PLATAFORMA SOA DE SERVICIOS DE VISIÓN ARTIFICIAL

UNED

MIGUEL LUCAS LACAL

Contenido

Índice de figuras

Capítulo 1 - Introducción

La inteligencia artificial está experimentando un gran auge estos últimos años, con el desarrollo de nuevos algoritmos y la mejora hardware que los soporta, consiguiendo así resultados que años atrás eran inimaginables. Tanto es así, que están apareciendo grupos de investigación en empresas que no se dedicaban en un principio a la inteligencia artificial, para aplicar diferentes técnicas de IA en sus procesos empresariales. Estos son grupos altamente especializados que estudian el negocio y lo integran con diferentes modelos de inteligencia artificial para conseguir soluciones que hasta ahora requerían altos recursos tanto humanos como hardware o logísticos.

Existen diferentes áreas en el ámbito de la IA, y cada una de ellas resulta demuestra su utilidad resolviendo un tipo concreto de problemas. Un ejemplo es el procesamiento del lenguaje natural, con el que se pueden crear programas de reconocimiento de voz a través del estudio de la sintaxis, semántica y gramática de un idioma. Otro ejemplo son las técnicas de visión artificial (VA), que se pueden utilizar para detectar enfermedades en personas mediante radiografías o resonancias, para cartografiar regiones con precisión, o hasta segmentar imágenes satélite diferenciando zonas con alta vegetación de zonas más desérticas. En este trabajo nos centraremos en estas últimas, las técnicas que integran algoritmos de visión artificial en procesos de negocio.

Por otra parte, la ingeniería software continúa creciendo conforme se digitalizan y automatizan los procesos en las empresas. Todos los sectores están absorbiendo grandes cantidades de perfiles IT para llevar a cabo esta digitalización, tanto en el sector público como en empresas privadas. Entre los sectores más importantes encontramos salud, defensa y finanzas, además de otros más orientados al ocio como las redes sociales y el streaming de contenido audiovisual.

Tal demanda de sistemas informáticos exige una metodología, patrones y estandarización que asegure ciertas características en estos sistemas tan críticos para las empresas. Así nace SOA, la arquitectura orientada a servicios. El servicio es la unidad básica de SOA, y puede definirse como una tarea o proceso definido y asilado, que presenta su funcionalidad para que sea utilizada por un tercero. Los servicios se agrupan en inventarios de servicios, que sirven a las empresas para gestionar su ciclo de vida y darles visibilidad. Las arquitecturas orientadas a servicios se caracterizan por reducir el código redundante, facilitar la integración con otros sistemas, reducir tiempos de desarrollo y costes de mantenimiento entre otros beneficios.

Hablando desde el punto de vista empresarial, el concepto de aplicación cambia, y ahora las aplicaciones dejan de ser programas acotados e independientes. Las aplicaciones se convierten en conjuntos de servicios reutilizables que sirven a diferentes aplicaciones de la empresa, haciendo que el concepto de integración desaparezca, ya que respetando los principios SOA de interoperabilidad y componibilidad, las integraciones entre aplicaciones resultan triviales.

Aproximándonos a la temática de nuestro proyecto, los beneficios que aportan las arquitecturas orientadas a servicios aplicados a una serie de servicios de visión artificial (VA) agilizarían el desarrollo de soluciones en este campo de la inteligencia artificial. SOA aporta una arquitectura estandarizada y eficiente para sistemas con alta actividad, encapsulando diferentes funcionalidades en servicios reutilizables y haciendo posible gestionar su ciclo de vida e integraciones. Un ejemplo del rendimiento que podría obtener un equipo de desarrollo de VA surge a la hora de procesar una imagen por etapas: las diferentes fases de procesamiento podrían ser implementadas en servicios individuales y reutilizables, que se integrarían fácilmente asegurando la calidad y rendimiento del algoritmo. Otro uso que se podría dar a un sistema SOA en este campo podría ser la provisión de servicios de procesado de imágenes vía web, permitiendo que una aplicación tercera accediera a estos algoritmos implementados en un servidor con características adecuadas para estas tareas.

El objetivo de este proyecto es diseñar un sistema que implemente los beneficios de SOA en la creación de soluciones de visión artificial. Una arquitectura eficiente y estructurada con funciones adicionales a los servicios IA como autenticación y autorización, microservicios específicos para tareas que requieran condiciones especiales de rendimiento, y estandarización para los servicios generados en términos de seguridad, interoperabilidad y reutilización. Esta herramienta servirá a equipos de desarrollo de soluciones IA para agilizar sus labores de investigación y crear soluciones de mayor calidad en un tiempo menor, mejorando el trabajo colaborativo dentro del equipo y con otros equipos. Además, la implementación de estos algoritmos en una plataforma accesible vía web permitiría ofrecer estos servicios a todo tipo de clientes.

Para llevar a cabo este proyecto se ha trabajado en conjunto con un equipo de investigación y desarrollo (en adelante equipo de I+D) especializado en visión artificial, más concretamente especializado en soluciones dentro del campo de la cartografía, superresolución y segmentación de imágenes satélite. Este equipo de investigación ha colaborado en este proyecto aportando los requisitos de la herramienta en términos de experiencia de usuario (acceso, interfaz, etc.) y aportando los servicios o ejemplos de estos servicios alrededor de los cuales se construye la infraestructura SOA.

Capítulo 2 - Estado del arte

El modo de trabajo secuencial siempre ha tenido una gran presencia en el campo de la informática. En los inicios de la informática, cuando evolucionó de la electrónica, la computación se basaba en una sola máquina y un hilo de procesamiento que llevaban a cabo el proceso ejecución de manera centralizada y secuencial. Esta secuencialidad encajaba con las limitaciones técnicas del hardware del momento, pero pronto, a partir de los años ochenta, comenzaron a aparecer procesadores con arquitecturas capaces de ejecutar instrucciones de manera paralela, al igual que aparecieron las primeras redes LAN con decenas de equipos interconectados y velocidades de transferencia que llegaban al megabyte por segundo. En este momento se introdujo un nuevo paradigma de programación que revolucionaría la industria, la computación distribuida.

Esta serie de avances desembocaron en el desarrollo y expansión de internet, causando la aparición de sistemas con arquitecturas orientadas a proveer funcionalidad a terceros, de manera local o a través de la red, y rápidamente surgió lo que hoy conocemos como programación web. La programación web se estableció rápidamente como uno de los principales pilares de la industria informática y poco a poco se comenzó a gestar el concepto de **servicio web**, que era la evolución obvia de estos programas dedicados a proveer de funcionalidad a otros.

Las aplicaciones ya no eran completamente autocontenidas, si no que ahora hacían uso de funcionalidad implementada en aplicaciones terceras. Algunos ejemplos de funcionalidad provista por estos servicios web podrían ser: autenticación y autorización, almacenamiento, procesamiento, funcionalidad relacionada con el negocio… Realmente cualquier funcionalidad podía ser implementada como un servicio en una aplicación externa y utilizada a través de una red.

Con la popularización de estas arquitecturas web, la industria se vio en la necesidad de establecer convenios, acuerdos y estándares para facilitar las comunicaciones y hacer posible la interoperabilidad entre sistemas. Así nacieron diferentes protocolos y estándares como CORBA, HTTP o SOAP.

CORBA (Common Object Request Borker Arquitecture) es un estándar que permite la comunicación entre diferentes sistemas software que pueden estar localizados en diferentes lugares independientemente del lenguaje de programación o la plataforma. Nació para mejorar el desarrollo de aplicaciones distribuidas, y fue creado por el grupo OMG en 1991, aunque fue evolucionando en los años posteriores. Otro ejemplo similar a CORBA es DCOM (Distributed Component Object Model), de Microsoft, que en la actualidad ha sido sustituido por el framework .NET.

Cabe destacar la importancia de CORBA RPC (Remote Procedure Call) en el desarrollo de sistemas distribuidos y en arquitecturas cliente servidor, ya que utilizando llamadas remotas a procedimientos ejecutados en otras máquinas se simplificaba la paralelización de procesos y el desarrollo de aplicaciones distribuidas. Al mismo tiempo, Java desarrolló su propio sistema de invocación de procedimientos remotos: Java RMI (Remote Method Invocation).

Un gran ejemplo de protocolo nacido del desarrollo de la programación web es HTTP, que ha sido utilizado para establecer comunicaciones por internet desde 1991, con el lanzamiento de la World Wide Web (WWW), y ha venido siendo el protocolo web estándar de comunicación hasta ahora.

Junto con los estándares y protocolos, tenemos lenguajes que permiten definirlos y utilizarlos, como es el caso de XML (Extensible Markup Language). Nació como un metalenguaje para definir otros lenguajes de marcado y almacenar datos. Al igual que un metamodelo se utiliza para describir modelos (UML por ejemplo), un metalenguaje sirve para describir lenguajes.

El estándar más común hoy en día para definir servicios web es SOAP, creado bajo el metalenguaje XML. Este estándar permite definir las reglas de comunicación entre dos procesos de manera asíncrona. De la mano de SOAP y XML tenemos WSDL (Web Services Description Language), que describe la interfaz de un servicio web, tipo de datos, protocolo, servicios expuestos etc., para que estos servicios puedan ser descubiertos y utilizados.

Actualmente, no se puede hablar de servicios web sin mencionar una de las arquitecturas más relevantes para implementar servicios web: REST (Representational State Transfer). Esta arquitectura, se centra en describir interfaces de comunicación entre sistemas ligeros que utilicen principalmente HTTP.

A diferencia de SOAP, REST define a grandes rasgos la implementación de un sistema, haciendo énfasis en los recursos o entidades (objetos) con los que se trabajan, definiendo al menos una operación por cada uno de los métodos más importantes de HTTP: GET, POST, PUT, DELETE. Estos métodos HTTP nacieron a raíz de las operaciones básicas de una base de datos: CREATE, READ, UPDATE y DELETE.

2.1. SOA – Software Oriented Architecture

La necesidad de mantener las aplicaciones en un intercambio continuo de datos ha causado la creación de requisitos y estándares mínimos que deben cumplir las aplicaciones. Es por eso por lo que desde la aparición de los primeros sistemas distribuidos y servicios web se han ido desarrollando arquitecturas, metodologías, estándares y patrones como los que hemos comentado anteriormente. Se podría decir que esta evolución ha desembocado en lo que conocemos como arquitecturas orientadas a servicios, un tipo de sistemas que favorecen la reutilización, la componibilidad, el mantenimiento del código y la integración con otros sistemas.

Las arquitecturas **SOA** utilizan como unidad básica el **servicio**. Un servicio se puede definir como una tarea o proceso bien definido y asilado, que puede ser accedido o invocado por un tercero. Los servicios publicarán su funcionalidad en una interfaz de comunicación o API, siguiendo un contrato previamente establecido. Los servicios pueden agruparse en **conjuntos de servicios**, que ofrecen una funcionalidad compuesta más compleja. Una entidad puede agrupar los servicios que desarrolla en **inventarios de servicios**, con los que organizar y gestionar los servicios, monitorizando su ciclo de vida y haciendo que sean visibles y accesibles para otras aplicaciones.

Los servicios de una arquitectura SOA han de seguir unas pautas de diseño, implementación y mantenimiento para asegurar el objetivo de estas arquitecturas:

- Contrato de servicio estandarizado
- Desacoplamiento
- Abstracción
- Reutilización
- Autonomía
- Sin estado
- Visibilidad
- Componibilidad

SOA aporta beneficios al desarrollo software desde la fase de diseño, obligando a diseñar servicios alrededor de estándares de comunicación e implementación, disminuyendo el código redundante para facilitar la reutilización y componibilidad de los servicios. Siguiendo estas pautas de diseño, la integración se convierte en un trabajo trivial. Los trabajos de mantenimiento de código se aligeran considerablemente y el desacoplamiento potencia la autonomía y fiabilidad del servicio, reduciendo el índice de error y mejorando la disponibilidad del servicio.

2.1.1. Principios de diseño de servicios

A continuación, se enuncian y explican los principios básicos de diseño e implementación de servicios SOA.

Contrato de servicio estandarizado

El contrato de los servicios debe estar estandarizado para agilizar el desarrollo de los servicios, y hacerlos compatibles e interoperables con otros servicios.

Desacoplamiento

Los servicios deben depender lo menos posible de otros servicios para ofrecer su funcionalidad como una unidad. De esta manera cuando se reutilicen se evitarán dependencias que aportarían complejidad al sistema. Además, evitando dependencias se consigue una mayor autonomía.

Abstracción

El contrato de servicio solo contiene información esencial, y no existe más información acerca del servicio. Además, encapsulando funcionalidad se consigue el desacoplamiento mencionado anteriormente.

Reutilización

Los servicios ofrecen lógica agnóstica de manera que se convierten en recursos reutilizables.

Autonomía

Los servicios deben ofrecer un alto control sobre su entorno de ejecución, de manera que lleven a cabo su funcionalidad de manera consistente y fiable, sin depender de otros servicios.

Sin estado

El manejo de demasiada información relacionada con el estado de los servicios puede comprometer la disponibilidad de estos.

Visibilidad

Los servicios deben ofrecer metadatos para ser encontrados efectivamente.

Componibilidad

Los servicios deben ser reutilizables para poder ser compuestos con otros, y así ofrecer una funcionalidad compleja reutilizando unidades pequeñas.

2.1.2. Tipos de servicios

Según su conocimiento del entorno, los servicios se pueden clasificar en **agnósticos** y **no agnósticos**. Los primeros no contienen lógica relacionada con el negocio. Siguen fielmente los estándares definidos para la arquitectura SOA, por lo que son altamente reutilizables y componibles con otros servicios. Por otro lado, la funcionalidad de los servicios no agnósticos proviene de la lógica de negocio y normalmente sirven para lleva a cabo una funcionalidad concreta, por lo que no suelen ser reutilizables.

