con un caso de estudio, que corresponde a un sistema optoelectrónico desarrollado en la Universidad de Concepción enfocado a la obtención de datos radiométricos de fundiciones de cobre. Dicho sistema se desglosa, reconociendo sus elementos principales relacionados con las fases típicas de un sistema como estos.

A través de documentación y conversaciones con el equipo de trabajo encargado del sistema, se identifican los problemas que presenta la adopción del tipo de arquitectura actual de la aplicación y se establecen requerimientos para establecer una nueva estructura que se adecúe

a ellos. Con esto, se propone la nueva arquitectura basada en microservicios que desacopla la interdependencia de los elementos internos.

Finalmente, la propuesta se verifica con un sistema de prueba de medición de distancia con Arduino, empleando un modelo cliente-servidor que contiene dos microservicios constituyentes de las fases esenciales de la cadena de datos, capturándolos y calculando la variable de interés, y enviándolos a través de gRPC para desplegarlos en una interfaz gráfica que muestra un gráfico distancia vs. tiempo.

4. Levantamiento del sistema

4.1. Contexto

La minería del cobre es una industria de indiscutible relevancia, tanto a nivel mundial como nacional. Es un pilar fundamental de sectores como la manufactura, la construcción y la tecnología, ya que proporciona la materia prima para crear una amplia gama de productos. En el año 2021, por ejemplo, la producción de cobre a nivel mundial alcanzó las 21 millones de toneladas métricas, de las cuales 5.588.084 fueron producidas por Chile, equivalente al 26,6 % del total. Esto posiciona al cobre como el mineral metálico más importante del mundo, solamente antecedido por el hierro, cuya producción fue de 2.550 millones de toneladas métricas [14].

El proceso de agregación de valor en la producción del cobre comienza con la extracción del mineral desde los yacimientos, pasa por la extracción de impurezas y termina con la refinación del material para la producción de planchas o cátodos de cobre con alto grado de pureza. Dentro de esta línea de eventos, existe una sección llamada pirometalurgia, que podría considerarse como el punto de inflexión entre el proceso minero y el de la manufactura. Es la fase que corresponde a todo el procesamiento fisicoquímico enfocado a la refinación del cobre. Esto, a su vez, se divide en tres subetapas denominadas fusión, conversión y refinación, las que en conjunto suman una gran cantidad de elementos, factores y parámetros de funcionamiento a evaluar.

Dado que en este proceso la mayoría de las variables operacionales se miden ex post, es decir, una vez que algún proceso o evento que se está monitoreando ya ha terminado, los resultados pueden tener retardos de varias horas o incluso días, lo que lleva a que la información recopilada sirva muchas veces solo de registro estadístico y no sea útil para el seguimiento y control en línea de estos procesos. Muchas de las decisiones asociadas a ellos que implican acciones inmediatas o a corto plazo, se basan en la experiencia de los operarios de planta y del criterio que se han formado para evaluar el estado del proceso productivo, lo cual carece de estandarización y afecta a la estabilidad del proceso. Además, las condiciones industriales en las que se encuentran son bastante desafiantes, lo que dificulta el desarrollo e integración de instrumentación que apoye la medición y regulación de todas estas variables implicadas.

Motivado por las limitaciones y la necesidad de mejorar el monitoreo y el control en esta última etapa del proceso minero, tanto en Chile como en el extranjero, se ha formado una empresa de base tecnológica universitaria (EBTU) que ha diseñado y puesto en marcha un sensor optoelectrónico específicamente dedicado a asistir en estas tareas. A través de experiencias en fundiciones reales de renombre mundial, el sistema ha demostrado ser capaz de cumplir con toda la cadena de datos necesaria y ahora se encuentra en proceso de elevar su madurez tecnológica para contar con características más específicas, aumentar su seguridad y simplificar el uso de esta herramienta.

4.2. Caso general

Cuando se habla de sensores, se está contemplando la combinación de dos elementos: un *sensing element*, que es la parte del sensor que responde directamente a la magnitud que se está midiendo, ya sea luz, temperatura, presión, humedad, entre otros; y un *transductor*, que es el dispositivo que convierte la respuesta del primero en una señal utilizable, como lo puede ser una señal eléctrica u óptica.

La medición de parámetros de estos sensores tiene aplicaciones en muchos dispositivos electrónicos y es útil en una gran variedad de campos. Sin embargo, para los procesos involucrados en un contexto industrial más amplio, esto no es suficiente. A raíz de ello y a lo largo del tiempo, se han ido desarrollando soluciones con un nivel de complejidad mayor, los cuales son capaces de llevar a cabo no solo la obtención de información, sino que también incorporan una gestión y un control integral de ella.

En general, los softwares actuales de los sistemas de sensores son bastante diversos y sofisticados, pero todos comparten una misma estructura y basan su funcionamiento en un mismo principio, lo cual se puede dividir en cuatro fases clave en su cadena de datos: adquisición, procesamiento, almacenamiento y visualización o despliegue.

- **Adquisición de datos:** En esta primera fase, los sistemas recolectan datos brutos del entorno a través de los sensores y los convierten en señales digitales que pueden ser leídas y usadas en la siguiente fase. Para realizar esto, los sistemas de adquisición de datos (DAQ) emplean el uso de conversores análogo-digital (ADC), componente electrónico responsable de muestrear la señal analógica proveniente del fenómeno físico que se está midiendo, cuantificarla y codificarla en un formato binario que puede ser procesado por un sistema informático (Figura 4.1).

  A menudo los equipos de sensado incluyen técnicas de acondicionamiento de la señal previo ingreso al ADC, las que preparan de mejor manera la señal y ayudan al posterior procesamiento. Dependiendo de la aplicación, algunas de las técnicas más comunes son: amplificación o atenuación de la señal, filtrado o aislamiento de ella para eliminar ruido, y desplazamientos o conversiones de nivel para ajustarla a algún valor de referencia.

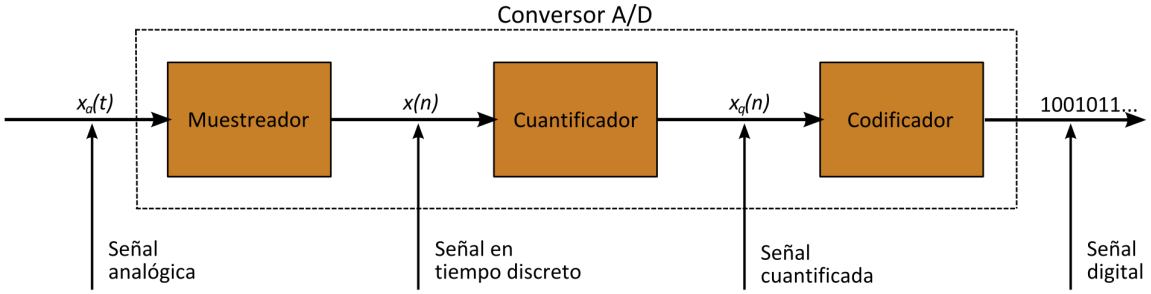


Figura 4.1: Diagrama de bloques general de un ADC.

Finalmente, una vez que la señal ya fue digitalizada, debe ser enviada al equipo central para continuar con las siguientes fases del sistema. La transmisión puede hacerse por medio de varios métodos y protocolos, dependiendo, de igual manera, de las necesidades particulares de la aplicación, de los equipos que se estén usando o de las condiciones del entorno en el que se está trabajando. Entre ellos están: USB, Wi-Fi, Ethernet, Bluetooth, RS-232 y protocolos industriales como Profibus, Modbus, EtherCAT, OPC o DNP3.

- **Procesamiento de datos:** Esta siguiente fase implica transformar los datos brutos en información de interés. Esto cambia enormemente de campo en campo, pudiendo ir desde procesamientos simples, como algunos cálculos de datos estadísticos o clasificación y orden de los datos según algún criterio, hasta la estimación de un comportamiento mediante la implementación de modelos predictivos basados en inteligencia artificial, la reconstrucción de modelos 3D a partir de un conjunto de imágenes, o el armado de redes y optimización de rutas para el tráfico en Internet.

  En el contexto ingenieril, todo el tratamiento para cumplir dichos procesamientos es apoyado, principalmente, por programas y sets de recursos específicamente desarrollados por el fabricante de los equipos de medición. Usualmente, estos fabricantes proveen una aplicación que es capaz de calibrar y sincronizar los instrumentos para la adquisición de distintos parámetros de interés, de utilizar filtros avanzados para el mejoramiento de la señal, de efectuar interpolación para la generación de modelos, y de aplicar transformadas y múltiples técnicas para reconocimiento de patrones e identificación de errores o anomalías, entre muchas otras características. Un ejemplo de mejoramiento de señal es el usado en el procesamiento de imágenes mostrado en la Figura 4.2, donde se usa la técnica de ecualización de histograma para obtener una distribución más uniforme y aumentar el contraste de la imagen.

  En función del área en el que se trabaja, puede ser necesario emitir archivos de registro,

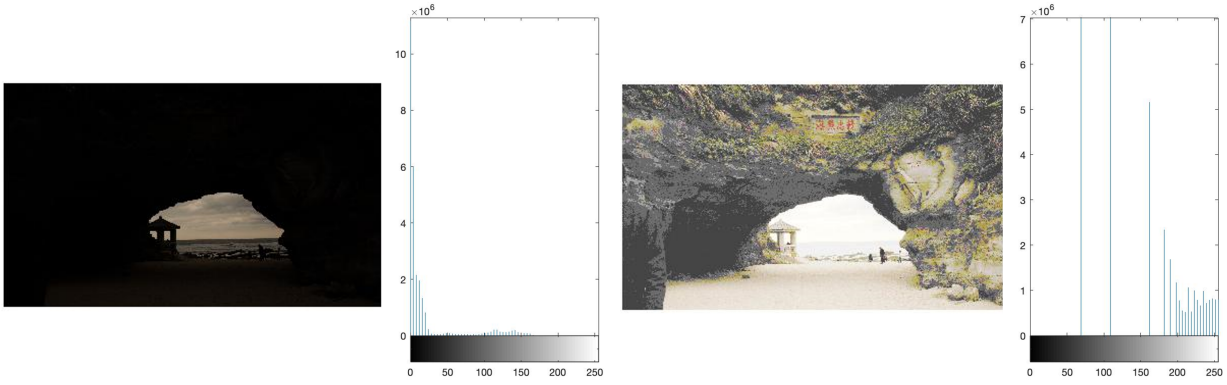


Figura 4.2: Ejemplo de mejoramiento de imagen mediante ecualización de histograma.

guardar la información procesada en formatos o estructuras concretas, o incluso trasladarla a otro ente de procesamiento para tratamiento y análisis adicional. Acá es donde se da paso a la siguiente fase de guardado.