Según su relación con el resto del sistema los servicios se clasifican en los siguientes tipos:

Servicios de entidad

Servicios agnósticos. Están directamente relacionados con una o más entidades de negocio.

Servicios de tarea

Servicios no agnósticos que presentan lógica componible con otros servicios de tarea.

Servicios de utilidad

Servicios agnósticos derivados del análisis de negocio, como servicios relacionados con peticiones web o funciones de mensajería.

Microservicios

Servicios no agnósticos con poca reusabilidad y poco alcance. Su objetivo es llevar a cabo trabajos muy concretos con necesidades específicas de rendimiento, seguridad etc.

2.1.3. SOA en la actualidad

En la actualidad la orientación a servicios tiene un peso importante en la gran mayoría de empresas. Tanto entidades públicas como privadas utilizan servicios web en su infraestructura software.

En algunas ocasiones la implementación de estos servicios se lleva a cabo de una manera simple, para resolver una problemática concreta. Pero otras empresas construyen su infraestructura software y hardware con base en los servicios, consiguiendo sistemas completamente construidos alrededor de estos.

Para llevar a cabo estas implementaciones de arquitecturas orientadas a servicios las empresas utilizan herramientas que facilitan estos diseños y desarrollos, como puede ser el concepto de ESB (Enterprise Service Bus). Este componente gestiona la comunicación entre servicios web a través de mensajes, colas de mensajes, transformación de mensajes y enrutamiento, aplicación de estándares de comunicación, control de autorización y autenticación, balanceo de carga… en resumen, una **orquestación** completa de las comunicaciones entre los servicios y aplicaciones de la empresa.

Generalmente los ESB son integrables con diferentes tipos de aplicaciones, como Java o .NET, y utilizan XML para la estructura de los mensajes. Como ejemplos de implementaciones podemos mencionar Oracle Service Bus y OpenESB.

Además de orientación a servicios desde un punto de vista de sistemas backend, hoy en día muchos frameworks de frontend incluyen soporte para arquitecturas orientadas a servicios. Un claro ejemplo es Angular, que siendo un framework para el desarrollo frontal de aplicaciones tiene una arquitectura claramente orientada al consumo de servicios web, que encaja con un backend orientado a servicios. Las aplicaciones Angular tienen componentes dedicados explícitamente a gestionar peticiones a servicios web, y los componentes de interfaz de usuario hacen uso de estos primeros para enviar y recibir datos.

Junto con Angular encontramos Vue y React, que siguen arquitecturas parecidas, orientadas a consumir servicios web externos.

2.1.4. SOA en la empresa

Como ya hemos mencionado, las empresas llevan implementando arquitecturas orientadas a servicios desde hace tiempo. Para integrar SOA con la infraestructura existente en la empresa es importante entender los procesos de la entidad como un sistema interactivo sobre el que basar toda la actividad empresarial. Las empresas generan valor a través de procesos productivos, por lo que es importante crear una arquitectura software basada en estos procesos. Esta centralización en los procesos exige además un nivel alto de trazabilidad para relacionar cada proceso con objetivos u outputs concretos de la empresa. Esta trazabilidad y gestión es una pieza clave de la que no disponen muchas entidades, y en lo que a software se refiere ocurre lo mismo. Esto es lo que se denomina “Governance” y es un tema muy estudiado en la implementación de SOA.

Desde el punto de vista empresarial, el aspecto más relevante de una arquitectura orientada a servicios es la desaparición del concepto tradicional de aplicación. Ahora, las aplicaciones no son entidades independientes entre sí, si no que utilizan servicios reutilizables para completar su funcionalidad, servicios que también serán utilizados por otras aplicaciones dentro del inventario de servicios de la entidad. El concepto de integración también desaparece, ya que respetando los principios de interoperabilidad y componibilidad las integraciones entre aplicaciones resultan triviales.

Estas características de los sistemas SOA vienen soportadas por una pieza clave llamada **ESB**, que se encarga de soportar la comunicación entre los diferentes servicios empresariales. Hablaremos más delante de este tipo de software.

La creación de un **inventario de servicios** requiere una preparación y planificación importantes. Un inventario de servicios puede proporcionar soporte a toda la infraestructura IT de una empresa, reduciendo los tiempos de desarrollo y mejorando la calidad del software porque los servicios del inventario siguen las mismas pautas y estándares de diseño. La orientación a servicios aumenta considerablemente el ROI de la empresa, ya que al construir servicios reutilizables y componibles se genera código robusto y reutilizable que mejora los tiempos de desarrollo, de integración y de respuesta frente a cambios.

SOA - Ejemplos reales de aplicación SOA

Para entender mejor el impacto de las arquitecturas orientadas a servicios hablaremos sobre varios casos de éxito reales de compañías que han implementado principios SOA, ya sea en parte de sus sistemas o de manera transversal mediante una migración completa de su infraestructura IT.

Startwood Hotels & Resorts

Startwood Hotels & Resorts es una cadena hotelera neoyorkina que aglutina alrededor de once marcas diferentes y más de 150000 empleados en todo el mundo. En la actualidad, se asienta como una de las cadenas hoteleras más grandes y con mayores ingresos del mundo.

Una empresa de este tipo necesita una gran infraestructura IT para sostener todos los sistemas de gestión con los que opera, por eso el director global de tecnología en Startwood apostó por la evolución de sus sistemas a una orientación a servicios con XML SOA. Defienden esta decisión argumentando que un sistema orientado a servicios puede operar de manera global sin limitaciones de red o de centros de datos, e insisten en la necesidad de arquitectos de seguridad durante el diseño del sistema SOA para asegurar la integridad de la información sin comprometer el rendimiento de los sistemas.

Con esta migración de funcionalidad anticuada a un sistema SOA Starwood Hotels & Resorts ha creado una sólida infraestructura IT escalable y mantenible a lo largo del tiempo, que mantiene una inmensa cantidad de sistemas a lo largo de todas sus cadenas de hoteles.

First Citizens Bank

First Citizens Bank es una entidad bancaria estadounidense que ha crecido exponencialmente desde su fundación en 1898 convirtiéndose en una de las instituciones financieras más grandes de Estados Unidos.

La principal característica diferenciadora de First Citizens Bank es que, además de consumidora, también es proveedora de servicios de otras entidades financieras. En la actualidad, First Citizens Bank provee servicios tanto a sus clientes como a otras 20 instituciones financieras. Proveen servicios de procesamiento y almacenamiento de cheques, como aplicaciones de servicio de clientes a estas entidades externas. Para conseguir proveer servicios a tantas instituciones, First Citizens Bank necesita mantener una estructura orientada a servicios sólidamente diseñada, testeada, segura y documentada para que sea utilizada por terceros.

Whitney National Bank

La entidad financiera Whitney National Bank fundada en 1899 en Estados Unidos comenzó con un capital inicial muy bajo, pero rápidamente fue creciendo y expandiéndose a diferentes estados. En la actualidad es una de las entidades financieras más grandes del país norteamericano y cuenta con un gran sistema IT para soportar sus aplicaciones.

Stan Limerick, director de estrategia tecnológica de Whitney National Bank, comenzó la transformación SOA del banco enfrentándose a una serie de sistemas legacy y vio la oportunidad de crear un sistema para estandarizar y agilizar los servicios y procesos del banco. Su principal objetivo fue desacoplar lo máximo posible las aplicaciones entre sí pues hasta entonces la mayoría de las aplicaciones de la infraestructura IT del banco funcionaban con interfaces punto a punto, con unos niveles de escalabilidad y reutilización realmente pobres.

El principal beneficio de esta migración ha sido la competitividad que les ha aportado en el mercado, agilizando las diferentes operaciones de transacción y gestión, y convirtiéndose en un referente en el campo de los proveedores de servicios bancarios.

Administración pública

Una de las áreas en las que más se nota el crecimiento de la industria software es en el sector público. Aunque es cierto que las administraciones públicas normalmente van un paso por detrás del sector privado en cuanto a evolución tecnológica, en los últimos años estos procesos de digitalización han comenzado a ser notables.

En este ejemplo nos referiremos al sector de salud de manera general, y al crecimiento y digitalización que está experimentando.

Los sistemas IT de este sector manejan una cantidad de datos inmensa y de vital importancia, entre los cuales se encuentran datos personales de pacientes, citas, historiales médicos… Es por esto por lo que este sector lleva años absorbiendo una gran cantidad de perfiles IT para llevar a cabo este proceso de digitalización, entre los que destacan los perfiles relacionados con la arquitectura de sistemas y la seguridad, y el diseño y desarrollo de aplicaciones de gestión.

La necesidad de agilizar los procesos médicos y la importancia de mantener la integridad y seguridad de los datos ha supuesto, como decimos, un rápido crecimiento de este sector. Esto ha forzado a utilizar metodologías y patrones de diseño y desarrollo como SOA, brindando a los sistemas las características propias de este tipo de arquitecturas que hemos comentado en apartados anteriores.

Un ejemplo de esto es el Servicio Navarro de Salud (SNS), que cuenta con más de 170 perfiles IT en Pamplona para desarrollar y mantener los diferentes sistemas de este sector. El SNS posee una gran cantidad de aplicaciones y servicios, orientados tanto a los pacientes, como al personal sanitario navarro. Estas aplicaciones están en constante comunicación y sincronización, formando así una red de aplicaciones bien coordinada gracias a la aplicación de principios de diseño SOA.

Finalizaremos comentando un caso similar, de la administración pública en Navarra. Tracasa es una empresa pública que nació de la necesidad de digitalizar los procesos relacionados con el catastro en este territorio, y poco a poco ha ido creciendo y abarcando más áreas de trabajo, como el área judicial y el área de la recaudación de impuestos.

En la actualidad cuenta con una plantilla de más de 400 perfiles IT y administrativos. Hace ya unos años se diseñaron varios sistemas para soportar toda la infraestructura IT en previsión del crecimiento que iba a tener la empresa. Las aplicaciones desarrolladas se implementan en su mayoría con características de la orientación a servicios, creando así una red de procesos y servicios accesible para toda la organización.

Esta arquitectura permite a los diferentes equipos trabajar compartiendo datos y servicios, y promoviendo el desarrollo software de calidad y orientado a servicios a lo largo de la organización.

2.1.5. Software como servicio

La industria informática ofrece una gran variedad de productos y servicios, desde aplicaciones para el usuario general hasta herramientas específicas para resolver problemas técnicos muy concretos. Además de programas y aplicaciones, en el mercado encontramos otro tipo de productos que se alejan del usuario general, como servicios de hardware y sistemas para usuarios IT especializados. Esta variedad de servicios se organiza de la siguiente manera:

IaaS – Infraestructura como servicio

Empresas como Amazon o Microsoft han lanzado al mercado servicios conocidos como infraestructura como servicio, alquilando máquinas y elementos hardware a terceros a cambio de una cuota. Los servicios más comunes del tipo IaaS son servicios de virtualización para empresas, cortafuegos o hasta la creación de sistemas de procesamiento para entrenar modelos de inteligencia artificial.

Uno de los principales atractivos de estos servicios reside en la comodidad de encender una máquina a partir de una imagen anterior, de manera que en segundos es posible arrancar un sistema totalmente configurado con tecnologías como Docker o Kubernetes.

Estos servicios son realmente útiles por la flexibilidad y seguridad que aportan, ya que están disponibles para el usuario en cualquier momento y son altamente configurables en cuanto a potencia de hardware o tipo de máquina. Una gran ventaja de estos sistemas implementados en la nube es que en momentos en los que se necesita mayor potencia de procesamiento, existe la posibilidad de encender al momento máquinas que apoyen el trabajo en curso, aportando un alto grado de flexibilidad. Finamente hablaremos sobre la seguridad que aportan estos sistemas, ya que al encontrarse hospedados en la infraestructura de una organización IT ampliamente reconocida, estas nubes resultan en la mayoría de los casos más segura que en intranets construidas dentro de las propias empresas.

PaaS – Plataforma como servicio

Un paso más allá de la infraestructura como servicio son las plataformas como servicio. Estos servicios se centran en proveer plataformas ya configuradas donde desplegar aplicaciones. La principal diferencia con IaaS es que en PaaS la máquina suele tener preinstalado un sistema operativo, y el hardware alquilado es menos flexible, de manera que el usuario solamente tiene que preocuparse de configurar su aplicación.

Algunos ejemplos de aplicaciones que suelen desplegarse sobre estas plataformas podrían ser bases de datos o aplicaciones web o como plataformas para llevar a cabo integración continua de software, ya que los usuarios solo necesitan crear los paquetes a ser desplegados en el entorno en la nube, haciendo estos servicios realmente útiles a la hora de testear aplicaciones. Además, muchos de estos servicios de PaaS se encargan de monitorizar la carga añadiendo recursos cuando es necesario para asegurar el rendimiento de la aplicación.

Ejemplos reales de servicios de PaaS podrían ser Heroku o Google App Engine. Cabe destacar que entre los servicios provistos por Amazon o Microsoft de IaaS mencionados anteriormente también encontramos servicios de PaaS.

SaaS – Software como servicio

En este modelo lo que se ofrece es directamente un software final orientado al usuario. Se busca crear aplicaciones ya completas en la nube para que sean utilizadas por todo tipo de usuarios.

Las principales ventajas de estos servicios es la orientación al usuario final, pues no requieren procesos largos ni complejos de instalación (sin instalación en el caso de las aplicaciones web), además de la escalabilidad, estandarización en términos de experiencia de usuario y el soporte y actualizaciones por parte del equipo desarrollador. Cabe destacar que estas aplicaciones están destinadas normalmente a proveer servicios para usuarios sin conocimientos informáticos técnicos.

Ejemplos de SaaS pueden ser MicrosoftOffice 365, la suite análoga de Google, Canva, WordPress etc. Asimismo, la mayoría de las aplicaciones web que se encuentran en internet podría considerarse software como servicio, pues son aplicaciones funcionales que proponen un servicio al usuario final.

Como no puede ser de otra manera, todos y cada uno de estos servicios vienen con unas tarifas y precios de diferentes tipos. Comenzando por los servicios del tipo IaaS, encontramos principalmente dos tipos de contrato:

- Pago por uso – On demand: el precio varía en función del procesamiento o el tamaño de la memoria utilizada, con un coste base. Esta es la forma más eficiente y flexible de desplegar aplicaciones que necesiten escalar su potencia en momentos concretos.
- Pago por suscripción: el precio depende del tipo de hardware contratado y es constante mientras no se modifique el acuerdo.

Sobre los servicios PaaS encontramos unas tarifas muy parecidas a las comentadas para IaaS. La principal diferencia es el software preinstalado en las máquinas alquiladas (el sistema operativo, por ejemplo).

Finalmente, para los servicios del tipo SaaS encontramos tarifas variadas ya que cada servicio ofrecido puede diferir enormemente de los demás. La mayoría de estos servicios cuentan con una versión gratuita limitada (modelos de negocio freemium), o para que el usuario pruebe la aplicación antes de pagar.

La mayoría de estos servicios funcionan mediante suscripciones como Microsoft Office 365 o Canva, con diferentes planes, precios y funcionalidad, con lo que se reduce la barrera de entrada para nuevos clientes respecto a un pago grande y único al inicio como se podría venir haciendo en el pasado, y se consigue una mayor afiliación de clientes.

Pero también encontramos algunos servicios completamente gratuitos como las redes sociales (aunque hay división sobre si son SaaS o no) o aplicaciones como Uber o Glovo. Estas aplicaciones obtienen ingresos gracias al flujo de usuarios y a su información, y a través de anuncios y tasas a los usuarios. En el caso de Uber y Glovo la mayor parte de los ingresos provienen de unas tasas que se cobra a los usuarios por uso.

Importancia y proyección de SaaS

El mercado software ha evolucionado rápidamente, desde los primeros ordenadores y aplicaciones de escritorio hasta ecosistemas de servicios totalmente implementados en la nube. Haciendo un análisis del mercado software actual desde el punto de vista del usuario, hay una clara prevalencia de aplicaciones cuyo modelo de negocio es el de software como servicio. Un claro ejemplo es que empresas conocidas como Microsoft o Adobe migraron varios de sus modelos de negocio software hace ya tiempo.

Las aplicaciones de escritorio quedaron obsoletas hace años y el usuario ya está bien acostumbrado a pagar por un servicio software confiable y fácil de usar. Esto junto con la infraestructura software y hardware existente para soportar este tipo de servicios las empresas continuarán con estos modelos de negocio en el futuro ya que están resultando ser altamente lucrativos.

Concluiremos comentando que el sistema que hemos desarrollado está claramente altamente orientado a la provisión de servicios en la nube siguiendo un modelo SaaS. El objetivo a corto plazo no es ofrecer un servicio de suscripción si no crear una infraestructura de servicios para mejorar el rendimiento de los equipos de desarrollo de soluciones IA dentro de la empresa, pero en un futuro sería posible escalar el sistema para llegar más lejos de la estructura empresarial actual.

2.1.6. Governance

Uno de los principales problemas de los sistemas software actuales es la falta de gobernabilidad, concepto también conocido como gobernanza o “governance”.

La gobernabilidad podría definirse como un proceso de gestión y control a alto nivel de los servicios para asegurar su estandarización y diseño, visibilidad y objetivos. La gobernabilidad permite mantener un entorno orientado a servicios controlado a la vez que flexible.

SOA hace especial énfasis en este aspecto del proceso de la transformación. Muchas empresas ya cuentan con una amplia serie de servicios repartidos por sus aplicaciones y sistemas, pero generalmente fueron diseñados de manera individual para cumplir con tareas específicas y no encajan con la filosofía SOA.

El primer paso para aplicar gobernabilidad en una arquitectura de servicios, y posiblemente el más costoso, es hacer un análisis de la infraestructura IT completa de la entidad recolectando y catalogando servicios, y reflejándolos en un repositorio de servicios para integrarlos en el ciclo de vida de los servicios SOA. Muchas empresas afirman que, tras aplicar principios de diseño SOA y trabajar con estas directrices durante un tiempo, el aspecto más importante en una arquitectura SOA es la gobernabilidad.

En la aplicación de SOA al mundo empresarial se han desarrollado técnicas de control de políticas, roles, identificación y permisos que resultan imprescindibles en sistemas SOA complejos, como se muestra en diferentes implementaciones de ESB y en el estudio “Context Model Based SOA Policy Framework” [1] del Laboratorio de Desarrollo de IBM en China.

Una vez comenzada la transformación SOA es importante mantener una cultura y organización orientada al mantenimiento de la infraestructura, ya que SOA no es un objetivo, si no que es un camino hacia una infraestructura software de calidad mantenible en el tiempo.

Las arquitecturas orientadas a servicios aportan beneficios a grandes y pequeñas entidades como los que hemos comentado en secciones anteriores: autonomía, desacoplamiento, estandarización… Lo que se traduce en sistemas que cumplen con las demandas de los clientes relativas a QoS, eficiencia, adaptabilidad frente a cambios en el negocio y seguridad e integridad.

A parte de los beneficios de SOA, una transformación de este tipo conlleva una serie de retos a superar, como la integración del código antiguo o “Code legacy” que explicamos a continuación.

2.1.7. Legacy Code

La transformación SOA genera una gran cantidad de servicios basados en implementaciones legacy por la cantidad de software que han generado las empresas a lo largo de los años. Con SOA el camino a seguir es analizar y modularizar esta funcionalidad legacy para adaptarla y encapsularla en forma de servicio autónomo, siguiendo los estándares de los demás servicios.

Aun así, existe la posibilidad de encontrar código desfasado que no cumpla con los mínimos de rendimiento, seguridad o interoperabilidad, en cuyo caso es necesario rehacer el servicio por completo.

Para combatir los problemas de interoperabilidad propios de los sistemas legacy, SOA propone el uso de estándares para diferentes partes del proceso. Los servicios adaptados de código antiguo se encapsularán y utilizarán los mismos mecanismos y lenguajes que los servicios nuevos, de manera que desde fuera ambos tipos de servicios serán indistinguibles y se crearán contratos para que cada servicio sea visible y accesible. Además, se seguirán unas pautas de diseño e implementación para los nuevos servicios que aseguren los principios SOA (autonomía, abstracción, seguridad, componibilidad…).

En resumen, el objetivo es crear un entorno de servicios estandarizados, componibles e interoperables y visibles para el resto de los servicios.

El primer paso para implementar SOA en sistemas existentes es la aceptación de heterogeneidad en la infraestructura IT de la organización. En la investigación de Liam O'Brien, Paul Brebner y Jon Gray [2] se mencionan una serie de pasos y aspectos a tener en cuenta para llevar a cabo migraciones de sistemas legacy a SOA, desde la definición y descubrimiento de candidatos a servicio hasta su posterior integración y orquestación, todo ello aplicado a su caso particular.

En nuestro proyecto no consideramos se esté aplicando una migración al uso, si no que el objetivo es crear una infraestructura de servicios desde cero utilizando como base el flujo de trabajo del equipo de científicos de datos con el que se ha desarrollado el proyecto.

Sin embargo, consideramos realmente interesante que la investigación mencionada anteriormente hace especial énfasis en el rendimiento y escalabilidad del sistema SOA. Es aquí donde comentan que la orquestación de servicios puede realizarse de diferentes maneras, pero que han elegido aligerar esta parte al máximo ya que los procesos de orquestación sobrecargan en gran medida los sistemas distribuidos.

Podemos decir que estamos aplicando una filosofía similar en nuestro sistema, ya que, como se detallará en secciones posteriores, la orquestación se lleva a cabo de manera ligera y simple. Por otro lado, se ha optado por utilizar el patrón REST sobre HTTP para el ESB y las comunicaciones entre servicios, para cumplir con nuestros objetivos de escalabilidad y agilidad.

2.2. IA - Inteligencia artificial

Existen multitud de definiciones y concepciones de lo que es la inteligencia artificial y lo que llegará a ser por la velocidad a la que se ha desarrollado en las últimas décadas y las diferentes aplicaciones que ha tenido, pero para este trabajo la definiremos como una imitación de la inteligencia humana: un proceso tal que si lo llevara a cabo una persona concluiríamos que es inteligente.

La inteligencia artificial es una de las disciplinas técnicas más jóvenes a la vez que más antiguas. Desde diferentes pensadores de la antigua Grecia hasta distintos filósofos medievales se han ido perfilando las bases del pensamiento formal y las reglas del razonamiento avanzando paso a paso hacia lo que hoy conocemos como computación, pero los mayores avances en inteligencia artificial y su industrialización han ocurrido en las últimas décadas.

Uno de los precursores más importantes de la computación y la informática moderna fue Alan Turing, quien formalizó los conceptos de algoritmo y computación con aportaciones como la famosa “Máquina de Turing”, la construcción de uno de los primeros computadores electrónicos programables digitales o “La tesis de Church-Turing”.

Avances como los de Turing y otras personalidades como Hebert Simon o Allen Newel trajeron de la mano una serie de creaciones y avances como diferentes lenguajes de programación y herramientas que apoyaban el desarrollo en este campo, como ILP-11 y Logic Theorist (en 1955 y 1956 por Hebert Simon, Allen Newell y J. C. Shaw), el lenguaje LISP (en 1958 por John McCarthy) y SmallTalk (en 1980 por Alan Kay) o el GPS (General Problem Solver, en 1957 por Newell y Simon).

A partir de los años 60 se sucedieron avances más conocidos en la actualidad como los sistemas expertos, redes semánticas, redes neuronales o algoritmos famosos como el conocido Deep Blue.

En la actualidad el desarrollo en el campo de la inteligencia artificial avanza de manera vertiginosa, con aplicaciones variadas como asistencia técnica a usuarios, algoritmos de recomendación, toma de decisiones autónoma o hasta detección de patrones y comportamientos. Multitud de empresas han abrazado este potencial y han comenzado procesos de evolución en los que implementan diferentes técnicas de IA en sus procesos productivos.

2.2.1. IA y Big Data

El avance de la tecnología ha resultado en la generación de una cantidad masiva de datos por parte de usuarios individuales, entidades gubernamentales, empresas privadas etc. Cuando se llega al punto en el que almacenar y procesar tal cantidad de datos resulta inviable con las técnicas software convencionales es cuando entra en acción el big data.

Es entonces cuando se dejan de tratar los datos de manera individual y se comienzan a desarrollar métodos y sistemas que procesan estas cantidades de datos a lo grande, para detectar patrones, comportamientos y tendencias a grandes rasgos.

Los sistemas de inteligencia artificial se basan en gran medida en la explotación de cantidades masivas de datos, que pueden recopilarse de distintas maneras. Estas cantidades de datos sirven a los sistemas de IA para "aprender", de manera similar a como lo haría un ser vivo. Hay dos maneras de entrenar una máquina, de manera supervisada y de manera no supervisada. El aprendizaje supervisado consiste en proporcionar a la máquina datos previamente clasificados, para que aprenda las características que definen cada etiqueta o categoría. Por otro lado, el aprendizaje no supervisado se basa en un algoritmo o métrica para determinar el patrón que siguen los elementos de una clase.

2.2.2. IA en la práctica

A continuación, comentamos algunas de las aplicaciones más exitosas de inteligencia artificial en el proceso productivo empresarial y otros ámbitos prácticos.

Chatbots y asistentes personales

Los chatbots son aplicaciones que simulan una conversación con una personal real, normalmente a través de texto. Estas aplicaciones software nacieron en los años 60 y han seguido perfeccionándose hasta el día de hoy, resultando en ocasiones difícil distinguir si realmente se trata de una conversación con una máquina o con una persona.

En el mundo empresarial los chatbots sirven para proporcionar ayuda a usuarios con problemas técnicos concretos, haciendo de "buscador" de soluciones utilizando las palabras que proporciona el usuario y devolviendo respuestas ya preparadas. Otro ejemplo conocido de chatbot podría ser los asistentes personales, como el asistente de iOS: Siri.

Sistemas de recomendación y buscadores

Este tipo de software se basa en la creación de perfiles de usuarios a partir de grandes cantidades de datos para distinguir grupos de usuarios con preferencias o necesidades similares. Las plataformas de streaming de video como Netflix o HBO son un gran ejemplo de puesta en práctica de estos algoritmos. Las plataformas de compras como Amazon también utilizan intensivamente estas técnicas.

La explotación masiva de datos no solo se utiliza actualmente para fines lucrativos, sino que también resulta de utilidad en otros campos como la ciberseguridad, la salud o el cambio climático.

Ciberseguridad

De manera similar a cómo se reconocen patrones de consumo en usuarios, se han desarrollado sistemas que estudian el tráfico en las redes internas de las empresas, identificando tipos de ciberataques y los neutralizándolos antes de que puedan causar daños.

Salud

En el ámbito de la salud cabe destacar la utilización de estos sistemas en la detección de posibles enfermedades que puedan contraer los pacientes, con datos de enfermedades pasadas, historial genético o hábitos en alimentación.

2.2.3. Visión artificial - VA

Una de las áreas más importantes en el mundo de la IA es la visión artificial. Este campo pretende enseñar a los ordenadores a entender el contenido de las imágenes y el vídeo digitales, tal y como lo haría una persona, y a tomar decisiones con base en ello.

Una de las técnicas más utilizadas para aplicar visión artificial son las redes neuronales, que están basadas en la unión de neuronas individuales conectadas con enlaces con un valor de peso que se calcula automáticamente gracias a una función de pérdida prestablecida. Las redes neuronales también se usan en otros tipos de aprendizaje automático como los sistemas de reconocimiento de voz. Un grupo de técnicas dentro del aprendizaje automático que destaca resolviendo problemas de visión artificial son los algoritmos dentro del grupo conocido como “deep learning” o aprendizaje profundo.