- **Almacenamiento de datos:** Para muchos procesos, el almacenamiento de los datos es una parte crucial, pues conlleva guardar el historial de manera segura y eficiente, garantizando que este esté disponible cuando sea necesario.

  Existen opciones locales, como discos duros, SSDs o servidores internos; remotas, como lo es el almacenamiento en la nube; y mixtas, donde se combina almacenamiento local con sincronización en la nube para asegurar accesibilidad y redundancia. Cada una tiene ventajas sobre otra según el uso para el cual se requiere.

  Ya sea en la etapa de procesamiento o luego de ella, la estructuración de los datos juega un papel importante en el guardado. Para permitir que el intercambio de información fluya entre diferentes sistemas y aplicaciones, se usan formatos como XML, CSV, JSON, Protobuf, HDF5, entre otros, y se decide entre el uso de bases de datos relacionales (SQL) o no relacionales (NoSQL). La elección puede impactar significativamente en el desempeño del sistema y se hace bajo criterios que se adecúen al enfoque de trabajo, la naturaleza de los datos, los volúmenes que se están trabajando y la frecuencia de generación de ellos.

  Con respecto a la seguridad, las organizaciones que usan los sistemas de sensores precisan de una confidencialidad e integridad de sus datos. Ante esto, las bases de datos involucradas, que pueden ser propias de estas organizaciones o desarrolladas por terceros, proveen estrategias y tecnologías que contrarrestan accesos no autorizados, manipulaciones no deseadas y ataques informáticos. Algunas medidas destacadas son las que se muestran en la Figura 4.3, entre las que se encuentran distintos tipos de autenticación para verificar qué usuarios pueden acceder a la base de datos; el control de acceso basado en roles, que define privilegios basados en el rol del usuario dentro del sistema; la auditoría y

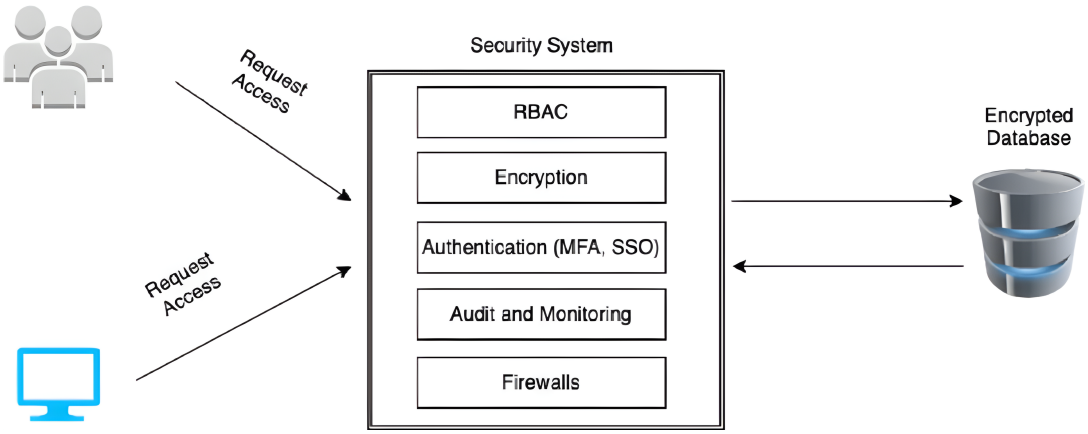


Figura 4.3: Modelo de protección general de una base de datos [15].

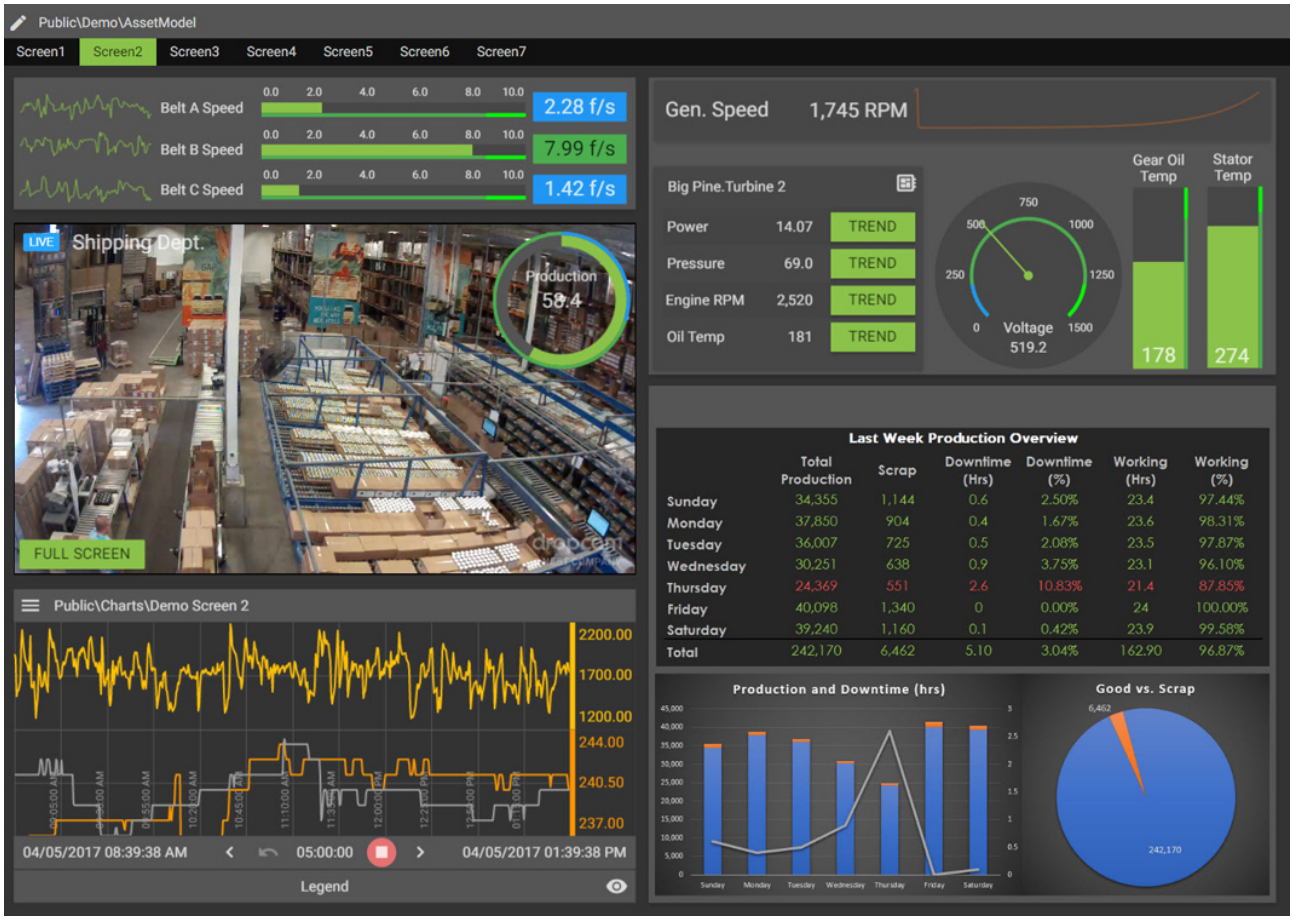


Figura 4.4: GUI de software de monitoreo de una línea de producción [16].

monitoreo, donde se registran todas las acciones realizadas dentro de la base de datos, lo que ayuda a detectar actividades sospechosas; el uso de *firewalls*, que asegura que solo los sistemas autorizados puedan entablar una comunicación; y la encriptación de todas las comunicaciones, protegiendo la transferencia de datos mediante el uso de protocolos como TLS/SSL.

- **Visualización de los datos:** La etapa de visualización permite que los operarios y analistas interpreten rápidamente la información de los procesos. Habitualmente, las aplicaciones cuentan con un sistema que captura los datos de puntos de interés particulares dentro de la cadena de datos y los exhibe en forma de paneles interactivos, los cuales incluyen gráficos y tablas con indicadores varios, y pueden alertar y generar informes sobre valores que caigan fuera de un rango preestablecido.

  Además de la capacidad de visualizar los datos en las pantallas de una oficina o en la misma planta industrial donde está operando el sistema, se ofrece también la opción de mostrar el contenido en dispositivos inteligentes de manera remota a través de la red. Esto es particularmente útil para equipos que requieren movilidad o en situaciones en las que se necesita tomar decisiones rápidas fuera del entorno habitual de trabajo.

Al igual que con las bases de datos, es posible que se necesite otro tipo de servicios más específicos, por lo que es viable incluir la participación de programas de terceros dedicados a eso. Estos proporcionan sus propios subsistemas de captura directa desde los sensores, sistemas SCADA, PLCs y otros dispositivos industriales, así como también sus propias APIs, con las que facilitan la interoperabilidad y la personalización de lo expuesto en los paneles. Un ejemplo de interfaz gráfica de usuario (GUI) es la de la Figura 4.4.

Ejemplos destacados de soluciones industriales robustas que tienen una estructura como esta y están relacionados con el uso de cámaras y/o sensores para el manejo del ciclo de vida completo de los datos obtenidos, son ThorImage OCT de Thorlabs y OceanView de Ocean Optics.

ThorImageOCT permite la adquisición de datos en 1D, 2D y 3D desde sistemas de Tomografía de Coherencia Óptica (OCT), lo que incluye la captura de imágenes y datos brutos desde los sensores. También incluye técnicas avanzadas de procesamiento, como Angiografía de Coherencia Óptica y Doppler OCT, permitiendo transformar esos datos para otros tipos de análisis. Luego, soporta la exportación y almacenamiento en varios formatos específicos como VTK, VFF, FITS, y RAW, asegurando la conservación de los datos para análisis posteriores. Por último, proporciona una interfaz gráfica completa para la visualización de los datos adquiridos y procesados, con opciones para ajustar contraste, brillo, y diferentes modos de visualización (seccional, superficial, y renderizado volumétrico en 3D) [17].

Por otro lado, OceanView admite la captura y registro de datos espectrales desde espectrómetros y sensores ópticos de Ocean Optics, con opciones para configuraciones específicas como tiempos de integración y calibración. En cuanto al procesamiento, se tienen herramientas que permiten aplicar correcciones y calibraciones a los datos, y realizar análisis espectrales avanzados como el ajuste de curvas, cálculo de concentraciones, y análisis de tendencias. El software guarda los datos en archivos de texto y formatos binarios específicos, y soporta la exportación en otros formatos para su uso en aplicaciones externas. Similar al caso anterior, OceanView deja a disposición el uso de una interfaz gráfica personalizable con diferentes modos de exhibición, para la visualización en tiempo real de los datos espectrales, comparación de espectros y otros estudios afines [18].