Hoy en día no es necesario desarrollar desde cero algoritmos para aplicar visión artificial pues estos ya han sido desarrollados y optimizados en diferentes librerías, por lo que un desarrollador de VA se centra en resolver problemas concretos con los algoritmos que mejor se adapten a cada escenario. Algunas librerías conocidas son OpenCV, TensorFlow o CUDA.

Entre las industrias que más aprovechan la visión artificial encontramos la siguientes:

Automoción

Los vehículos con sistemas de detección de colisiones o de auto pilotaje llevan incorporados una serie de cámaras cuyas imágenes se procesan al momento para detectar calles, peatones y obstáculos.

Seguridad

El mejor ejemplo de la seguridad que puede aportar la VA son los sistemas de reconocimiento facial utilizados tanto en dispositivos móviles corrientes como en las cámaras de seguridad de aeropuertos o zonas que requieren un control exhaustivo.

Salud

De nuevo, en el campo de la salud se vuelve a hacer uso de técnicas de inteligencia artificial, en este caso, de manera más concreta, con visión artificial podemos comentar cómo se utilizan miles de imágenes de enfermedades (como radiografías) para relacionarlas con imágenes tomadas de pacientes, llegando a superar en porcentaje de acierto a expertos en la detección de diferentes tipos de enfermedades como el cáncer y los melanomas.

2.2.4. Accesibilidad a la IA y formación

La revolución que ha supuesto la inteligencia artificial en diferentes industrias ha desarrollado la necesidad de crear puestos de trabajo para perfiles expertos en estas disciplinas. Multitud de empresas han creado equipos de científicos de datos, matemáticos, arquitectos software y programadores con el objetivo de optimizar sus procesos productivos a través de técnicas de inteligencia artificial. Es por esto por lo que viene creciendo rápidamente el número de aprendices y maestros, y con ellos diferentes plataformas que apoyan la formación en inteligencia artificial y big data.

Universidades y academias se han adaptado en las últimas décadas a estos intereses de la industria y se han creado especializaciones, cursos y maestrías enfocadas en inteligencia artificial.

De la misma manera, la formación autónoma ha crecido de manera exponencial. En la actualidad se puede encontrar infinidad de contenido didáctico e industrial, documentos de investigación, charlas de expertos y tutoriales sobre cualquier área de la inteligencia artificial. Y lo más remarcable es que gran parte si no la mayoría de este contenido es gratuito y accesible por prácticamente cualquier usuario.

Los promotores más influyentes son las megacorporaciones tecnológicas como Google, Amazon o Microsoft, que en sus propias páginas webs ofrecen formación gratuita en este campo. Estas organizaciones se benefician enormemente de formar expertos en el tema, ya que son los que más beneficio obtienen del estudio y la explotación de estas cantidades masivas de datos.

Un ejemplo del impulso que dan las megacorporaciones al aprendizaje y la formación son las plataformas y recursos orientados al desarrollo y aprendizaje como Google Colab, un espacio en la nube donde programar y ejecutar código muy demandante utilizando la potencia de procesamiento de servidores de Google.

Otro ejemplo es la plataforma: https://ai.google/education , donde se pueden encontrar materiales didácticos de toda clase, desde contenido teórico como libros y documentos de investigación hasta podcasts y cursos prácticos.

2.2.5. Legislación

Queda clara la importancia de la inteligencia artificial tanto en el mundo académico y de la investigación como sus aplicaciones en diferentes industrias. Por el impacto que supone de manera general, hace ya varios años que se empezó a crear legislación para establecer límites y asegurar que esto no causa problemas, desbalances o injusticias en la sociedad.

Uno de los principales motivos para crear legislación para regular la aplicación de inteligencia artificial es la opacidad que presentan algunos algoritmos. Técnicas basadas en redes neuronales o lógica difusa tienen una naturaleza que, aunque se haya estudiado y haya demostrado los beneficios que aportan, conllevan un porcentaje de incertidumbre difícil de calcular o predecir, además de la dificultad que presenta llevar una trazabilidad de las decisiones que se han tomado hasta obtener el resultado del algoritmo.

Por otro lado, los sistemas de identificación biométricos han demostrado una gran fiabilidad y los implementan dispositivos móviles para desbloquear el dispositivo o empresas para realizar fichajes. Sin embargo, siempre hay un pequeño porcentaje de error por el que puede darse un falso positivo, por lo que la industria se muestra en ocasiones reticente a implementar estas aplicaciones biométricas en sistemas críticos.

En la actualidad, industrias como la de la ciberseguridad y la bancaria hacen un uso intensivo de técnicas de inteligencia artificial. Como hemos comentado anteriormente, se utilizan algoritmos de reconocimiento de patrones para detectar con antelación posibles ciberataques, y, de la misma manera, los bancos utilizan estas técnicas para detectar clientes morosos o predecir cambios en la economía. Sin embargo, este último ejemplo es realmente polémico. La toma de decisiones basada en algoritmos de inteligencia artificial ha demostrado un gran porcentaje de acierto (mayor que el humano), pero se siente realmente arriesgado dejar esta toma de decisiones en manos de un algoritmo cuando se trata del futuro de un banco, sobre el que pueden estar apoyadas multitud de empresas o incluso la economía de varios países.

Como ejemplos de las medidas tomadas desde el punto de vista jurídico para regular la aplicación de la inteligencia artificial, podríamos decir que la mas conocía es el Reglamento General de Protección de Datos, o RGPD, un reglamento europeo aplicado a todos los países de la Unión Europea que regula el tratamiento y circulación de los datos personales de las personas físicas. Podemos mencionar también directivas europeas como la Directiva 2019/1024 que regula la utilización de datos y documentos conservados por entidades públicas o con capital público, o la conocida ley de cookies que regula el uso de cookies alrededor del consentimiento y la transparencia hacia el usuario.

Otro escenario donde la visión artificial se vuelve especialmente útil es con la videovigilancia. Mediante el procesamiento de los fotogramas de un video un sistema de VA puede reconocer formas, acciones y hasta rostros si la definición de la imagen es suficiente. En este campo hay que tener en cuenta la legislación existente sobre grabación de zonas públicas y la videovigilancia privada, ya que leyes como la "*Instrucción 1/2006"* o la misma "*Ley Orgánica 3/2018 de Protección de Datos Personales y garantía de los derechos digitales"* restringen y concretan las situaciones en las que esta permitidas estas prácticas relacionadas con la grabación y uso de imágenes y vídeo.

Como argumento final para volver a destacar peso de la inteligencia artificial en la industria y la consecuente necesidad de regulación, comentaremos la creación de organizaciones internacionales como el Comité Europeo de Inteligencia Artificial y El Libro Blanco de la Inteligencia Artificial, ambos con el objetivo de perseguir el beneficio de las personas y la creación de un ecosistema regulado de confianza.

2.3. Herramientas existentes

Dado la agilidad en el desarrollo de nuevos productos que presenta el mercado software, encontramos infinidad de tipos de aplicaciones y herramientas. Las más conocidas son normalmente las aplicaciones orientadas al usuario final, que ofrecen servicios de todo tipo y no requieren conocimientos informáticos técnicos (aplicaciones de compraventa, de entretenimiento como videojuegos o streaming de video, información como blogs o periódicos…).

Por otro lado, encontramos herramientas creadas para satisfacer necesidades de perfiles técnicos dentro de la industria software. Dentro de esta gran categoría de aplicaciones las más comúnmente usadas son los frameworks de desarrollo y editores de texto como NetBeans y VisualStudio, el software de gestión de recursos y despliegues como Jenkins, o software de gestión de la configuración como Git.

Introduciéndonos en el campo de la inteligencia artificial encontramos herramientas que apoyan la experiencia de desarrollo como Anaconda, CUDA o el proyecto Jupyter, otras con características más instructivas que apoyan la creación de perfiles de ciencia de datos como Google Colab, y se podría incluso considerar herramientas de desarrollo librerías de código como NumPy o Pytorch.

Por último, mencionaremos las herramientas orientadas a la creación de contenedores como Docker o Kubernetes. Estos contenedores en su interior contienen aplicaciones o sistemas encapsulados, autónomos y autocontenidos, ya configurados y listos para desplegar. Profundizaremos en este tipo de herramientas más adelante ya que uno de los objetivos de este proyecto es la creación de un contenedor para el despliegue del sistema que hemos diseñado.

Nuestro trabajo se sitúa en el grupo de software de gestión de recursos y desarrollo software. Existe una amplia variedad de aplicaciones para gestionar el trabajo como Trello o Jira, herramientas de automatización de procesos como Jenkins o software para gestionar la comunicación entre procesos y sistemas como las herramientas del tipo ESB, que son las que más nos interesan en este trabajo y de las que hablaremos a continuación.

2.3.1. Bus de servicios Empresariales

Concretamente, nuestro sistema encaja en este último grupo de herramientas, pues el objetivo principal es enlazar diferentes servicios para crear composiciones de servicios más complejas. Es por esto por lo que hablaremos de la arquitectura software orientada a la gestión de servicios web por excelencia: el Bus de Servicios Empresariales (ESB), que se ha venido utilizando en proyectos que perseguían una arquitectura orientada a servicios.

Un sistema que incluye un ESB para soportar las comunicaciones entre sus diferentes unidades propicia seguridad, trazabilidad y estandarización a los diferentes componentes y al sistema de manera general, entre otras características. Estas arquitecturas son además altamente escalables y componibles gracias al sistema de comunicación que implementa un ESB.

El ESB puede considerarse la piedra angular de una arquitectura orientada a servicios (SOA) ya que todas las organizaciones que han optado por migrar sus sistemas monolíticos hacia un enfoque de servicios lo han implementado, y tenemos como ejemplos las organizaciones de las que hemos hablado en secciones anteriores. Las principales características que aporta un ESB son:

- Enrutamiento
- Transformaciones y mapeo
- Orquestación de servicios
- Seguridad: cifrado, transacciones, autenticación y autorización…
- Monitorización, trazabilidad y auditoría
- Visibilidad de los servicios

Cada implementación del bus de servicios empresariales puede ser completamente diferente a la anterior, ya que cada empresa tiene unas necesidades diferentes. Algunas de las opciones más conocidas del mercado son Oracle Service Bus, IBM Integration Bus y OpenESB.

Adicionalmente mencionaremos que es muy común encontrar implementaciones de software de **gestión de colas de mensajes** en los ESB de las empresas, como RabbitMQ, ApacheKafka o Spark Streaming. De manera muy resumida, este tipo de software se puede entender como un middleware de mensajería, y supone un apoyo enorme en sistemas que tienen que soportar altas cargas de tráfico de datos y de mensajes.

2.3.2. Orquestación de servicios

Habiendo hablado sobre arquitecturas orientadas a servicios no se puede dejar de lado el software de orquestación de servicios. Este tipo de software se utiliza normalmente en sistemas implementados en la nube y se aplican usualmente con aplicaciones del estilo Kubernetes o Docker.

La orquestación de servicios consiste en la creación de una arquitectura con la que coordinar y gestionar servicios independientemente de dónde o cómo estén implementados. Su eficacia se basa en la abstracción que reside en la definición y encapsulación del servicio. Esta filosofía de diseño hace que los servicios se diseñen no solo como un proceso que devuelve un resultado, si no poniendo énfasis en la escalabilidad y optimización del proceso para cumplir con el objetivo de negocio.

Algunos ejemplos de software de orquestación de servicios son Google Kubernetes Engine, Nomad o Amazon ECS.

2.3.3. Governance y el inventario de servicios

Hemos explicado el significado de Governance y su importancia a la hora de crear un inventario de servicios para la entidad en la implementación de una arquitectura SOA.

En cuanto a las herramientas orientadas a dar soporte a este aspecto de SOA comentaremos que el software del estilo ESB o del tipo orquestador de servicios siempre incluye características que aportan gobernanza, pues es un aspecto indispensable a la hora de registrar, enrutar, mapear y monitorizar servicios.

Estas herramientas sirven de inventario de servicios al mismo tiempo, dado que es en ellas donde ocurre el registro y la agrupación de servicios, y su posterior gestión del ciclo de vida.

2.3.4. Herramientas en el mercado

En este apartado hemos recopilado información sobre diferentes tipos de herramientas que se utilizan hoy en día en el diseño y desarrollo de soluciones SOA, ya sea formando parte de la implementación de la arquitectura como aplicaciones y herramientas que no forman parte de la implementación del sistema, como es el caso de las aplicaciones de gestión del ciclo de vida de los servicios o las herramientas que hacen de middleware para el intercambio de mensajes.

a. Infraestructura para mensajería

Este tipo de herramientas implementan funcionalidad relacionada con el intercambio de mensajes entre aplicaciones. La característica principal de este software es que actúa como middleware entre sistemas heterogéneos y hace posible la comunicación entre ellos. Normalmente utilizan un protocolo de comunicación que deberán respetar las aplicaciones que utilicen la herramienta de mensajería, dejando libre el resto de la implementación.

Otra característica importante para los sistemas SOA es la capacidad que brindan este tipo de aplicaciones de aportar estandarización al sistema. Cualquier comunicación que pase a través de este middleware es susceptible de seguir el estándar marcado. Esta estandarización abarca desde la definición del protocolo de comunicación (SOAP con WSDLs, REST sobre HTTP…) hasta la estructura interna de los mensajes intercambiados.

Esta estandarización sirve también para aplicar patrones de seguridad en cada intercambio de mensajes, a través de los protocolos y la estructura que se aplican a las comunicaciones.

La trazabilidad que presenta el sistema completo se ve altamente potenciada cuando se utiliza un middleware de intercambio de mensajes. Cuando los mensajes viajan siguiendo el mismo protocolo y formato se simplifican las tareas de detección de errores y registro de actividad.

A continuación, comentamos diferentes ejemplos de herramientas de este tipo:

IBM App Connect Enterprise

- Plataforma de integración de software de IBM.
- Protocolos SOAP, HTTP, JMS.
- Integrable con multitud de sistemas.
- Diferentes protocolos de seguridad: LDAP, X-AUTH, O-AUTH…

RabbitMQ

- Código abierto.
- Implementa el protocolo estándar de enrutamiento y encolamiento AMQP.
- Protocolos HTTP, XMPP, STOMP.

WSO2

- Código abierto.
- Integración de interfaces de comunicación entre aplicaciones heterogéneas y con interfaces de comunicación ligeras.
- Integración ágil para aplicaciones con interfaces de comunicación ligeras.

b. Gestión del ciclo de vida de servicios – Governance

Las herramientas del tipo Governance aportan diferentes medios para gestionar el inventario de servicios, por lo que es necesario que exista previamente una infraestructura orientada a servicios implementada.

La gobernanza puede llevarse a cabo de diferentes maneras. Por ejemplo, en nuestro caso, como veremos más adelante, hemos optado por diseñar el proceso de gestión del ciclo de vida del servicio como una aplicación propia, la aplicación web de gestión. Por otro lado, al publicar todos nuestros servicios en forma de web APIs hemos utilizado Swagger para generar documentación y aportar una visión dinámica del estado del inventario.

No existen multitud de herramientas que cubran solo la parte de gobernanza de un sistema SOA porque en la mayoría de los casos se implementan como funcionalidad complementaria de las aplicaciones que implementan el ESB. A continuación, mencionamos algunas de estas herramientas orientadas a la gestión del ciclo de vida del servicio:

- WSO2 Governance Registry: complementaria de WSO2.
- CentraSite
- Consul

c. Implementaciones de ESB

Las herramientas y aplicaciones que consisten en implementaciones completas del bus de servicios empresariales son plataformas completas y enfocadas al usuario final desarrollador de SOA, altamente configurables para adaptar el bus a diferentes tipos de sistemas.

Este tipo de herramientas normalmente están desarrolladas por grandes compañías software que las venden como productos finales. Son aplicaciones realmente potentes y robustas, ya que son el pilar fundamental sobre el que se construye toda la arquitectura SOA.

Entre otras funciones, estas herramientas aportan una amplia variedad de características SOA como son la gobernanza sobre el ciclo de vida de los servicios del inventario, escalabilidad, estandarización de mensajes, trazabilidad y registro de incidencias. La mayoría de estas herramientas incluye infraestructura para mensajería y gestión del ciclo de vida del servicio o gobernanza.

A continuación, comentamos algunos de los ejemplos más importantes de este tipo de herramientas.

Oracle Service Bus (OSB)

- Implementación de ESB de Oracle.
- Integrable con diferentes intermediarios de mensajes (message brokers).
- Definición y registro de servicios web mediante WSDLs.

IBM Websphere ESB

- Implementación de ESB de IBM.
- Utiliza WebSphere MQ como middleware de mensajería.
- Integrable con diferentes tipos de bases de datos y aplicaciones de transporte de datos.

JBoss ESB

- También conocida como JBoss Enterprise SOA Platform. Implementación de ESB de código abierto basada en Java EE.
- Multiplataforma. Funciona con cualquier SO que utilice Java.
- Utiliza HornetQ como middleware de mensajería.

Como vemos, existen diferentes tipos de herramientas en el mercado para facilitar el diseño de sistemas SOA. En función de sus necesidades, el desarrollador puede optar por utilizar herramientas de mensajería y gobernanza por separado para conseguir una implementación SOA más transparente y configurable, o utilizar herramientas completas que implementen el ESB con toda la funcionalidad comentada previamente.

Las aplicaciones orientadas a proporcionar infraestructura como mensajería son interesantes para implementaciones no solo de sistemas SOA, si no de cualquier tipo de sistema distribuido que necesite gestionar cierto tráfico de datos de una manera ordenada, segura y robusta. El concepto de cola de mensajes y los protocolos y algoritmos para gestionarlas cobra especial importancia en la construcción construir sistemas distribuidos eficientes y robustos.

Una vez implementados estos sistemas los mensajes empiezan a fluir de manera automática siguiendo los protocolos y estándares establecidos, que deben ser respetados por las aplicaciones del sistema. Como contraparte, el mantenimiento de estas herramientas puede resultar ser una tarea exigente en cuanto a tiempo y experiencia, ya que en ocasiones se convierten en cajas negras donde es complicado aplicar trazabilidad y monitorizar el flujo de información.

Por otro lado, las herramientas que gestionan el ciclo de vida de los servicios y aplican gobernanza también pueden ser utilizadas en sistemas que no siguen patrones SOA, ya que los servicios gestionados pueden ser completamente independientes y seguir sus propios procesos de mantenimiento y ciclo de vida.

Sin embargo, este tipo de herramientas se vuelven imprescindibles en implementaciones SOA por la necesidad de estandarizar tanto el diseño de los servicios como su mantenimiento e integración con los demás servicios del inventario.

Finalmente, como ya hemos comentado, las herramientas más completas son las diferentes implementaciones de ESB que se encuentran en el mercado. Estas herramientas en la mayoría de los casos implementan una infraestructura que cubre la funcionalidad de los dos tipos de herramientas, implementando el software necesario para hacer funcionar una infraestructura SOA completa.

Como veremos, en nuestro caso hemos realizado una implementación completa desde cero tanto del ESB como de los sistemas de mensajería y gobernanza, utilizando diferentes tecnologías y herramientas, consiguiendo un sistema propio altamente adaptable y modular.

Capítulo 3 – La plataforma

3.1. Objetivos transversales

Durante el desarrollo, uno de los objetivos ha sido la creación de una plataforma potente y fácil de mantener y utilizar. Hemos aplicado diferentes técnicas de modularización, como la arquitectura basada en componentes que propone angular para la parte frontal, o la orientación al consumo de servicios web.

A esto se le suma la orientación a servicios que se consigue a través del backend compuesto por el ESB y el host de microservicios, creando una infraestructura donde implementar un inventario de servicios escalable, estandarizado, componible y preparado para aplicar la gobernanza característica de SOA.

La parte frontal de la aplicación hará uso de esta infraestructura de servicios modulares y estandarizados para proponer al usuario una gestión del ciclo de vida del servicio transparente a la vez que potente. Asimismo, este diseño hace que se cumpla el objetivo de abrir este inventario de servicios a equipos y aplicaciones externas que podrán ejecutar estos servicios a demanda.

Uno de los objetivos principales del proyecto es el diseño y creación de un inventario de servicios que mejore la eficiencia del equipo de I+D con el que se ha trabajado. La mejora en visibilidad y reutilización de servicios, gobernanza y gestión del ciclo de vida de los microservicios de IA implementados repercutirá positivamente en el trabajo del día a día de los científicos de datos del equipo. Esto también permitirá a este equipo de desarrollo de soluciones IA presentar un inventario de servicios para que sea utilizado por equipos y aplicaciones terceras aportando un mayor valor a la empresa.

Esta infraestructura de comunicación entre servicios que implementa el ESB no añadirá latencias ni tiempos de espera que entorpezcan el desempeño de los servicios gracias a las tecnologías y protocolos que se han utilizado, que se detallarán más adelante.

Se ha estudiado el trabajo de Sujatha Kuppuraju y Aravind Kumar[3] sobre el despliegue de servicios SOA en entornos de producción. En este trabajo se detallan una serie de problemas relacionados con el despliegue de servicios en sistemas SOA, y se analizan y detallan soluciones y aspectos a tener en cuenta. Nuestro proyecto, como se mostrará más adelante, se ha inspirado en diferentes partes de este análisis para conseguir una infraestructura SOA de calidad, como son los pasos para definir servicios o la creación de estándares para los mensajes con el fin de mejorar la trazabilidad del sistema.

3.2. Qué aportamos

Hemos hablado sobre varios tipos de herramientas, plataformas y aplicaciones que se usan comúnmente en la industria comentando sus características principales y la utilidad que tienen.

Podemos decir que nuestro sistema se ubica dentro de las aplicaciones de gestión de recursos software, concretamente dentro de los buses de servicios empresariales, porque proporciona un alto grado de gobernanza sobre los servicios implementados a la vez que gestiona las composiciones de servicios, sus interfaces de comunicación y compone un inventario de servicios.

Nuestro sistema destaca como una opción interesante respecto a las demás herramientas del tipo ESB del mercado por las siguientes propiedades que presenta:

3.2.1. Transparencia

Una de las principales características que destacan de este sistema es la transparencia con la que presenta los datos al usuario. De un vistazo el usuario puede hacerse una idea de la arquitectura y el tamaño del inventario de servicios que constituye la aplicación, a la vez que puede acceder a los flujos de datos que existen entre los diferentes servicios.

Se ha creado una interfaz que representa el estado del sistema SOA de una manera fiel y comprensible para que el usuario entienda cómo gestionar el inventario, alejándose de la complejidad que presentan algunos sistemas de gestión de servicios del mercado.

3.2.2. Escalabilidad

Otro punto fuerte de nuestro sistema es su escalabilidad. Si se llegara a un flujo de tráfico que resultara crítico para el ESB bastaría con levantar un balanceador de carga y una instancia nueva del ESB para duplicar la capacidad de gestionar peticiones.

La concurrencia no sería un problema porque el ESB accede a la base de datos por medio de transacciones y cada cierto tiempo consulta el estado de los objetos del sistema para sincronizar la interfaz.

Del mismo modo sería posible levantar una nueva instancia del host de microservicios para duplicar la potencia de procesamiento tanto para los servicios de inferencia como para los servicios de entrenamiento de modelos. Otro modo de aportar escalabilidad a esta parte del sistema es a través de la configuración de las máquinas Azure donde será desplegado, ya que será posible ajustar la potencia de procesamiento de estas en función de la demanda.

3.2.3. Componibilidad y estandarización

La componibilidad tenía que ser obviamente uno de los puntos fuertes de la herramienta pues ese es su principal objetivo: la composición de microservicios de inteligencia artificial para crear soluciones IA complejas. La mayoría de ESB implementan esta característica, pero no de una manera tan clara e intuitiva.

El hecho de que todas las comunicaciones pasen a través del ESB fuerza a utilizar un estándar de comunicación entre los servicios. Esta estandarización brinda un alto nivel de componibilidad haciendo posible compatibilizar distintos tipos de servicios para crear soluciones complejas.

Aun así, siempre existen casos de microservicios que manejan un tipo de datos completamente diferente al resto, por eso existen servicios auxiliares de formateo o de mapeo, que adaptan y encapsulan estos datos para que sean comprensibles por el sistema y por el resto de los servicios.

El sistema utiliza una estructura de datos de tipo JSON para realizar todas las comunicaciones, que se ejecutan a través de APIs REST a través de HTTP. Más adelante se detalla la razón de usar REST sobre SOAP, así como las reglas de estandarización para el intercambio de mensajes a través del ESB.

La implementación de este sistema SOA, como viene siendo habitual en la industria, obliga a que cada mensaje entre servicios pase a través del ESB, por lo que se maximiza el nivel de estandarización que se puede aplicar a todas las comunicaciones. A esto se le suma la capacidad del ESB de registrar y trazar el origen de cualquier error facilitando los procesos de detección y corrección de errores, además de la posibilidad de aplicar métricas para evaluar cualquier parte del sistema, como veremos posteriormente.

3.2.4. Trazabilidad y robustez

En cuanto a la trazabilidad y robustez del sistema, cualquier error en los servicios y microservicios es capturado y registrado en un registro organizado de errores.

El sistema no experimenta ninguna anomalía ante un fallo en el ESB porque el sistema lo controla mediante excepciones y registros. En cuanto a los microservicios, al estar implementados en una máquina diferente a la del ESB, ante un error en tiempo de ejecución el ESB simplemente registra el punto exacto donde ha ocurrido el error y notifica al usuario. El usuario recibe el resultado de error con diferentes metadatos que le pueden ayudar a resolver el problema.

Como se comenta en la investigación de Mamdouh Ibrahim, Kerrie Holley y Nicolai M. Josuttis [4], hay que tener en cuenta que los procesos de testeo y debugueo se complican a medida que el sistema crece y se dispersa. Estos problemas nacen de la propia idea de computación distribuida. Es necesario diseñar y mantener los sistemas de trazabilidad el ESB ya que esta pieza se convierte en la principal herramienta de debugueo y detección de errores.

3.2.5. Ligereza

Nuestro sistema está implementado principalmente con Node y Express, y Python y Flask, por lo que son sistemas rápidamente configurables y que ocupan un espacio mínimo una vez compilados (o empaquetados). Esto lo hace un sistema flexible y ligero muy cómo de desplegar en cualquier tipo de máquina.

A esto se le suma la intencionalidad de desplegar en forma de contenedores Docker que ha tenido el proyecto desde el inicio, maximizando la modularización de las aplicaciones del sistema y la agilidad de desarrollo y despliegue.

3.2.6. Experiencia de usuario

Finalmente hablaremos de la experiencia de usuario, que no es más que un reflejo de todas las características anteriores. Se trata de un sistema completamente transparente para el usuario, con mucha potencia a la hora de componer servicios por la estandarización y componibilidad que presenta, además de proponer una manera organizada de gestionar el ciclo de vida de estos.

La escalabilidad y robustez del sistema lo hacen altamente atractivo porque trivializa las tareas de configuración y despliegue con características propias de los sistemas pensados para “dockerizar”.

Posteriormente se profundizará en las diferentes características de la interfaz de usuario, correspondientes a la Aplicación Web de Gestión.

3.3. Objetivos de implementación

Como hemos mencionado en capítulos anteriores, el objetivo de este proyecto es el diseño e implementación de un sistema de gestión de servicios y microservicios, y las comunicaciones entre estos. Como característica SOA a destacar, el sistema está altamente orientado a la provisión de servicios que aportan valor al negocio, que en este caso es la implementación de microservicios de inteligencia artificial.