4.3. Caso de estudio

La EBTU en cuestión ha enfocado el desarrollo de su sensor optoelectrónico y el software asociado, a la obtención de datos provenientes de fundiciones de cobre, al cálculo de KPIs en base a ellos y a la creación de modelos predictivos para la estimación de ley de cobre. A pesar de su especialización, el sistema sigue los mismos principios comunes de los sistemas de sensores expuestos anteriormente, adoptando la organización que se muestra en la Figura 4.5.

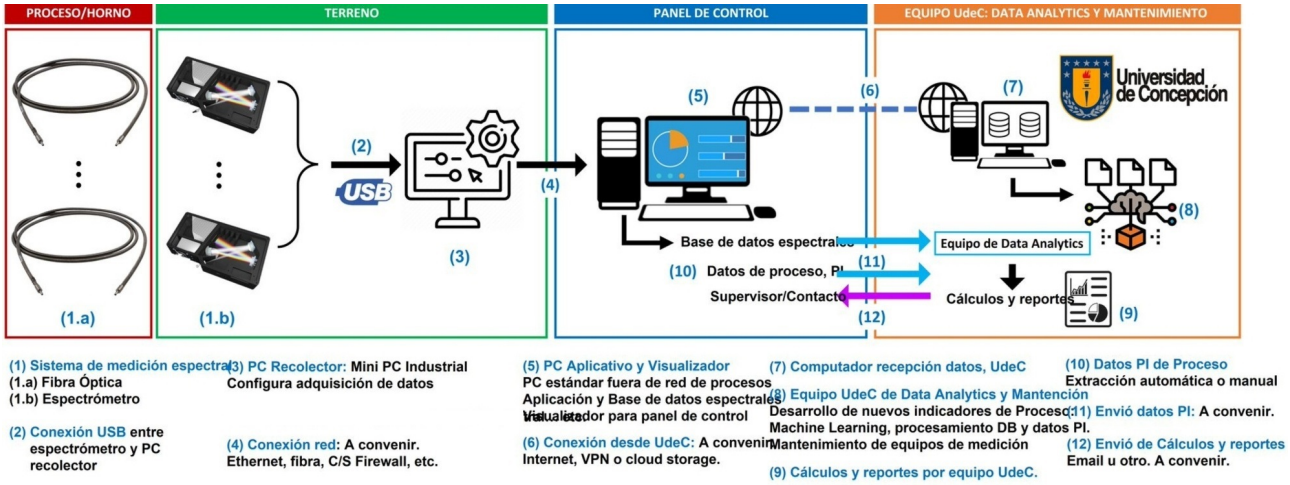
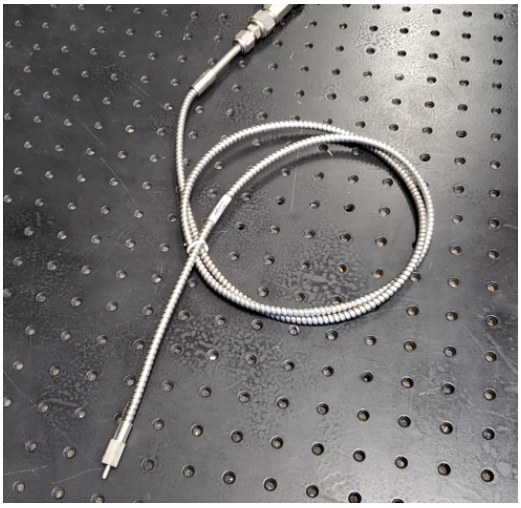


Figura 4.5: Estructura del sistema de sensor optoelectrónico desarrollado por la EBTU.

- **Adquisición de datos:** Corresponde a toda la parte física de obtención directa de los datos radiométricos, capturándolos desde lo que se conoce como *baño*, que es donde se encuentra el metal fundido, y transportándolos hacia un PC de cómputo lejos de la hostilidad del entorno. En la Figura 4.6 se muestran versiones físicas de referencia de los componentes que se explican a continuación.

  - *Fibra óptica acondicionada:* La captura de la señal analógica del proceso se realiza mediante el uso de una fibra óptica multinúcleos monomodo, lo cual cuida la integridad y el uso de ella, en el sentido de que se tienen núcleos de reserva por si ocurriesen fallas. Va recubierta por una estructura segura y flexible en toda su extensión hasta la conexión con el espectrómetro, además de estar protegida por un *housing* en el extremo medidor para soportar las altas temperaturas en las cercanías del horno.
  - *Unidad de alimentación de datos:* El espectrómetro se encarga de recibir y digitalizar la señal de irradiancia que llega desde la fibra, a partir de la cual se podrán hacer los cálculos y procesamiento pertinentes en lo que resta de la cadena de datos. Dicha señal se exporta como datos en formato binario con resolución de 64 bits compatibles con C. Un script en Python en el PC aplicativo extrae estos datos, los organiza en estructuras de arreglos y genera archivos con formato HDF5 jerárquico para facilitar su análisis y almacenamiento en las siguientes fases.
  - *PC de configuración:* Es importante aclarar que en el diagrama de la Figura 4.5 que fue proporcionado por la EBTU, existe una diferencia en el flujo de los datos. El PC de configuración no necesariamente está entre las unidades de alimentación de datos y tiene conexión con el PC de cómputo, sino que contiene lo necesario para calibrar y preparar a los equipos antes de la medición. Dentro de los ajustes radiométricos de calibración, como el cambio entre modos de escaneo, la selección de espectros y



(a) Fibra óptica

(b) Espectrómetro Thorlabs CCS175

(c) Ejemplo de PC de configuración

(d) Cableado para transmisión de datos

Figura 4.6: Componentes de la fase de adquisición de datos radiométricos.

la ejecución de controles y correcciones varias de la señal, existe el de la elección del tiempo de integración. Esta característica es responsable directa de la frecuencia con la que se generan los datos, la que puede variar aproximadamente entre 1 y 3 segundos por archivo, cada uno de unos 40MB.

- *Cableado:* Por limitaciones monetarias, la longitud de la fibra es tal que el espectrómetro pueda ubicarse a una distancia segura dentro de una caseta denominada *coolbox*. Lo restante hasta el PC de cómputo, que pueden ser varias decenas de metros, muchas veces es imposible de alcanzar únicamente con el protocolo estándar USB de salida del espectrómetro. Ante esto, una solución práctica que no degrada significativamente la calidad de la señal, es la de usar un adaptador USB/Ethernet en el lado del equipo de medición, cubrir la distancia con cableado UTP y usar un adaptador inverso para la conexión con este PC aplicativo.

- **Control de datos y cómputo:** Todo el procesamiento se lleva a cabo en un PC base que se encuentra en la sala de control de la planta. Allí se ejecutan los códigos que recogen los datos de la fase anterior para almacenarlos como registros históricos y se calculan los KPIs necesarios para entregarlos a la fase de visualización.

  Formulado en Python, el código fuente funciona en base a varios archivos en forma de módulos encargados de tareas específicas. En ellos se crean clases de objetos que heredan atributos entre sí para manejar los datos de manera secuencial o en cascada, mediante llamadas dentro de la misma máquina. Así, se extrae información particular para cada uno de los hilos de procesamiento del código, lo que en última instancia entrega, en tiempo real, el cálculo de indicadores relevantes como la emisividad, la temperatura del metal fundido, la presencia o ausencia de cuajo y la ley de cobre.

  Ninguno de los módulos que se mencionan está aislados de los demás y entre todos contribuyen a todas las fases de la cadena de datos del sistema. Parte de la lectura, calibración y acondicionamiento de los datos brutos se sigue realizando aquí, como la estructuración de los archivos al formato HDF5. Porciones del código manejan los directorios para guardar los archivos como una base de datos histórica, y es este PC base el que tiene conexión a internet supervisada por la empresa usuaria para llevar a cabo el almacenamiento externo de los datos.

- **Despliegue de datos:** La ejecución del programa viene equipada con una interfaz gráfica intuitiva centrada en proporcionar a los operarios todos los indicadores de rendimiento necesarios para su toma de decisiones. A ella son enviados los datos de las distintas fases anteriores, lo que permite que sus paneles abarquen gráficos de visualización de tendencias, tablas con sistemas de alerta y datos de ajustes medición de los espectrómetros.

  En la práctica, la GUI mostrada en la Figura 4.7 está desarrollada en Kivy y está acoplada al script principal del sistema, por lo que está implementada en Python. Este marco de desarrollo de aplicaciones tiene soporte multiplataforma, otorgándole a la GUI la característica de responsividad para ajustarse a pantallas de distintos dispositivos. A pesar de esto, actualmente la exhibición de la información no está contenida en un subproceso que entable una comunicación con otro dispositivo, sino que se usan aplicaciones de terceros para acceder de manera remota a la pantalla del PC base.

- **Almacenamiento de datos:** Tal como se ve en el diagrama de la estructura del sistema, el PC aplicativo posee una carpeta de almacenamiento local en Windows, donde el programa guarda un registro cronológico de toda la información de interés que ha sido capturada, estructurada y guardada en los archivos `.h5`. Para acceder a ella de manera remota, esta está sincronizada como carpeta compartida con una unidad de almacenamiento en la nube de Google Drive.

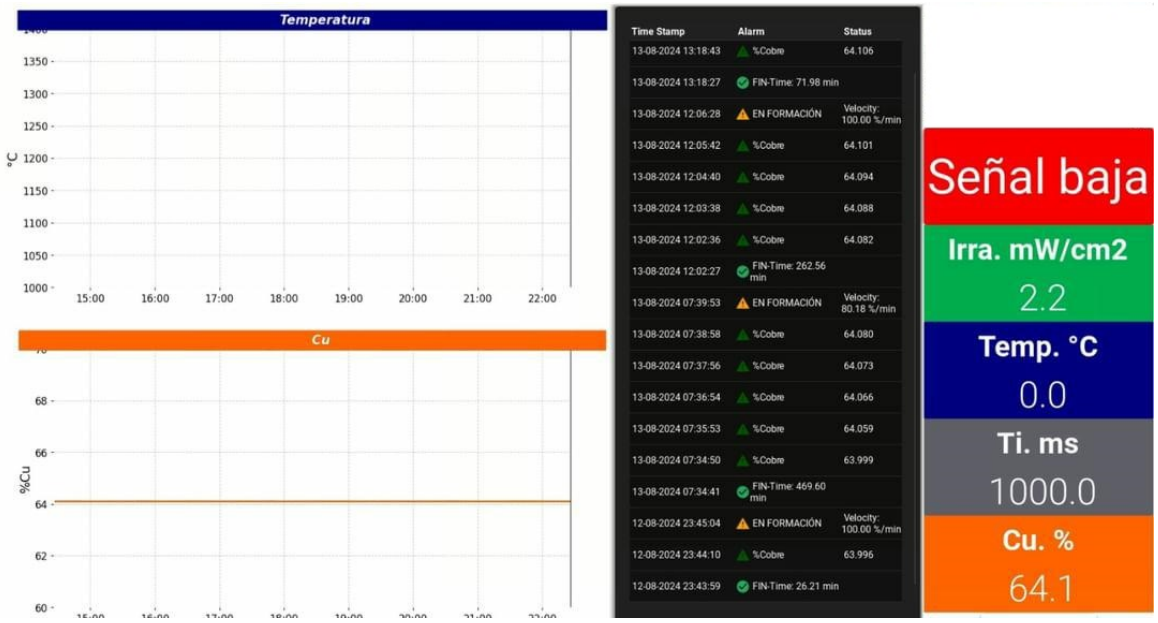


Figura 4.7: GUI del sistema desarrollado por la EBTU.