Las partes principales de este sistema son las siguientes:

- Aplicación web de gestión
- ESB
- Host de microservicios

3.3.1. Host de Microservicios

El host de microservicios consiste en una API REST que hospeda y encapsula llamadas a programas de inteligencia artificial: microservicios. Estos programas son en su mayoría soluciones de inferencia programados en Python ligados a una ruta en esta API.

Por otro lado, existen otro tipo de microservicios que son los de entrenamiento. La diferencia con los microservicios anteriores es que estos implementan algoritmos de entrenamiento de modelos, que luego serán utilizados por los algoritmos de inferencia anteriores.

Nuestro objetivo es implementar en forma de microservicios estos algoritmos que se vienen usando habitualmente (y nuevos desarrollos). Esta infraestructura brinda un alto nivel de reusabilidad y componibilidad gracias a la estandarización aplicada en el ciclo de vida del servicio, desde su definición hasta su implementación y mantenimiento.

Adicionalmente, las diferentes interfaces de comunicación no comprometen el rendimiento del sistema pues los protocolos y tecnologías que se utilizan son ligeros y eficientes. Este ha sido desde el principio uno de los objetivos principales del proyecto pues no es admisible que el rendimiento empeore cuando el objetivo de la plataforma es crear una infraestructura de servicios eficiente, reutilizable y mantenible en el tiempo.

3.3.2. Aplicación Web de Gestión

La aplicación web de gestión hace de interfaz de usuario del ESB. Con esta interfaz el usuario puede gestionar el inventario de servicios de una manera clara y transparente, y controlar el ciclo de vida de los servicios y microservicios implementados.

El usuario puede crear composiciones de servicios siempre y cuando los servicios conectados sean compatibles. Para los casos en los que no lo sean existe la opción de crear servicios de mapeo y transformación de mensajes que harán las veces de traductor entre servicios. Estas composiciones de servicios aportarán soluciones complejas que, además, podrán ser compuestas de nuevo en otros servicios compuestos.

Las principales secciones de esta aplicación frontal serán la de gestión de microservicios y gestión de servicios de tarea. En estas dos secciones principales encontraremos la edición, creación y eliminado de servicios, así como la creación de enlaces entre estos, para crear composiciones entre los servicios. Esta aplicación también contará con una página de inicio de sesión.

Esta plataforma web crea un ecosistema en el que se simplifica la gestión del ciclo de vida del servicio y permite aplicar principios SOA que mejoran la gobernanza del inventario de servicios de manera general. Más adelante veremos ejemplos de los diferentes componentes UI y su relación e inspiración en diferentes aspectos del patrón SOA, además de seguir diseños y patrones contrastados para la creación de interfaces de usuario web.

3.3.3. ESB y el inventario de servicios

El ESB podría resumirse en una API REST que controla toda la gestión de servicios (creación, actualización, borrado… de servicios) por un lado, y en el motor que procesa las llamadas a las composiciones de servicios por otro. Cuando se crea una composición de servicios, se registra en el ESB con una ruta pública accesible desde el exterior, de manera que estas composiciones de servicios se convierten en servicios en sí mismos.

Resumidamente, cuando se lanza una llamada a uno de estos servicios compuestos el ESB se encarga de gestionar el proceso enrutando la respuesta de la ejecución de un servicio con la entrada de la llamada al siguiente, de manera síncrona, creando una cadena de ejecución de varios servicios. Esta lógica se explicará en detalle más adelante.

Retomando el trabajo de investigación sobre gobernanza en sistemas SOA [5], Sujatha Kuppuraju y Aravind Kumar destacan una serie de posibles problemas como la falta de un sistema de reutilización, estandarización y definición de políticas, y falta de gestión de la configuración de servicios. Junto con estos aspectos problemáticos se detallan una serie de posibles soluciones, que hemos tenido muy en cuenta a la hora de diseñar nuestro sistema, como pueden ser: diseño del ciclo de vida del servicio, definición de propiedades obligatorias

para la creación de un servicio, creación de taxonomías en el inventario y definición del catálogo de servicios.

Al igual que Sujatha Kuppuraju y Aravind Kumar en su investigación [3], hemos diseñado el ciclo de vida que se aplicará a cada servicio de manera individual para crear un inventario de servicios SOA funcional. De este ciclo de vida cuelga la identificación de las propiedades clave para la definición de cada servicio, al igual que la creación de estándares y reglas para asegurar la visibilidad y gobernanza del catálogo de servicios.

En cuanto a lo que en el campo de la computación distribuida se conoce orquestación de servicios, en nuestro caso no es un tema con tanto peso como en otros sistemas ya que no estamos trabajando con ejecución de procesos paralela ni necesitamos protocolos complejos que aporten integridad, orden o coherencia a los datos o mensajes. No modelamos la orquestación de servicios como tal, sino que la simplificamos e implementamos a través del motor de enrutado del ESB, que se encarga de procesar las llamadas a servicios de tarea, enlazando un servicio con el siguiente como un proceso secuencial compuesto.

Cuando se requiere algún tipo de almacenado de datos por parte del ESB, como por ejemplo a la hora de registrar servicios, se hace uso de del sistema gestor de bases de datos MongoDB, del cual se hablará más adelante.

Entre el API REST y MongoDB, se conforma el inventario de servicios, que es el pilar principal de una infraestructura SOA. El inventario de servicios debe constituirse siguiendo unas pautas de estandarización, visibilidad y componibilidad que se aplican a cada servicio individual, y al inventario de servicios de manera transversal. Estas pautas y reglas de definición e implementación de servicios, que se detallarán en secciones posteriores, constituyen las bases que dotarán al inventario de servicios de la gobernabilidad y gestión del ciclo de vida del servicio que requiere un sistema SOA.

En cuando a la visibilidad y descubrimiento de servicios, cabe destacar que en nuestro trabajo se ha buscado crear un método simple y robusto para aplicar visibilidad a los servicios del inventario. Como se detallará más adelante, se ha hecho uso de la herramienta Swagger para documentar y exportar la información de la API del host de microservicios.

3.3.4. Docker

Por otro lado, una vez diseñado y desarrollado el sistema, el último paso será preparar toda la infraestructura necesaria para comenzar un proceso migración a Docker. Esta decisión viene respaldada por la naturaleza que queremos darle al sistema, pues una aplicación “dockerizada” tiene ventajas como la agilidad que aporta a los despliegues por no requerir configuración a la hora de realizarlos. Se trivializa la configuración relacionada con la plataforma que hospeda el sistema ya que el único requerimiento es que sea capaz de ejecutar Docker. Esto también mejora sustancialmente el proceso de desarrollo porque una vez desarrollado el sistema o implementada alguna actualización, si el sistema funciona en el Docker de desarrollo es seguro que funcionará en diferentes entornos.

Otra característica interesante de Docker es la posibilidad de duplicar el contenedor si fuera necesario. Un ejemplo donde Docker sería útil sería en el caso en el que se experimentara un alto tráfico de peticiones de ejecución de servicios. En este caso se podrían levantar varias instancias del sistema fácilmente porque estaría completamente autocontenido en un contenedor replicable.

El sistema “dockerizado” sería desplegado en la nube para proporcionar acceso tanto a desarrolladores como a consumidores de los servicios implementados. Aquí realizaríamos un estudio sobre la viabilidad de un sistema de este tipo en cuanto a rendimiento y eficiencia con diferentes tipos de pruebas de carga y estrés.

Este estudio es importante porque los microservicios del tipo de entrenamiento de modelos consumen una gran cantidad de recursos y es necesario conocer las necesidades del sistema en cuanto a hardware en función del tráfico.

Además, sería conveniente conocer la capacidad de las diferentes APIs de gestionar peticiones concurrentes pues en un futuro una posibilidad es abrir el sistema a un público fuera de la empresa y comercializar la plataforma como SaaS.

Por otro lado, se realizará un estudio sobre la mejora de rendimiento del equipo de I+D con el que se ha trabajado una vez se integre el sistema con su flujo de trabajo habitual a la vez que se encuestará a los equipos terceros que lo utilicen para estudiar el impacto de la nueva herramienta.

Docker y Azure

El diseño de este proyecto se ha orientado, como hemos comentado, al despliegue final utilizando Docker, permitiendo crear aplicaciones escalables y componibles de una manera ágil y escalable.

Junto con Docker, las pruebas de despliegue se han realizado en servidores en la nube de Microsoft Azure. Como se ha comentado en apartados anteriores, esta nube aporta seguridad y estabilidad a nuestro sistema, además de permitir escalar las aplicaciones por la flexibilidad que proporciona para encender máquinas al momento. Azure ha demostrado ser una plataforma IaaS segura, en la que confían miles de instituciones, empresas y particulares.

En nuestro caso particular, Docker y Azure están perfectamente sincronizados, principalmente por la aceptación por parte de la industria de esta plataforma IaaS junto con los contenedores Docker. Azure tiene una gran capacidad para gestionar aplicaciones desplegadas en contenedores Docker muy potente, permitiendo realizar multitud de operaciones controlando el acceso por roles.

Se han estudiado diferentes alternativas como AWS o Cubernetes, pero finalmente se ha decidido continuar con Docker y Azure porque en la empresa en la que se ha desarrollado el proyecto existen equipos con experiencia en estas tecnologías.

El equipo de I+D con el que se ha trabajado a lo largo de este proyecto tiene experiencia desplegando diferentes tipos de aplicaciones dockerizadas en servidores de Azure, por lo que esta integración se hará de manera ágil. Adicionalmente, el mantenimiento y resolución de errores relacionadas con este tipo de tareas DevOps se llevará a cabo en conjunto por los equipos de I+D y el desarrollador de la plataforma SOA de microservicios.

3.4. Flujo de trabajo

Como ya hemos explicado, nuestra herramienta está compuesta por tres pilares fundamentales: la aplicación de gestión, el ESB y el host de microservicios. Sin embargo, los posibles usuarios de este sistema no tienen por qué conocer o utilizar todas sus partes.

Diferenciaremos 3 tipos de usuarios del sistema, desarrollador, consumidor y soporte, siendo este último el perfil que dará soporte a la aplicación de manera general y podría no considerarse un cliente de la aplicación como tal.

Antes de comenzar con los tipos de usuarios del sistema, detallaremos el proceso de definición de servicios.

3.4.1. Proceso de definición de servicios

En esta sección detallaremos el proceso de definición de servicios que se ha diseñado para asegurar el cumplimiento de estándares y pautas SOA tanto para servicios de tipo tarea como para microservicios.

Comentaremos también que estas pautas están fundamentadas en diferentes artículos de investigación y otras fuentes de información, entre las cuales destaca la investigación Sujatha Kuppuraju y Aravind Kumar [3].

Validación y publicación de servicios

El proceso comienza con la definición del proceso de **validación y publicación** de servicios. Estas pautas se aplican por igual tanto a servicios de tarea como a microservicios:

- Un servicio debe ser **identificable de manera única**: cada servicio de tarea y microservicio creado y almacenado en el sistema gestor de base de datos tiene una propiedad que lo identifica de manera única, el nombre.

- Los servicios deben presentar una **interfaz estandarizada de entrada y salida de datos**. Se ha definido una estructura para los mensajes de entrada y salida de todos los servicios de manera que cada comunicación sigue un patrón estandarizado y comprensible por el sistema.

- Todos los servicios deben ser **publicados** en sus respectivas APIs: microservicios en la API del host de microservicios y servicios de tarea en la API del ESB. Estas interfaces de comunicación son APIs REST que publican servicios listos para ser utilizados por aplicaciones terceras.

Definición del servicio

La definición del servicio varía en función de si se trata de un servicio de tarea (composición de servicios) o de un microservicio de inteligencia artificial.

Ambos tipos de servicios exigen definir los siguientes campos:

- **Nombre**: identificación única del servicio.
- **Tipo de servicio**: tarea o microservicio.
- **URI de punto final**: ubicación del servicio. En este caso al tratarse de un servicio de tarea la URI de punto final deberá estar compuesta por la dirección del ESB mas el directorio con el mismo nombre del servicio. Es decir: *host/nombreServicio.* Esto es así porque el ESB gestiona y publica las rutas de los servicios de tarea de manera dinámica conforme se registran estos servicios en la base de datos. En el caso de un microservicio su URI será la correspondiente en el host de microservicios.
- **Tipo entrada**: tipo de datos que recibirá el servicio como entrada.
- **Tipo salida**: tipo de datos que devolverá el servicio como respuesta.
- **Body**: lista de servicios que componen el servicio de tarea. Estos servicios pueden ser tanto del tipo tarea como del tipo microservicio, y serán invocados de manera secuencial enrutando la respuesta de un servicio con la entrada del siguiente. Es importante remarcar que el campo “body” no existe en la creación de microservicios ya que es un dato exclusivo de los servicios de tipo tarea.

Catálogo de servicios y visibilidad

El inventario de servicios está constituido por varias aplicaciones que en conjunto crean la gestión y gobernabilidad propias de SOA. Estas aplicaciones, como veremos más adelante, son el ESB, la aplicación web de gestión y la base de datos MongoDB. El host de microservicios también constituye en parte el inventario de servicios, pero lo dejaremos a parte porque su única tarea es la de implementar la lógica de los microservicios y publicar una URI de punto final para acceder a ellos.

El ESB es la aplicación que comunica la base de datos, donde se guarda la definición de los servicios tanto de tarea como microservicios, con la aplicación web de gestión que utiliza el usuario. Es esta relación la que construye la experiencia de gestión del ciclo de vida de servicios SOA.

La visibilidad está asegurada por dos partes. La primera porque la aplicación web de gestión muestra al usuario la información almacenada en la base de datos, que debe ser mantenida por los perfiles que implementan y definen los servicios. Y por otro lado porque con la herramienta Swagger se creará documentación dinámica de las diferentes interfaces web REST (ESB y host de microservicios) para que usuarios externos conozcan estas interfaces.

Usuarios, roles y acceso al inventario de servicios

Finalmente hablaremos de las diferentes formas de acceder a la plataforma y el control de acceso que se llevará a cabo.

La base de datos recoge un registro de usuarios pertenecientes a diferentes roles. Estos roles afectan a la actividad de los usuarios en la aplicación web de gestión:

El rol **administrador** tiene acceso a toda la funcionalidad de la aplicación web de gestión, desde creación y edición de servicios, hasta administración de accesos y usuarios.

El rol **desarrollador** puede crear, editar, eliminar y ejecutar servicios del sistema.

El rol **externo** tan solo puede ejecutar servicios en la plataforma utilizando la interfaz creada explícitamente para ello.

Por otro lado, el inventario de servicios está pensado para ser utilizado por aplicaciones terceras, sin necesidad de pasar por la aplicación web de gestión, por lo que tanto el ESB como el host de microservicios presentan interfaces de comunicación REST públicas.

En este momento no se está protegiendo este acceso a la ejecución de servicios ya que el sistema será desplegado en un entorno altamente controlado y sin acceso al exterior de la empresa. Sin embargo, próximamente se creará un sistema de permisos y roles para controlar el uso de recursos por parte de los usuarios externos y aplicaciones terceras para tener un mayor control sobre el uso del sistema.

3.4.2. Desarrollador científico de datos

Este perfil es el usuario que más intensivamente utiliza el sistema ya que el diseño de la herramienta se ha orientado a mejorar su trabajo.

Este perfil es el encargado de crear soluciones IA mediante la implementación de servicios de inferencia y entrenamiento de modelos en el host de microservicios, su registro en el inventario de servicios, y la composición de los anteriores como servicios de tarea.

Caso de uso

Un caso de uso real sería el siguiente, en el caso de implementar un servicio del tipo microservicio IA:

- Diseño a alto nivel del servicio que se quiere implementar. Definir:
  - Tipo de microservicio: inferencia/entrenamiento
  - Tipo de datos de entrada y salida del servicio
  - Creación de los servicios de mapeo y transformación si fueran necesarios
- Implementación del algoritmo de inferencia o entrenamiento en Python en el host de microservicios
- Integración de la implementación anterior con el API REST Flask y publicación de la ruta para ejecutar el servicio
- Registro del microservicio en el ESB a través de la aplicación web de gestión
- El servicio está listo para usar a través de HTTP

Por otro lado, un caso de uso en el que se implementara un servicio de tarea compuesto por varios microservicios seguiría el siguiente flujo:

- Diseño a alto nivel del servicio. Definir:
  - Tipo de datos de entrada y salida del servicio
  - Creación de los servicios de mapeo y transformación si fueran necesarios
- Registro del microservicio en el ESB a través de la aplicación web de gestión enlazando los microservicios, servicios de transformación y servicios de tarea requeridos.

  **Todos ellos deben ser compatibles con la salida del servicio anterior y con la entrada del siguiente.*

- El servicio está listo para usar a través de HTTP. El ESB autopublica las rutas a los servicios de tarea que se registran.

Con estos dos flujos de trabajo un desarrollador de soluciones IA es capaz de crear composiciones de servicios complejas de una manera modularizada y componible. Los servicios del tipo microservicio creados quedan encapsulados y registrados en el inventario de servicios para su posterior uso. Estos microservicios creados están preparados para ejecutarse tanto como composición junto con otros servicios, como de manera individual.

Más adelante se mostrará con ejemplos el flujo de creación de ambos tipos de servicios, así como las diferentes partes de la interfaz.

3.4.3. Consumidor de servicios

El perfil de consumidor de servicios es más limitado que el anterior, ya que su actividad se reduce al consumo de servicios del inventario de servicios de la organización, que ha sido previamente creados y modelado por el equipo de desarrollo de ciencia de datos. El propio equipo que implementa los servicios y microservicios también puede hacer de consumidor de servicios para crear sus propias soluciones.

Como ya hemos comentado, tanto los servicios del tipo microservicio como las composiciones de servicios (tarea) son candidatos para consumirse de manera individual.

Un ejemplo de caso de uso podría ser a la hora de crear una interfaz para un usuario no técnico con la que subir una imagen para detectar qué objeto se muestra en ella. El equipo tercero que cree esta aplicación utilizaría el microservicio o servicio de tarea correspondiente, que recibe como entrada una imagen y devuelve una respuesta con los resultados de la inferencia, que serán interpretados por esta aplicación tercera. Más adelante se mostrarán ejemplos más detallados de casos de uso.

Dentro del perfil de consumidor de servicios mencionaremos que uno de los posibles proyectos futuros es diseñar e implementar la infraestructura necesaria para abrir el acceso a estos servicios compuestos a un público fuera de la empresa, ya sea a otras empresas o al público general. Esto vendría acompañado de un método de monetización como pago por subscripción o por consumo de recursos.

3.4.4. Soporte

Finalmente hablaremos del equipo de soporte, que, aunque no es un cliente real de la aplicación, es uno de los perfiles que más contacto va a tener con ella.

Este equipo ha sido el encargado de diseñar y desarrollar el sistema por completo, por lo que es el más indicado para dar soporte al mismo. Este equipo ha realizado el proceso de desarrollo en conjunto con las indicaciones y requerimientos del equipo de ciencia de datos objetivo, siguiendo una metodología de entregas y demostraciones periódicas, por lo que las actualizaciones y tareas de soporte se verán agilizadas si se continúa trabajando de la misma manera.

Concretamente, este perfil se ocupa de las actualizaciones y correcciones de la aplicación web de gestión y del ESB por completo. Del host de microservicios se encarga de todo lo relacionado con la infraestructura de la API y el enlazado de peticiones con los microservicios, ya que del propio desarrollo de la lógica de los microservicios se encarga el equipo de desarrollo de I+D.

3.5. Tecnologías y herramientas utilizadas

A lo largo del desarrollo de este proyecto se han utilizado una serie de herramientas y tecnologías para implementar las diferentes partes del ESB, el host de microservicios y la aplicación web de gestión de servicios.

Podemos dividir el tipo de herramientas en 2 categorías: de desarrollo y de implementación. Las de desarrollo consisten en herramientas que solamente se han utilizado para facilitar el diseño y desarrollo del sistema, y que no están presentes de ninguna manera en la aplicación final. Por otro lado, las herramientas de implementación son aquellas que forman parte del código del sistema como podrían ser librerías o frameworks.

3.5.1. Herramientas de desarrollo

Dentro del grupo de herramientas de desarrollo, la más remarcable es la herramienta de gestión de la configuración **Git**, que se ha utilizado de manera continua para implementar los diferentes componentes del sistema y trabajar de manera conjunta y coordinada.

Se ha utilizado Postman para probar las diferentes interfaces de comunicación o APIs presentes en las aplicaciones del sistema. Esta herramienta sirve para configurar y ejecutar diferentes tipos de peticiones web. Además, permite crear un listado de llamadas a servicios web organizado y exportable, que se puede configurar mediante variables globales para adaptar las llamadas a diferentes entornos y utilizar diferentes formas de autenticación de manera cómoda.

Se ha utilizado el editor de código VisualStudio Code por ser un editor ágil a la vez que potente, que permite instalar una gran variedad de plugins para agilizar el desarrollo de código, como formateadores o generadores de código automático, o hasta soporte para otras herramientas como Python y Jupyter Notebooks. Sobre esta última herramienta, se ha utilizado Jupyter Notebooks durante el desarrollo porque es la herramienta con la que trabaja el equipo de I+D normalmente, para entender su flujo de trabajo y crear una experiencia de usuario similar a la hora de implementar la lógica de los microservicios en el host de microservicios.

Finalmente comentaremos el uso de las herramientas Npm y Pip, los gestores de paquetes que hemos utilizado para incorporar las diferentes librerías a los proyectos tanto del host de microservicios (Pip), el ESB (Npm) y la aplicación web de gestión (Npm).

3.5.2. Herramientas de implementación

Por otro lado, se han utilizado herramientas que hemos denominado herramientas de implementación, que han apoyado el diseño y desarrollo del sistema y que son parte de su estructura e implementación.

El primer ejemplo que comentaremos es Flask. El host de microservicios hace uso de esta librería para crear APIs REST en Python. Con esta librería se ha creado una API para acceder a las funciones de entrenamiento e inferencia implementadas en Python por el equipo de I+D. La principal razón para utilizar Flask es por lo sencillo que resulta desarrollar software de este tipo para perfiles que no son expertos en el desarrollo y consumo de servicios web.

En el ESB, la principal tecnología que se ha utilizado es Node y Express para crear el servidor web que contiene el WebAPI que soporta la gestión del inventario de servicios. De la misma manera que Flask, con Express se agiliza enormemente el diseño y desarrollo de interfaces de comunicación web, por la cantidad de librerías y módulos presentes para gestionar comunicaciones web en Npm, que ofrecen funcionalidad como transformación, formateo y logeo de peticiones. Algunos ejemplos de estas librerías son body-parser, cors o morgan.

Continuando con el backend del ESB, el sistema gestor de base de datos utilizado ha sido MongoDB, una base de datos basada en documentos, a través del paquete de Npm Mongoose. Esta base de datos no relacional permite gestionar de colecciones de objetos de manera sencilla, rápida y segura.

Como se detallará más adelante, se ha usado JSON Web Token como tecnología para implementar la autenticación y autorización de las interfaces de comunicación del sistema.

Para la parte frontal de la aplicación web de gestión se ha utilizado el framework de desarrollo web frontal creado por Google, Angular. Este framework permite desarrollar aplicaciones web y móviles frontales de manera ágil y estructurada. Esto se consigue con la arquitectura orientada a componentes y el uso de typescript como lenguaje de programación. Una vez terminado el desarrollo, la aplicación Angular se compila y minifica para crear un paquete JavaScript, HTML y CSS multiplataforma para servirse como un árbol de ficheros estáticos en una ruta de la aplicación Node Express, junto con el API REST de gestión de servicios.

La principal ventaja de este tipo de frameworks es que con un solo desarrollo se puede crear una aplicación completa y funcional para entornos web, al mismo tiempo que para los sistemas operativos iOS y Android (con la ayuda de herramientas externas), por lo que en un futuro la aplicación web de gestión podría ser adaptada para gestionar el inventario de servicios a través de dispositivos móviles.

Hablando de la parte estética de la aplicación Angular, se ha utilizado el conjunto de librerías Angular Material para mejorar la experiencia de usuario, y para agilizar el desarrollo evitando desarrollar componentes y estilos propios.

3.5.3. SOAP vs REST sobre HTTP

Uno de los puntos clave de una arquitectura orientada a servicios es la elección de las herramientas, protocolos y patrones sobre los que funcionará toda la mensajería. Hemos estudiado cómo se implementa la mensajería en diferentes sistemas, de diferentes tipos, y hemos elegido una composición de protocolos y herramientas que se adapta a nuestro sistema, considerando como candidatos SOAP y REST sobre HTTP.

Hemos tenido en cuenta principalmente los siguientes aspectos a la hora de realizar el estudio y la comparativa:

Estandarización de mensajes: como se ha comentado en varias secciones, la estandarización de mensajes es un aspecto fundamental en un sistema SOA, para proporcionar al sistema componibilidad, reusabilidad e interoperabilidad. La estandarización es una característica sobre la que se construye toda la infraestructura SOA, por lo que es importante dedicar tiempo a la elección y diseño del estándar.

Trazabilidad y robustez: de nuevo, es necesario contar con un sistema de comunicación que favorezca la trazabilidad de las comunicaciones del sistema, tanto para la aplicación y estudio de métricas de rendimiento, como para la resolución de errores y mantenibilidad del sistema. La estandarización comentada anteriormente mejora además las tareas de registro de errores y monitorización de sistema.

Componibilidad y reutilización: el protocolo utilizado repercute directamente en la capacidad de componibilidad e integración de las aplicaciones del sistema SOA. Es deseable que el protocolo utilizado sea lo más adaptable, configurable y estandarizado posible. Esto potencia mejora además el mantenimiento del sistema a lo largo del tiempo ya que los futuros cambios serán fácilmente integrables.

Además, mencionaremos el trabajo de Alain Hsiung, Giovanni Rivelli y Georg Hüttenegger en el que hablan del diseño de una infraestructura SOA global [6], que, aunque en primera instancia parezca que no encaja con nuestro proyecto por estar dirigido a una pequeña empresa, aporta un punto de vista interesante y varios aspectos que hemos tenido en cuenta tanto en el diseño como en la implementación.

Uno de estos aspectos es la parte donde hablan de la elección del protocolo en relación con la seguridad y diseño de los servicios. En particular, remarcan la necesidad de utilizar un estándar de comunicación que fomente la trazabilidad, y la creación de identificadores únicos para los mensajes y las aplicaciones. Para facilitar la monitorización de la actividad del ESB, que en ocasiones se comporta como una caja negra en la que es difícil detectar errores.

A pesar de que en el artículo de investigación mencionado se defiende el uso de SOAP, nosotros hemos optado por REST sobre HTTP para soportar la comunicación entre servicios y a continuación detallamos las razones.

SOAP

El protocolo de comunicación predilecto en este tipo de sistemas siempre ha sido SOAP. SOAP implementa un protocolo a través del cual los procesos intercambian información utilizando objetos en formato XML. No tiene estado por lo que todos los mensajes intercambiados siempre son comprensibles por todas las partes, y permite construir protocolos más complejos escalando el objeto XML porque su estructura de "Envelope", "Header", "Body" y "Fault" es muy potente. Un ejemplo de esto es la posibilidad de introducir encabezados con autenticación en el "Header".

Una de las características más interesantes de SOAP es que puede ser utilizado sobre HTTP, SMTP y JMS, haciéndolo una opción realmente interesante para sistemas distribuidos que presentan mucha heterogeneidad entre sus aplicaciones.

Por su versatilidad, SOAP se ha venido utilizando para implementar sistemas de comunicación de todo tipo, entre los que podemos destacar la invocación de procedimientos remotos o el envío de correos electrónicos.