De forma externa, independiente y no necesariamente en sincronía con el almacenamiento en la base de datos espectrales local del PC de cómputo, un equipo de trabajo de *data analytics* accede a la carpeta compartida en la nube y aplica algoritmos de estimación de los KPI, junto con variadas técnicas de radiometría y *machine learning*, con el fin de lograr la mayor precisión ante las variaciones operacionales de planta.

4.4. Identificación de problemas

Hasta ahora, el sistema ha demostrado ser capaz de operar en distintos ambientes industriales y de entregar el control y monitoreo esperado en estas áreas. Sin embargo y según conversaciones con el equipo de trabajo, se han presentado problemas referentes a la escalabilidad del sistema y a la seguridad en la transmisión de los datos desde planta a la Universidad.

Si bien no han existido ataques dirigidos ni problemas de ciberseguridad, al trabajar con empresas importantes, buena parte del flujo se traba en la desconfianza de publicar la información crítica de sus procesos. En ocasiones, también están involucrados temas legales y regulaciones internas que exigen estándares, certificados y firmas digitales, por lo que asegurar ese aspecto es clave y es algo con lo que se está al debe.

Como se dijo, el software realiza todas las tareas de procesamiento dentro de un mismo PC, ejecutando el código base que funciona mediante creación de clases y herencias, y cada atributo se va traspasando de un nivel de procesamiento al siguiente en forma de capas. Esto, aunque cada tarea esté bien definida y separada por módulos, sugiere que el sistema trabaja con una estructura monolítica y que toda la carga computacional se la está llevando una sola máquina. Así, se tienen los problemas típicos de un sistema de este tipo:

- *Escalabilidad limitada:* El sistema actual está preparado para trabajar con un solo espectrómetro, por lo que aumentar la cantidad de unidades de alimentación de datos traería problemas relacionados con el rendimiento del sistema. Trabajar con la frecuencia de generación y tamaño de archivos actual ya supone un desafío a la hora de analizar datos asociados a un día de mediciones, donde se deben manejar volúmenes del orden de decenas de GB por una única máquina. Aunque hoy por hoy es posible hacer el envío de esos datos e incluso realizar el procesamiento en dicha máquina, esta está próxima a alcanzar un límite en términos de costo computacional, por lo que añadir la inyección de datos por parte de otro equipo de medición multiplicaría el volumen de datos, superando las capacidades actuales del hardware y alcanzando un umbral de factibilidad para el entorno en el que se está trabajando.

- *Falta de distribución de carga:* En un sistema monolítico, es difícil distribuir las cargas de trabajo en diferentes máquinas. Esto limita la capacidad de aprovechar la computación distribuida y presenta un potencial cuello de botella cuando se requieran procesamientos incluso más intensivos de otros KPIs a medida que escalan las exigencias el sistema. Llevar a cabo una distribución en el estado actual requiere replicar el mismo sistema en distintos computadores, necesitando el mismo hardware en cada uno de ellos sin tener un impacto relevante en el rendimiento y elevando los gastos monetarios en vano.

- *Fallas centralizadas:* De la mano con lo anterior, si el sistema depende de un solo PC, un fallo en el hardware o software de esa máquina detiene todo el sistema y generar altos MTTR, afectando la disponibilidad del sensor y dejando a los operarios sin información crítica de la fundición. En momentos en los que ha fallado el sensor responsable de la captura de datos, ocurre una falla en cascada que compromete todas las fases del sistema y afecta a la veracidad de los indicadores calculados. Para regularizar la situación en estos casos, se ha debido dar de baja el sistema, realizar las modificaciones pertinentes y volver a ejecutar todo, asumiendo las pérdidas de datos durante el tiempo de recuperación.

- *Baja flexibilidad:* Otro problema asociado, es que, al estar todo el sistema integrado en una sola unidad, realizar cambios o actualizaciones en una parte específica del código es complejo y arriesgado, ya que todas las fases están estrechamente acopladas. Estos cambios pueden ir desde cambios visuales simples, como mostrar en pantalla algún indicador necesario para el usuario u ocultar otro que no precisa, hasta cambios sustanciales, como modificar o mejorar el sistema completo a una nueva versión que incluya otro tipo de cálculos, mediciones o manejo de datos. Independiente del tipo de modificación, se ha necesitado tomar la misma medida del caso anterior de bajar y reinstalar el sistema.

5. Propuesta de diseño

5.1. Identificación de requisitos

1) ***El sistema debe estar preparado para el aumento en la cantidad de sensores:*** El sistema de sensores debe ser capaz de escalar eficientemente para manejar un aumento en el número de sensores y la cantidad de datos recolectados. Esto implica que la arquitectura debe soportar la adición de nuevos sensores y servicios de traspaso de información sin afectar el rendimiento general del sistema. Además, es importante que el sistema tenga la capacidad de escalar dinámicamente en respuesta a la demanda, lo cual es esencial para garantizar que el sistema continúe operando de manera confiable, incluso en escenarios de alta carga.

2) ***El sistema debe ser flexible ante cambios operacionales:*** En conjunto con lo anterior, el sistema debe ser diseñado con un alto grado de flexibilidad para adaptarse a cambios futuros en los requisitos, tecnologías y condiciones operativas. Esto incluye la capacidad de integrar nuevos tipos de sensores, modificar o reemplazar componentes de procesamiento o cambiar la forma en que se despliegan los datos, todo sin necesidad de reestructurar toda la arquitectura. También, es crucial que el sistema soporte múltiples protocolos de comunicación y formatos de datos, asegurando que pueda interoperar con una amplia variedad de dispositivos y servicios externos.

3) ***El sistema debe incorporar herramientas y medidas de seguridad alineadas con los estándares de grandes empresas:*** La aplicación debe tener la posibilidad de adoptar mecanismos de seguridad para proteger tanto los datos como las comunicaciones entre los componentes del sistema. Al implementarse en entornos industriales, es fundamental que el sistema cuente con una gestión segura de la información y que los métodos de intercomunicación que se usen, permitan medidas de seguridad estrictas que eviten cualquier ataque que pueda comprometer la integridad de los datos.

5.2. Descripción del diseño

De acuerdo a los requerimientos expuestos, a la naturaleza del sistema y teniendo en consideración las opciones presentadas en la sección de *Marco teórico*, la arquitectura que se acomoda a esto es una basada en microservicios, donde cada proceso del sistema se implementa como un módulo independiente que entrega los datos a los demás módulos de una manera concreta. Como técnica de comunicación interproceso, la opción más adecuada es el uso de RPC, específicamente gRPC, que admite la adición de servicios o funciones de acuerdo a las

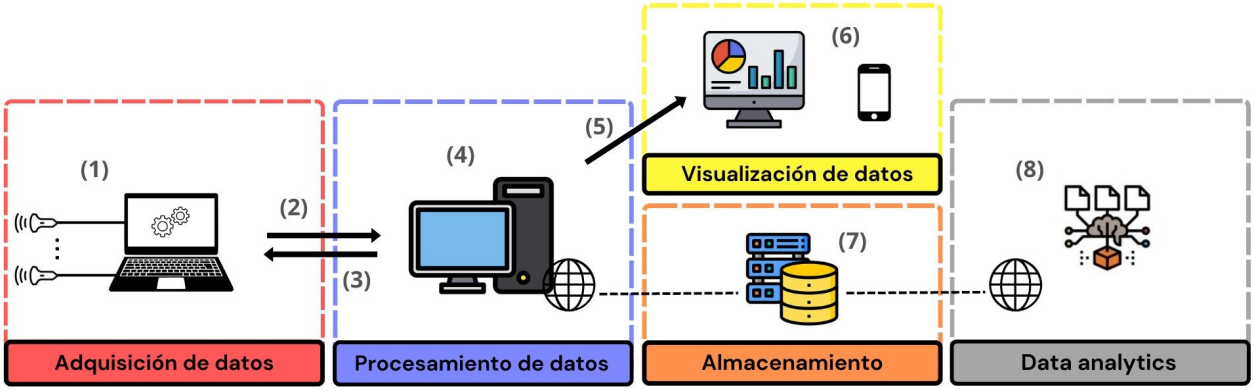


Figura 5.1: Diagrama lógico de la propuesta de diseño.

necesidades de la aplicación y ofrece un alto rendimiento y eficiencia en la comunicación entre los distintos módulos del sistema.

En la Figura 5.1 se muestra un diagrama con el flujo de datos del sistema propuesto. Enseguida se detallan los puntos de interés:

(1) *Módulo de adquisición de datos:* Contiene la porción de software encargada de la configuración de las unidades de alimentación de datos requeridas. Realiza la lectura de los datos brutos, las calibraciones necesarias y el acondicionamiento de los datos, para luego estructurarlos en el formato `.h5` de trabajo y enviarlos al siguiente módulo.

(2) Envío de datos medidos y ordenados mediante un método gRPC que permita un *stream* de información como solicitud.

(3) Instrucciones de cambio de parámetros de medición desde el módulo de cómputo, a través de un método de tipo *unary*.

(4) *Módulo de procesamiento de datos:* Sección del sistema en el que se hace el cálculo de variables operacionales en base a lo recibido. Entabla comunicación con el módulo de visualización para enviarle los resultados de interés y con el módulo de almacenamiento para guardar el registro histórico del proceso.

(5) Envío por medio de un método gRPC de un *stream* continuo de datos procesados hacia el equipo que despliegue la interfaz gráfica.

(6) *Módulo de visualización:* Tiene la tarea de recibir los KPIs desde el módulo de procesamiento, de crear sistemas de alertas, de exhibir gráficos del proceso en una interfaz gráfica y de levantar un subsistema para el acceso remoto desde dispositivos personales (opcional).

(7) *Módulo de almacenamiento:* Parte del sistema que maneja los archivos. En él se registran los datos históricos y a él se recurre para usar los datos como base de la creación de modelos

predictivos por parte del equipo de *data analytics*. La comunicación puede realizarse con gRPC para realizar las operaciones CRUD, como también puede hacerse con REST. Esto queda a convenir dependiendo del framework del sistema de almacenamiento.

(8) Extracción de información desde la base de datos para la generación de modelos de estimación y reportes.

En la distribución propuesta, cada unidad del sistema realiza sus funciones de manera aislada con respecto a las demás unidades. En particular, se propone que cada módulo opere en una máquina distinta, teniéndose así, además de una distribución a nivel de software, una distribución a nivel de hardware que evite tener equipos sobredimensionados. La escalabilidad se logra al permitir que los módulos puedan crecer libremente y la modularidad de la arquitectura ofrece la flexibilidad de adaptar el sistema a nuevos requerimientos operativos, permitiendo editar y definir nuevos formatos de traspaso de datos, así como también añadir métodos para intercomunicar nuevos módulos.

La comunicación entre los equipos se hace vía una red privada supervisada por la empresa usuaria, va cifrada por TLS y se le puede aplicar a conveniencia cualquiera de las técnicas de seguridad que ofrece el entorno gRPC, como la autenticación por OAuth2 o tokens, la implementación de políticas RBAC o el uso de interceptores.

Debido a que las llamadas entre los módulos, ya sean *unary* o *client/server streaming*, siguen un modelo cliente-servidor, es importante recalcar que el papel de servidor que ofrece los servicios y métodos detallados en el archivo `.proto`, lo puede desempeñar un equipo considerado central, el cual sería responsable de todo el control de flujo de los datos y sería el PC de procesamiento en este caso; o puede ser adoptado por varios computadores, donde cada uno podría ofrecer los servicios que cualquier otro módulo de otra máquina quiera consumir.

Con respecto al almacenamiento de los datos, de acuerdo a conversaciones con el equipo de diseño, la idea es desarrollar una base de datos propia que permita el trabajo con gRPC. Sin embargo, las empresas con las que trabaja la EBTU son de gran calibre, por lo que existen temas legales que no se pueden pasar por alto. Esto lleva a que estos clientes requieran una seguridad mayor a la hora de almacenar sus datos y que no confíen en una unidad de almacenamiento propia del sistema. Para sortear estos inconvenientes, se inclinó por la opción de trabajar con un producto que ofrezca todos los estándares y las medidas de seguridad e integridad de los datos que necesitan los clientes de este tipo, como lo es la Autonomous JSON Database de Oracle. Este sistema de almacenamiento trabaja con una RESTful API y sería la excepción a las comunicaciones por gRPC que se usan en el resto del sistema.

En la Tabla 5.1 se presentan las especificaciones de las soluciones adoptadas para satisfacer los requisitos de escalabilidad, flexibilidad y seguridad de la nueva estructura del sistema de sensores.

Tabla 5.1: Especificaciones de las soluciones para el cumplimiento de requisitos del nuevo sistema de sensores.

Requisito	Solución	Especificaciones
Escalabilidad	gRPC	**Multiplexación HTTP/2:** Permite del orden de miles o decenas de miles de conexiones y solicitudes simultáneas, sin comprometer el tamaño de estas últimas y optimizando el ancho de banda, limitándolo netamente a la conexión a internet.
		Streaming bidireccional: Facilita el envío de datos en flujos continuos, aprovechando mejor el ancho de banda disponible y evitando cuellos de botella
		Balanceo de carga: Provee opciones de configuración para la escalabilidad horizontal mediante la distribución de sus conexiones y la compresión de mensajes, mejorando así también la eficiencia en la transmisión
		Definición de servicios Protobuf: Ofrece la posibilidad de definir una cantidad virtualmente ilimitada de servicios en el archivo `.proto`, para la incorporación de distintas tareas sin modificar la arquitectura a medida que crece el sistema
Flexibilidad	Microservicios	**Desacoplamiento de servicios:** Cada módulo (microservicio) se puede modificar o actualizar sin afectar a otros, facilitando la adaptación a nuevos requisitos
		Adaptabilidad y escalabilidad de servicios: Integrar nuevos tipos de sensores puede requerir la creación o modificación de microservicios, manteniéndose con esta arquitectura la coherencia y eficiencia del sistema general
		Compatibilidad y conectividad externa: Cada microservicio puede proporcionar una API o implementar adaptadores para interactuar con sistemas externos que pueden llegar a tener una amplia variedad de protocolos, aumentando así la interoperabilidad del sistema
Seguridad	gRPC	**Cifrado TLS:** gRPC permite implementar TLS para cifrar las comunicaciones entre los microservicios, evitando intercepciones o alteraciones no autorizadas
	Autonomous JSON Database	**Cifrado de datos y auditoría:** La Autonomous JSON Database ofrece cifrado en reposo y auditoría de accesos, registrando todas las interacciones y protegiendo los datos de accesos no autorizados a través de RBAC
		Prevención y detección de ataques: Esta opción de almacenamiento de datos puede detectar amenazas en tiempo real y aplicar parches de seguridad automáticamente, asegurando la integridad y disponibilidad de los datos

6. Prueba de concepto

El cambio de una arquitectura en el desarrollo de software es un proceso generalmente complejo que incluye grandes desafíos que se van abordando con el tiempo, de manera bien estructurada y tomando todas las consideraciones correspondientes. Es una tarea que involucra una coordinación entre todos los equipos de trabajo y puede requerir afrontar algunos riesgos. Hay algunos como lo puede ser la complejidad técnica, en el que se debe asegurar que los componentes que van a constituir la nueva arquitectura sean compatibles entre sí, además de necesitarse que se haga una buena integración con servicios externos o de terceros que se estén usando. También se puede incurrir en riesgos de interrupción si el sistema está trabajando en tiempo real, pues puede haber fallas en el sistema que lleven a tiempos de inactividad durante esta transición. Por último, si el cambio técnico es importante, esto puede traer grandes costos, tanto si existen o no problemas con los nuevos componentes.

En cuanto a los desafíos sociales, también existen varios a tener en cuenta. Como se dijo, una transición así debe ser bien estructurada y ordenada. Generalmente, proyectos como estos tienen una gran planificación previa para llevarse a cabo, donde se precisa de buena comunicación entre los distintos departamentos o equipos que existan. No es poco común encontrarse con que hay trabajadores resistentes al cambio y a adoptar nuevos métodos o tecnologías, por lo que se podría incluso necesitar capacitación para entrenar a este tipo de personal. Actualizar la documentación es clave para que cualquier integrante sea capaz de entender todo el funcionamiento del sistema. Así, entonces, se deben tener informaciones o manuales claros y accesibles para todos los usuarios y desarrolladores. Finalmente, una vez que la actualización se completó, es imperante tener en la mira el tema del soporte, donde se debe establecer un plan de mantenimiento y de identificación y solución de errores.

Ante esta cantidad de temas emergentes, por limitaciones de tiempo y por el alcance de este trabajo, se llevó a cabo una prueba de concepto de una versión muy acotada del sistema presentado en el capítulo anterior. Esto es, se diseñó e implementó lo justo y necesario que contempla toda la cadena de datos, desde la adquisición hasta el despliegue de estos.

6.1. Sistema de prueba

El sistema consiste en un Arduino con un sensor ultrasónico conectado, el cual está leyendo, calculando y enviando cada dos segundos la distancia desde él a un objeto situado en frente. Esta implementación sigue la misma filosofía de microservicios, en la que cada módulo trabaja de manera independiente del otro y están mediados por un servidor que entabla la comunicación entre ambos por gRPC, recibiendo los datos calculados y enviándolos hacia la sección de exposición de datos. Su despliegue conceptual es el que se muestra en la Figura 6.1, donde también se ve la dirección del flujo de datos. Aquí vemos que el número de módulos se redujo a

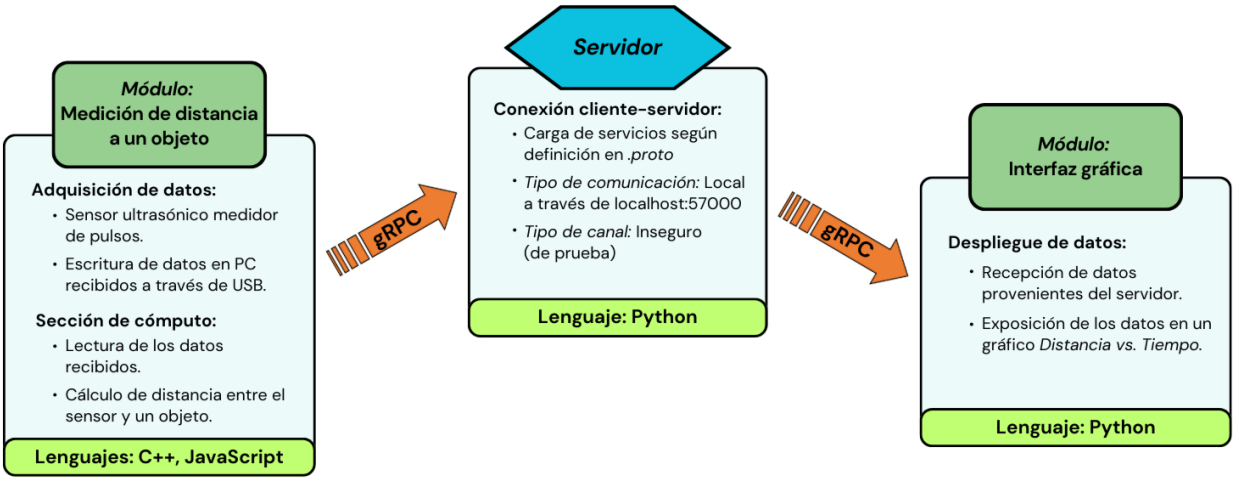


Figura 6.1: Diagrama conceptual para la implementación del sistema de prueba.

dos, dentro de los cuales cada uno contempla alguna o algunas de las cuatro fases generales de un software para sensor. Cada uno contiene lo siguiente:

- ***Módulo:* Medición de distancia a un objeto**
  El módulo en general consiste de dos partes; la física de detección y cálculo de distancia, junto con su programación correspondiente, y la de creación del cliente, carga de servicios y métodos de gRPC, y envío de datos al servidor.

  Para la primera parte de adquisición y cálculo se escogieron los siguientes componentes:

  - *Microcontrolador Arduino Uno:* ofrece un IDE diseñado para especificar las acciones o tareas a realizar con las señales que llegan a sus distintos pines y puertos. Alimenta y controla los distintos dispositivos que se le puedan conectar, como por ejemplo, actuadores, sensores de humedad, de temperatura o de magnetismo.
  - *Sensor ultrasónico HC-SR04:* consta de un transductor que envía una señal en pulsos con una frecuencia de $40[kHz]$, y otro transductor que recibe el rebote de esos pulsos, para luego registrar el tiempo de viaje de la señal. Con ese tiempo, se calcula la distancia hasta el objeto en el que rebotó el pulso.
  - *Protoboard y cables:* Al conectar el sensor a una protoboard, se debe hacer la conexión al Arduino con cables Dupont. Además, se usa el cable de alimentación del microcontrolador para enviar los datos hacia el PC de control, a través de una comunicación serial por USB.

  En cuanto a los entornos y lenguajes usados, tenemos que el IDE del Arduino usa un lenguaje C++ modificado y acondicionado particularmente para su uso. Este es usado en el sistema para especificar los pines a los que se tiene conectado el sensor y procesar las

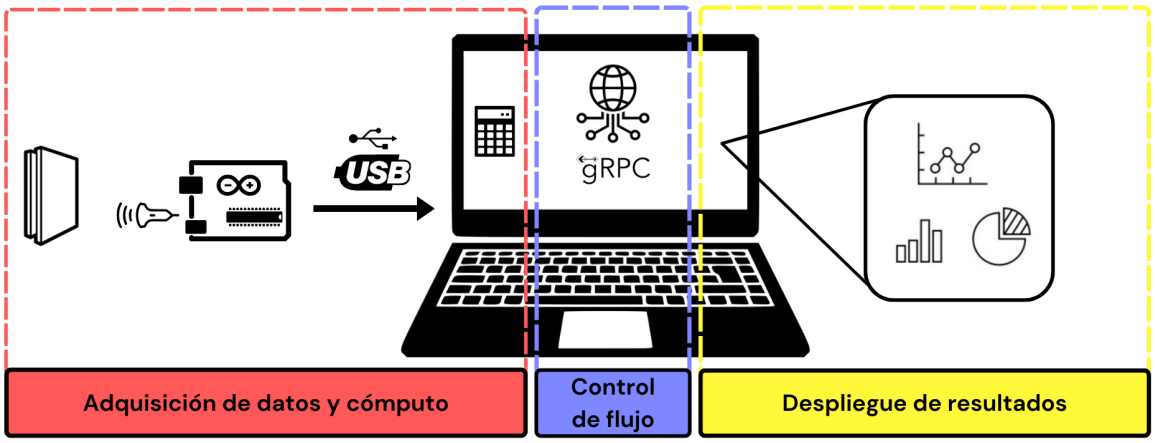


Figura 6.2: Diagrama lógico para la implementación física del sistema de prueba.

señales entrantes, capturándolas y usando los valores recibidos para calcular la distancia al objeto mediante la fórmula aproximada

$$d = \frac{v \cdot t}{2} \ , \tag{6.1}$$

donde t es el tiempo de ida y vuelta de la señal, v es una velocidad estándar de $343[m/s]$ y d es la distancia final a medir.

Una vez se completa esta primera fase, se da comienzo a la lectura de los datos de distancia medida. Mediante un script en JavaScript que es el que hace de cliente, se toman los datos enviados por USB cada dos segundos y se escriben en un archivo `.csv` de forma volátil. Esto significa que cada vez que llega un nuevo dato, se sobrescribe y no hay memoria de las mediciones anteriores. Ahora, cada dos segundos también, se leen y se envían esos datos al servidor.

La conexión con el servidor tiene las siguientes características:

- *Tipo de comunicación:* La comunicación entre este módulo y el servidor se hace de manera local y en la misma máquina. El socket en cuestión sería `localhost:57000`, donde 57000 es un puerto arbitrario abierto por el servidor.
- *Tipos de llamadas:* En este sistema se usan las llamadas Unary, en las que se envía una sola solicitud con el dato de distancia calculado y se recibe una respuesta vacía, pues el servidor no tiene nada que enviar de vuelta.
- *Seguridad:* Este sistema de prueba no cuenta con ninguna seguridad en particular. El canal de comunicación que se abre es del tipo `insecure_channel()` y no se implementan atributos de TLS como encriptación, autenticación o certificados digitales.

- ***Servidor* y *Módulo:* Interfaz gráfica**
  Referente al módulo de despliegue de datos, es bastante similar al primero si nos enfocamos en el flujo de datos. Mediante un script en Python, también dentro del mismo computador, se entabla la comunicación con el servidor con el mismo socket `localhost:57000` y el canal inseguro de prueba. A diferencia del otro módulo, este envía una solicitud vacía cada dos segundos para recibir como respuesta el envío de datos de distancia que toma el servidor. Luego, expone una interfaz simple en la que se ve un gráfico *Distancia vs. Tiempo* en tiempo real a lo largo de 20 segundos.

  Por último, el servidor corresponde a un script en Python ejecutado de igual manera en el mismo equipo. Este ente mediador es el que se encarga de crear el servicio y cargar los métodos que usan los clientes para hacer sus llamadas, métodos que ya están definidos en el archivo `.proto` distribuido previamente a todos los participantes de la comunicación. Un diagrama lógico para la implementación física de este sistema es el que se muestra en la Figura 6.2.

Cabe mencionar que la elección de los lenguajes de programación fue hecha meramente para evidenciar la posibilidad que ofrece la API de gRPC de trabajar de manera independiente y con el lenguaje que se desee, dentro de las opciones para las cuales tiene soporte [19].

6.2. Implementación

De acuerdo a lo expuesto en la sección anterior, el setup final para la prueba de concepto es el de la Figura 6.3. El experimento consiste en ubicar un objeto frente al sensor por unos segundos, luego alejarlo por otros segundos y finalmente alejarlo de nuevo para que los datos que se envíen tengan algún tipo de variación dentro del lapso de 20 segundos de medición. Ya teniendo el setup montado, se realizan los pasos necesarios:

1. *Prender el servidor:* Al ejecutar el Código 1 del servidor, este queda en un estado de escucha indefinida para enviar y recibir solicitudes según corresponda. Dentro de él, se crea la clase que implementa los métodos definidos en el archivo `protocolo.proto` (Código 2). Estos son dos: uno para almacenar la solicitud entrante del módulo de adquisición llamado `Envio`, y otro para recuperarla y reenviarla al módulo de despliegue llamado `Recibo`. El primero recibe un valor de punto flotante y retorna una respuesta con un campo vacío, pues la comunicación en este caso no requiere que el módulo de adquisición reciba nada de vuelta. El segundo es muy similar, pero esta vez en el otro sentido, recibiendo una solicitud vacía y retornando el valor que se obtuvo del servicio anterior.

2. *Comenzar la adquisición de datos:* Habiendo abierto el IDE de Arduino, se sube el Código 3 que configura al microcontrolador para interpretar las señales del sensor ultrasónico y

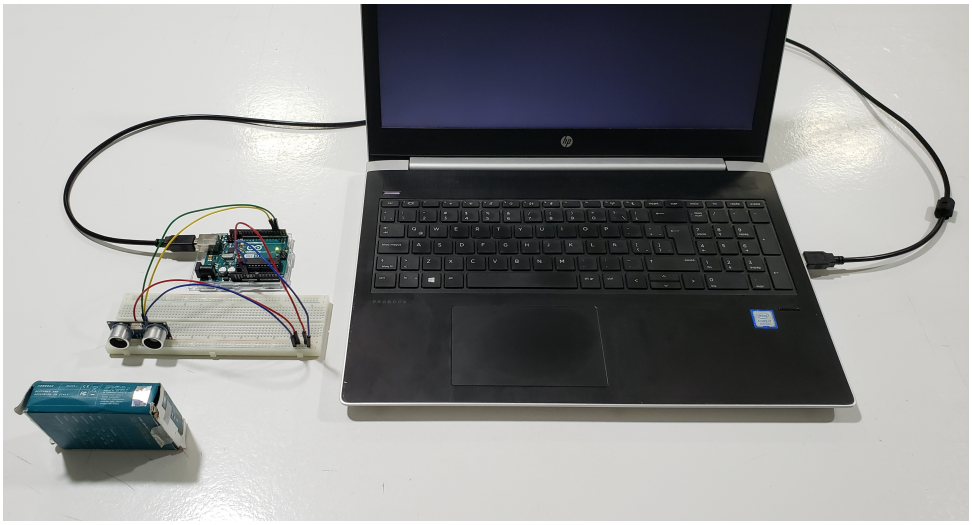


Figura 6.3: Setup final del sistema de prueba.

realizar la medición. Esto envía los datos a través de una comunicación serial al puerto `'COM4'` del PC con una tasa de baudios estándar de 9600. Luego, se inicializa el Código 4 para captar el tráfico entrante por ese puerto, crear el archivo `archivo.csv` y escribir en él dicho tráfico. En este punto, se llaman las funciones que leen el dato y que crean el socket para entablar la comunicación por gRPC, enviando el valor flotante al servidor cada 2 segundos.

3. *Ejecutar la interfaz gráfica:* Utilizando el mismo socket gRPC para comunicarse con el servidor, el Código 5 se encarga de capturar los datos reenviados desde el servidor y mostrarlos en un gráfico *Distancia vs. Tiempo*. Este tiene definidas las cotas 0 y 20, tanto para el eje x como para el eje y. Es decir, se exponen las distancias medidas en un lapso de 20 segundos y se exhiben mediciones de hasta 20 centímetros. Para recibir la información, el cliente envía cada 2 segundos la solicitud vacía `EmptyMessage` con la que avisa al servidor que está listo para recibir la respuesta con los datos en cuestión. En el momento en que se recibe esa respuesta, se registra lo recibido en la gráfica. Con el fin de proveer una interfaz gráfica como tal, se incluyeron también unos recuadros con los que el usuario puede ajustar los límites del eje y, en caso de recibir datos que estén fuera del rango preestablecido, junto con un botón para actualizar dichas cotas.

Luego de haber hecho todo esto, se ve en la Figura 6.4 que el sistema se desempeña de la manera esperada. Durante su funcionamiento, se logra la lectura y cálculo de distancia por parte del sensor, y el primer cliente registra con éxito dichos datos en el archivo `.csv`, tal como se muestra en la Figura 6.5, para luego leerlos y realizar el envío según lo configurado. A su vez, se ve también que el siguiente cliente está recibiendo los datos de distancia, los que se exponen en el gráfico de la Figura 6.6.

Con todo lo anterior se evidencia que esta organización en microservicios comunicados por

gRPC es capaz de cubrir toda la cadena de datos de un sistema de adquisición de datos simple, el cual se puede complejizar según las necesidades, ya que cada módulo trabaja de manera independiente y puede llevar a cabo las funciones que se deseen, siempre y cuando se respete el modo de uso de los servicios predefinidos y las estructuración de los datos a la hora de llegar a los puntos finales o *endpoints* de tales módulos. En la Tabla 6.1 se muestra una comparativa de las capacidades que tiene el sistema monolítico actual y las que puede suplir o mejorar una arquitectura modular como la implementada.

Figura 6.4: Consolas correspondientes a los códigos del cliente de adquisición de datos (izquierda), del servidor mediador (centro) y del cliente de despliegue de datos (derecha).

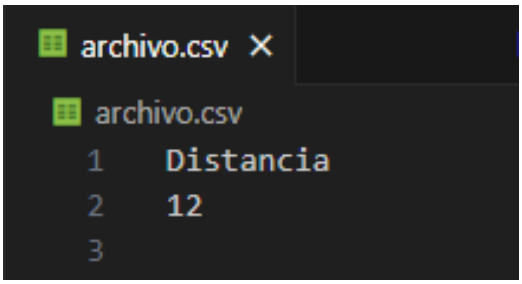


Figura 6.5: Muestra del archivo CSV en un instante arbitrario de la medición.

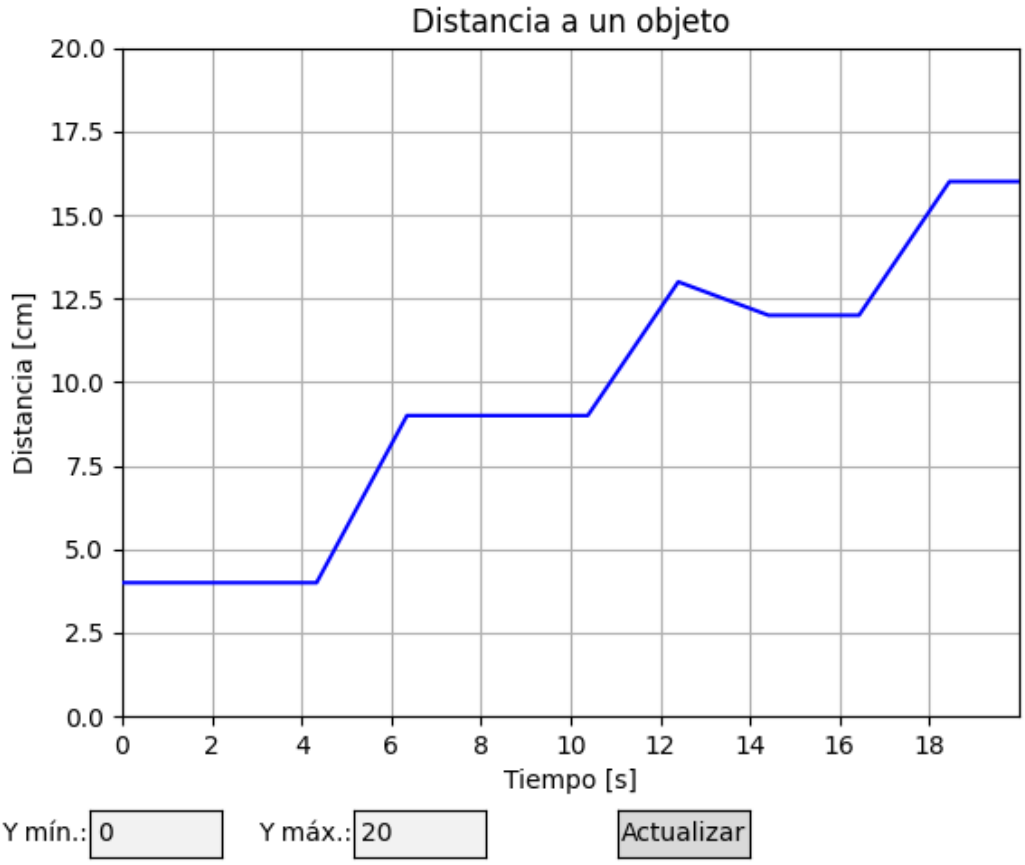


Figura 6.6: Interfaz gráfica del sistema de prueba con los datos de distancia recibidos.

Tabla 6.1: Comparación entre las capacidades del sistema monolítico actual y la prueba de concepto modular.

Sistema actual	Criterio	Prueba de concepto
Agregar nuevos procesamientos requiere una gran sincronización y documentación de la totalidad del sistema, además de un alto tiempo de desarrollo para cada modificación.	**Adición de funcionalidades**	Se evidencia la simplicidad a la hora de agregar una nueva funcionalidad, como otro tipo de medición o el uso de otro sensor, agregando métodos en el archivo .proto, junto con la definición del tipo de mensajes a usar.
Todo tipo de actualización debe seguir la misma estructura, el mismo lenguaje y entorno de programación, necesitando que los desarrolladores adapten las funciones, clases o subsistemas que se quieran agregar al código base.	**Agilidad**	A través de la estructura modular y del agnosticismo frente al lenguaje que provee gRPC, se ve que se pueden incorporar nuevos equipos de trabajo sin la necesidad de seguir una estructura rígida, sorteando los problemas operacionales de ese estilo que surgen en el desarrollo de software.
Teniendo llamadas integradas dentro de una única aplicación, no se tienen las cargas y latencias asociadas a las llamadas remotas.	**Desempeño**	Si bien existe un delay entre la generación de datos y la visualización en el gráfico, esto se debe a una fallad en el manejo de los datos dentro de los módulos. Con un buen diseño de los microservicios y una infraestructura de red rápida y estable, gRPC tiene el potencial de equipararse al desempeño de un monolito gracias a sus técnicas de multiplexación y serialización de los datos.
Al ser un bloque sólido de código, la seguridad se puede gestionar desde un solo punto central con una capa de middleware dedicada a la autenticación y autorización. Sin embargo, si el sistema crece, se vuelve complejo separar de manera adecuada privilegios y roles y se debe revisar exhaustivamente el código para evitar vulnerabilidades.	**Aseguramiento de los datos**	gRPC ofrece una gran variedad de maneras de ofrecer una alta seguridad en la comunicación entre sus servicios y no tiene el problema asociado a la escalabilidad.

7. Conclusiones

A lo largo de este informe se realizó una revisión teórica de patrones arquitectónicos y técnicas de comunicación interproceso relevantes dentro de la ingeniería de software, con el objetivo de entender y aplicar los conceptos al diseño y optimización de un sistema industrial. Se investigó tanto el caso general como el caso particular de un sistema de sensores para adquisición de datos radiométricos de fundiciones de cobre, el cual presenta problemas típicos de un sistema estructurado en forma de monolito.

A partir de este análisis y fundamentado por la necesidad de crear un sistema modular y escalable que permita la autonomía de sus componentes y la interacción eficiente entre ellos, se propuso un diseño basado en una arquitectura de microservicios comunicados de manera segura a través de gRPC. Esta arquitectura consideró las tareas particulares de cada sección del sistema creado por la EBTU, desacoplando las responsabilidades, distribuyendo cargas computacionales y asignando conexiones que se adecúan a los requerimientos de operación.

Finalmente, se implementó una prueba de concepto que captura toda la cadena de datos del sistema, desde la adquisición hasta el despliegue, con la flexibilidad de sumar o quitar módulos y de desarrollar cada uno en el entorno informático que se adapte mejor al contexto. Este prototipo demostró que se cumple la hipótesis de permitir una mayor adaptabilidad general a nuevas tecnologías y requerimientos operativos, pues, a partir de esta filosofía, es posible construir sistemas complejos utilizando esta arquitectura modular como base. Además, la comunicación y las herramientas que ofrece gRPC permiten replicar, adaptar y asegurar el flujo de información de un sistema para distintos casos de uso.

Se determinó que, en este caso, realizar una comparación cuantitativa entre el desempeño del sistema actual y el de la prueba de concepto con la nueva opción de diseño, no tiene propósito, pues los sistemas no son equiparables ni en cuanto a carga de trabajo dentro de los módulos ni en cuanto al volumen de datos de la transmisión entre ellos. Aun así, se hizo una comparación cualitativa, en la que se exponen las potenciales mejoras al adoptar una organización modular como la propuesta, considerando que esta soluciona problemas operacionales asociados a la ingeniería de software y abre las puertas a la separación de responsabilidades, la adición de tareas independientes y la implementación de técnicas de seguridad y balanceo de carga para la eficiencia de la transmisión. Con esto, se llegó a la decisión de que la nueva arquitectura ratifica la hipótesis de permitir una mayor adaptabilidad general a nuevas tecnologías y requerimientos operativos, y valida la puesta en marcha de un cambio de arquitectura monolítica del sistema actual hacia la de microservicios del sistema propuesto.

7.1. Trabajos futuros

El desarrollo y el planteamiento mostrados en este trabajo sientan las bases para futuras exploraciones y mejoras en el diseño e implementación de sistemas distribuidos basados en microservicios. Las líneas de investigación podrían incluir una incursión en las características que ofrece gRPC, como la implementación de técnicas de balanceo de carga ante altas cargas de datos.

Considerando también que la seguridad es un aspecto crítico en los sistemas distribuidos, se podría entrar al área de la ciberseguridad, estudiando y aplicando mecanismos avanzados de autenticación, autorización y encriptación dentro del framework de gRPC, asegurando que el sistema propuesto sea robusto frente a amenazas y accesos no autorizados.

Aunque gRPC demostró ser una solución efectiva para la comunicación entre microservicios, sería valioso comparar su rendimiento y características con otras tecnologías emergentes o existentes, como GraphQL, o con protocolos industriales o basados en HTTP/3, para identificar posibles mejoras o adaptaciones según el caso de uso.

Todo lo anterior permitiría mejorar y expandir no solo el sistema presentado en este trabajo, sino que también contribuirían al avance general en el campo de la ingeniería de sistemas distribuidos, ofreciendo soluciones más robustas, eficientes y seguras para la adquisición y procesamiento de datos.

Bibliografía

[1] Andrew S. Tanenbaum. *Sistemas Distribuidos*. Pearson Educación, 2008. ISBN 9786074428858.

[2] Kasun Indrasiri. *gRPC: Up and Running*. O'Reilly, 2020. ISBN 1492058335.

[3] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. "Microservices: Yesterday, Today, and Tomorrow". *Present and ulterior software engineering*, pages 195–216, 2017.

[4] Freddy Tapia, Miguel Ángel Mora, Walter Fuertes, Hernán Aules, Edwin Flores, and Theofilos Toulkeridis. "From Monolithic Systems to Microservices: A Comparative Study of Performance". *Applied Sciences*, 10(17), August 2020. ISSN 2076-3417. doi: 10.3390/app10175797.

[5] Manuel Mazzara, Nicola Dragoni, Antonio Bucchiarone, Alberto Giaretta, Stephan T. Larsen, and Schahram Dustdar. "Microservices: Migration of a Mission Critical System". *IEEE Transactions on Services Computing*, 14(5):1464–1477, 2021. doi: 10.1109/TSC.2018.2889087.

[6] Antonio Bucchiarone, Nicola Dragoni, Schahram Dustdar, Stephan T Larsen, and Manuel Mazzara. "From Monolithic to Microservices: An experience report". Technical report, Technical University of Denmark, Örebro University, TU Wien, Danske Bank and Innopolis University, 2017.

[7] Mohsen Mosleh, Kia Dalili, and Babak Heydari. "Distributed or Monolithic? A Computational Architecture Decision Framework". *IEEE Systems Journal*, 12(1):125–136, 2018. doi: 10.1109/JSYST.2016.2594290.

[8] Federica Frabetti. *Software Theory: A Cultural and Philosophical Study*. Rowman & Littlefield, 2015. ISBN 178348196X.

[9] Alvine B. Belle, Ghizlane El Boussaidi, Timothy C. Lethbridge, Segla Kpodjedo, Hafedh Mili, and Andres Paz. "Systematically reviewing the layered architectural pattern principles and their use to reconstruct software architectures". *arXiv preprint arXiv:2112.01644*, December 2021. doi: 10.48550/ARXIV.2112.01644.

[10] Mark Richards. *Software Architecture Patterns*. O'Reilly, 2022.

[11] W. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack. "HYDRA: the kernel of a multiprocessor operating system". *Communications of the ACM*, 17(6): 337–345, June 1974. ISSN 1557-7317. doi: 10.1145/355616.364017.

[12] Samesun Singh and Stephen Torri. "Microkernel operating systems compared to monolithic operating systems: a review on functional safety". December 2023. URL https://www.researchgate.net/publication/376582652_Microkernel_operating_systems_compared_to_monolithic_operating_systems_a_review_on_functional_safety.

[13] Alfred Theorin, Kristofer Bengtsson, Julien Provost, Michael Lieder, Charlotta Johnsson, Thomas Lundholm, and Bengt Lennartson. "An Event-Driven Manufacturing Information System Architecture". *IFAC-PapersOnLine*, 48(3):547–554, 2015. ISSN 2405-8963. doi:

https://doi.org/10.1016/j.ifacol.2015.06.138. URL https://www.sciencedirect.com/science/article/pii/S2405896315003778.

[14] SERNAGEOMIN. Anuario de la minería de chile 2021, May 2022.

[15] Prakash Somasundaram. "Enhancing Organizational Data Protection: Advanced Security Measures for Database Systems". *International Journal of Research in Computer Applications and Information Technology (IJRCAIT)*, 6(1):58–62, 2023. ISSN 2347-5099.

[16] DenissLLC. Production Monitoring Software, 2021. URL https://www.denissllc.com/production-monitoring-software/.

[17] Thorlabs. *Optical Coherence Tomography ThorImage OCT Operating Manual*, January 2024.

[18] OceanOptics. *OceanView Installation and Operation Manual*, 2013.

[19] gRPC Supported Languages. URL https://grpc.io/docs/languages/.

8. Anexo: Códigos

Código 1: Código del servidor gRPC para control de envío de datos a través del sistema.

```
import grpc
import protocolo_pb2
import protocolo_pb2_grpc
import concurrent.futures

class DistanciaServicer(protocolo_pb2_grpc.DistanciaServicer):

    def Envio(self, request, context):
        self.solicitud = request.solicitud
        return protocolo_pb2.EmptyMessage()

    def Recibo(self, request, context):
        return protocolo_pb2.Request(solicitud=self.solicitud)


def serve():
    server = grpc.server(concurrent.futures.
                         ThreadPoolExecutor(max_workers=10))
    protocolo_pb2_grpc.add_DistanciaServicer_to_server(DistanciaServicer(),
                                                       server)
    server.add_insecure_port('localhost:57000')
    server.start()
    server.wait_for_termination()

if __name__ == '__main__':
    print("Servidor activo...")
    serve()
```

Código 2: Archivo `.proto` para la definición de mensajes y servicios gRPC de la prueba de concepto.

```
syntax = "proto3";

service Distancia {
    rpc Envio (Request) returns (EmptyMessage) {}
    rpc Recibo(EmptyMessage) returns (Request) {}
}

message Request {
    float solicitud = 1;
}

message EmptyMessage  {
     // No se definen campos
}
```

Código 3: Código de Arduino para medición y cálculo de datos de distancia.

```
// Pines a usar
const int pinTrig = 2;
const int pinEcho = 3;

// Variable de control para detener la ejecucion
bool ejecucion = true;

void setup() {
  pinMode(pinTrig, OUTPUT);
  pinMode(pinEcho, INPUT);

  Serial.begin(9600);
}

void loop() {
  if (!ejecucion) {
    return;
  }

  digitalWrite(pinTrig, LOW);
  delayMicroseconds(2);
  digitalWrite(pinTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(pinTrig, LOW);

  long duracion = pulseIn(pinEcho, HIGH);

  // Calcular distancia
  int distancia = duracion * 0.0343 / 2;

  Serial.println(distancia);

  delay(2000); // Pausa antes de la proxima lectura
}
```

Código 4: Código del cliente gRPC de adquisición de datos en JavaScript.

```
const grpc = require('@grpc/grpc-js');
const protoLoader = require('@grpc/proto-loader');
const fs = require('fs');
const csv = require('csv-parser');
const { promisify } = require('util');
const { SerialPort } = require('serialport');
const { ReadlineParser } = require('@serialport/parser-readline');

const readFile = promisify(fs.readFile);

// Carga del protobuf
const packageDefinition = protoLoader.loadSync('protocolo.proto');
const protoDescriptor = grpc.loadPackageDefinition(packageDefinition);
const serviceName = 'Distancia'; // Nombre del servicio segun .proto
const DistanciaClient = protoDescriptor[serviceName];
if (!DistanciaClient) {
    console.error(`No se pudo implementar "${serviceName}".`);
    process.exit(1);
}

// Inicializacion del puerto serial
async function initSerialPort() {
    const port = new SerialPort({ path: 'COM4', baudRate: 9600 });
    const parser = port.pipe(new ReadlineParser({ delimiter: '\n' }));
    return parser;
}

// Funcion para escribir el CSV
async function writeToCSV(value) {
    const writer = fs.createWriteStream('archivo.csv');
    writer.write('Distancia\n');
    writer.write(`${value}\n`);

    return new Promise((resolve) => {
        writer.end(resolve);
    });
}

async function readCSV() {
    return new Promise((resolve, reject) => {
        const rows = [];
        fs.createReadStream('archivo.csv')
            .pipe(csv())
            .on('data', (row) => {
```

```
45                  rows.push(row);
46              })
47              .on('end', () => {
48                  resolve(rows);
49              })
50              .on('error', reject);
51      });
52  }
53
54  async function run() {
55      const client = new DistanciaClient('localhost:57000',
56          grpc.credentials.createInsecure());
57      const parser = await initSerialPort();
58
59      parser.on('data', async (data) => {
60          const value = parseFloat(data.trim());
61          await writeToCSV(value);
62          try {
63              const rows = await readCSV();
64              if (rows.length > 0) {
65                  const csvValue = parseFloat(rows[0].Distancia);
66                  console.log(`Enviado: ${csvValue}`);
67                  const rpcRequest = { solicitud: csvValue };
68
69                  client.Envio(rpcRequest, (error) => {
70                      if (error) {
71                          console.error(error);
72                          return;
73                      }
74                  });
75              } else {
76                  console.log("El archivo CSV est  vac o.");
77              }
78          } catch (error) {
79              console.error("Error al leer el archivo CSV:", error);
80          }
81      });
82      parser.on('error', (error) => {
83          console.error('Error en el puerto serie:', error);
84      });
85  }
86
87  run().catch(console.error);
```

Código 5: Código del cliente gRPC responsable del despliegue de datos en una GUI.

```
import time
import grpc
import protocolo_pb2
import protocolo_pb2_grpc
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib.widgets import TextBox, Button
import numpy as np

x_data = []
y_data = []
start_time = time.time()

def run():
    global x_data, y_data, start_time

    channel = grpc.insecure_channel('localhost:57000')
    stub = protocolo_pb2_grpc.DistanciaStub(channel)

    fig, ax = plt.subplots()
    ln, = plt.plot([], [], 'b', animated=True)
    ax.set_xlim(0, 20)
    ax.set_ylim(0, 20)
    ax.set_xticks(np.arange(0, 20, 2))

    ax.set_title("Distancia a un objeto")
    ax.set_xlabel("Tiempo [s]")
    ax.set_ylabel("Distancia [cm]")

    ax.grid(True)

    def init():
        ln.set_data([], [])
        return ln,

    def update(frame):
        global x_data, y_data

        rpc_request = protocolo_pb2.EmptyMessage()
        response = stub.Recibo(rpc_request)
        print("Recibido: %s" % response.solicitud)

        elapsed_time = time.time() - start_time
        x_data.append(elapsed_time)
```

```
45          y_data.append(float(response.solicitud))
46
47          x_data = [t - x_data[0] for t in x_data if t - x_data[0] ≤ 21]
48          y_data = y_data[-len(x_data):]
49
50          ln.set_data(x_data, y_data)
51          return ln,
52
53      ani = FuncAnimation(fig, update, frames=None, init_func=init,
54                          blit=True, interval=2000)
55
56      def update_y_limit(event):
57          ymin, ymax = ax.get_ylim()
58          new_ymin, new_ymax = float(input_ymin.text), float(input_ymax.text)
59          ax.set_ylim(new_ymin, new_ymax)
60          fig.canvas.draw()
61
62      plt.subplots_adjust(bottom=0.3)
63      axbox = plt.axes([0.1, 0.15, 0.1, 0.05])
64      input_ymin = TextBox(axbox, 'Y m n.:', initial='0')
65      axbox = plt.axes([0.3, 0.15, 0.1, 0.05])
66      input_ymax = TextBox(axbox, 'Y m x.:', initial='20')
67      axbox = plt.axes([0.5, 0.15, 0.1, 0.05])
68      button = Button(axbox, 'Actualizar')
69      button.on_clicked(update_y_limit)
70
71      plt.show()
72
73  if __name__ == '__main__':
74      run()
```