Sin embargo, SOAP no solamente tiene ventajas. Por la naturaleza del formato de los mensajes XML, SOAP es un protocolo más lento que otros como CORBA. Otro punto negativo es la dependencia que tiene del WSDL, que, si bien es cierto que el WSDL puede resultar útil para mejorar la visibilidad de los servicios web SOAP, es una parte más del servicio a mantener. Finalmente remarcaremos que los mensajes XML de SOAP son difícilmente comprensibles por humanos, por lo que en ocasiones complica las tareas de detección y resolución de errores.

REST

REST realmente es considerado una arquitectura más que un protocolo de comunicación. Cuando hablamos de utilizar REST sobre HTTP nos referimos al diseño de una arquitectura REST utilizando el protocolo HTTP y sus operaciones como soporte.

Como comentaremos en secciones posteriores, las operaciones de HTTP encajan con una arquitectura REST pues las operaciones CRUD se equiparan con CREATE, GET, PUT y POST de HTTP. REST define recursos, que son accedidos mediante un identificador (URI), por lo que el descubrimiento de servicios en interfaces REST se convierte en una tarea trivial, sin necesidad de WSDLs como con SOAP. Dentro de cada petición HTTP se puede utilizar cualquier formato (JSON, XML, HTML …), por lo que para que dos componentes se comuniquen solo necesitan tener acceso a la red para usar HTTP.

La principal característica de REST, al igual que SOAP, es que se utiliza como un protocolo sin estado pues todos los mensajes son autocontenidos y comprensibles por ambas partes. Este tipo de arquitecturas se utiliza en sistemas del tipo cliente-servidor, por lo que resulta idónea para nuestro ESB pues hace de servidor para el sistema SOA. El host de microservicios también hace de servidor para la ejecución de microservicios.

Otra característica que hace que muchos desarrolladores terminen eligiendo REST sobre HTTP es la utilización de JSON para el formato de los mensajes, pues es un formato ligero y comprensible fácilmente por las personas. De esta manera se facilitan las tareas de monitorización y detección de errores.

REST es comúnmente utilizado en la construcción de servicios web ligeros, ya que, al no ser un protocolo como tal, gran parte del diseño e implementación es decisión del equipo desarrollador, permitiendo implementar estándares o protocolos propios. Por esto, esta arquitectura es comúnmente utilizada por equipos que siguen metodologías de desarrollo ágiles como SCRUM o Kanban, porque habiendo definido unas pautas para la creación de las interfaces de comunicación, el resto de la implementación es libre para cada equipo.

Remarcaremos que esta libertad que brinda REST sobre HTTP es un arma de doble filo. A diferencia de SOAP, el equipo que implementa una interfaz REST debe encargarse por completo de la infraestructura relacionada con la seguridad y la estandarización de los mensajes. SOAP es un protocolo que no tiene esta libertad por lo que el desarrollador no tiene que preocuparse por estos aspectos ya que SOAP los trae implementados en su protocolo.

REST para nuestro sistema

Como bien afirman Geetha Presena, Balaji Kandan y Asish Kumar en su investigación [7], la adopción SOA en un entorno empresarial significa la creación de una infraestructura IT que potencie la reutilización de servicios en un entorno heterogéneo. Para nuestro sistema esto significa que tanto el ESB, como la aplicación web de gestión, como todos y cada uno de los servicios implementados en el host de microservicios tienen que ser capaces de trabajar conjuntamente, y esto ocurre gracias a la utilización de estándares y protocolos de los que llevamos hablando en este capítulo.

Respecto a lo que se comenta en esta investigación sobre la escalabilidad y flexibilidad que deben tener este tipo de implementaciones SOA, podemos decir que nuestro sistema es altamente escalable. Desde la flexibilidad que aporta la “dockerización” en cuanto a potencia de procesamiento para el Host de microservicios hasta la posibilidad de levantar balanceadores de carga para soportar altas cargas de peticiones. Con la “dockerización” conseguimos un nivel realmente alto de desacoplamiento entre implementación y hardware por lo que el tipo de maquina sobre el que corre cada sistema se vuelve irrelevante siempre y cuando pueda ejecutar un contenedor.

Tras el estudio realizado acerca de REST y SOAP de las ventajas y desventajas que presentan relacionadas con nuestro sistema, se ha concluido que en nuestro caso resulta más ventajoso utilizar HTTP REST principalmente por la simplicidad de las estructuras de los mensajes (estructuras JSON que creamos nosotros), y su consecuente velocidad de envío y decodificación.

Además, los tiempos de desarrollo se ven aumentados gracias a la libertad de implementación de cada servicio, pues el único requisito es seguir el estándar de mensajería que se ha definido.

Es cierto que SOAP fuerza a la creación de sistemas de comunicación seguros y estandarizados, pero la sobrecarga de procesamiento que requieren los mensajes SOAP y la agilidad que se pierde a la hora de desarrollar servicios que cumplan con el protocolo, no merece la pena.

Para mitigar los riesgos de la libertad de implementación que trae REST, se ha hecho especial énfasis en la creación de estándares para los objetos que transportan las peticiones HTTP y se implementará middleware específico en las diferentes APIs (ESB y host de microservicios) para asegurar su cumplimiento.

3.6. Arquitectura e implementación

En esta sección explicaremos en detalle los componentes que forman las tres aplicaciones de este sistema gestor de servicios de inteligencia artificial. Hablaremos sobre las tecnologías que implementan y sus funciones, así como sobre su arquitectura y relación con los demás componentes.

3.6.1. ESB

El eje sobre el que gira una infraestructura SOA es, como se ha comentado en varias ocasiones, el Bus de Servicios Empresariales. Es por esto por lo que en nuestra investigación se ha puesto especial énfasis en el estudio de diferentes trabajos de investigación acerca del diseño e implementación de esta compleja pieza de la arquitectura SOA.

A continuación, retomamos el trabajo de investigación de Alain Hsiung, Giovanni Rivelli y Georg Hüttenegger [6]. Hablan de la empírica necesidad de dedicar tiempo y esfuerzo en el diseño del ESB ya que es la clave para la interoperabilidad del sistema y los servicios. El bus permite la comunicación entre todos los servicios de una manera estandarizada y controlada.

En nuestro caso, el ESB es una aplicación Node. Esta aplicación implementa una API web Express con la que se gestionan todas las operaciones relacionadas con los servicios como la creación, eliminación, actualización, obtención y ejecución de estos. Esta última operación, ejecución, implementa el motor que hace que el ESB conecte servicios entre sí. Más adelante detallaremos y mostraremos diferentes esquemas del funcionamiento de esta funcionalidad de ejecución de servicios.

Todos los servicios se publican mediante interfaces web API, por lo que no es necesario gestionar ningún tipo de documento como haría falta con SOAP y sus WSDL. En cambio, utilizaremos Swagger para dotar de visibilidad a nuestro inventario de servicios, y documentar y exportar la estructura de las interfaces. De esta manera, la visibilidad del inventario de servicios se mantiene actualizado con los diferentes cambios que pueda sufrir la estructura de servicios.

Por otro lado, como en toda infraestructura SOA con ESB, al obligar a que todas las comunicaciones pasen a través del ESB se habilitan procesos de monitorización, estandarización y trazabilidad al sistema de manera general.

El ESB, como ya hemos mencionado, es una aplicación Node Express, que, junto con una implementación de base de datos MongoDB, hace las funciones de inventario de servicios y gestor de operaciones sobre los servicios. Su principal función es la de presentar una API con la que gestionar el inventario de servicios siguiendo el patrón de diseño REST, definiendo el recurso Servicio. Como veremos, esa API también implementa llamadas relacionadas con autenticación y autorización, y con la ejecución de servicios.

REST, HTTP y Mongoose

Como hemos explicado anteriormente, una arquitectura REST aplicada a una interfaz consiste en un protocolo cliente-servidor sin estado, de manera que todas las peticiones son autocontenidas, completas y comprensibles por ambas partes. Normalmente se definen unas operaciones básicas por defecto, lo que encaja perfectamente con el protocolo HTTP, cuyos métodos más importantes son POST, GET, PUT y DELETE.

Finalmente es importante la localización de los recursos con los que se trabaja, que en REST sobre HTTP suelen definirse mediante la propia URI. Se ha definido también una operación para gestionar las llamadas a la ejecución de los servicios que se comentará más adelante.

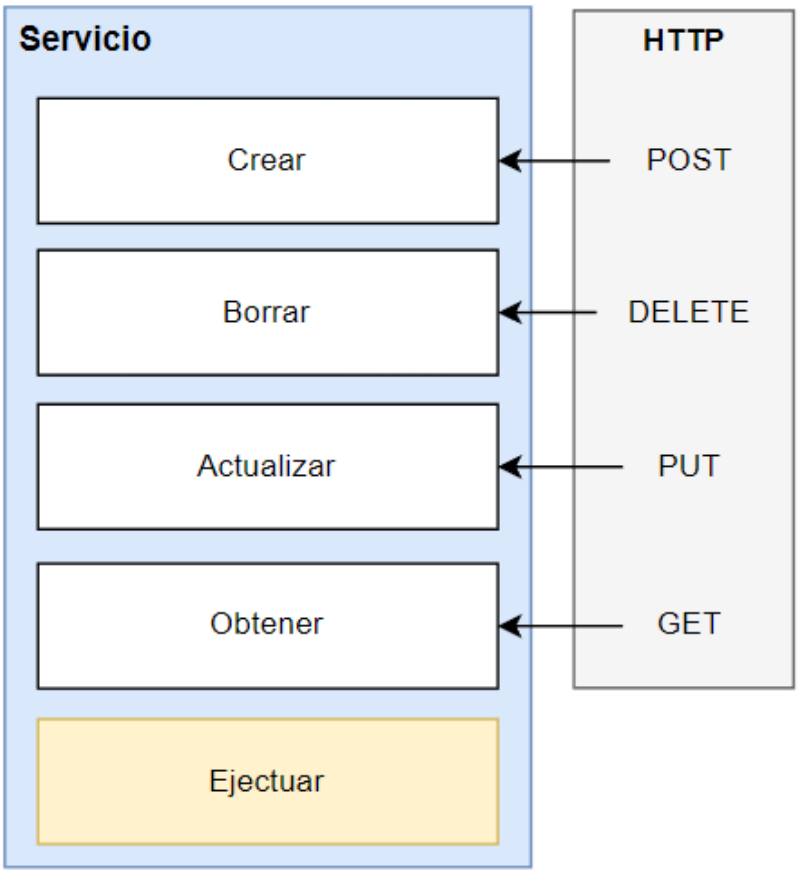

Figura 1. Operaciones REST sobre el recurso servicio (fuente propia).

La librería con la que se conforma el recurso servicio es Mongoose porque el sistema gestor de bases de datos MongoDB trabaja con este tipo de estructuras. Se ha elegido una base de datos no relacional porque la configuración se simplifica con respecto a una base de datos relacional tradicional.

Las bases de datos no relacionales se basan en la creación de tipos de colecciones, lo que encaja perfectamente con los tipos de datos con los que trabaja nuestro sistema: Servicio (y microservicio) y usuarios.

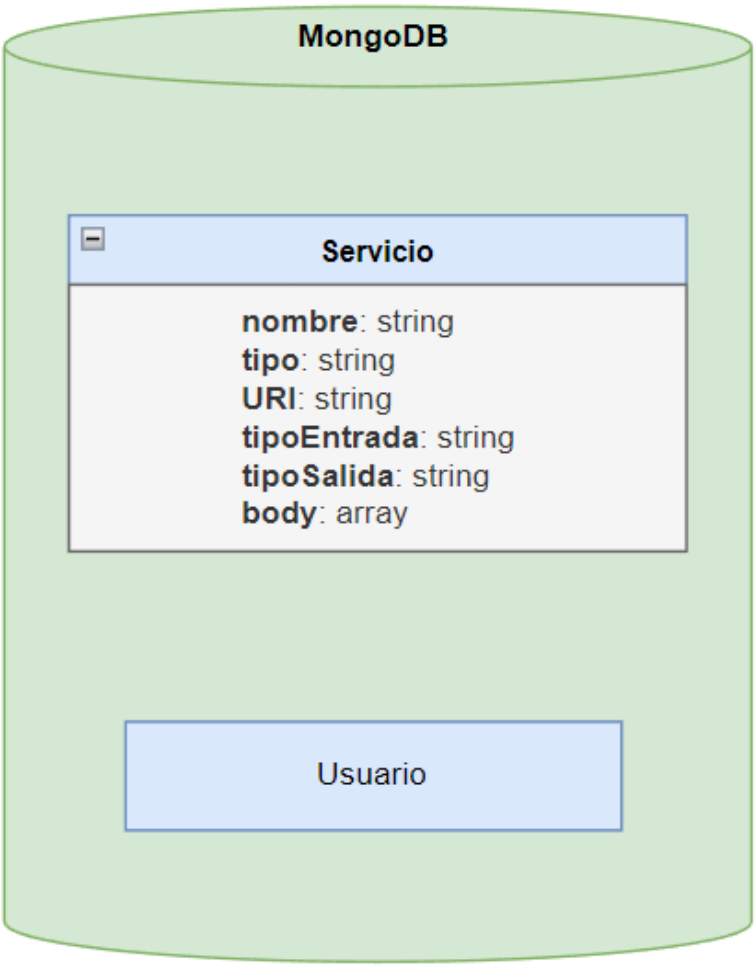

Figura 2. Estructura de las colecciones Servicio y Usuario (fuente propia).

JSON Web Token

A continuación, comentaremos la funcionalidad de autenticación y autorización que presenta el ESB. Se utiliza autenticación del tipo JWT o JSON Web Token. Este estándar se basa en el formato JSON para crear estructuras de datos que encapsulan información sobre la identificación del usuario, el tiempo de expiración de la identificación o el proveedor de esta identificación entre otros datos. Básicamente cuando el usuario introduce sus credenciales y el servidor comprueba su validez, se genera esta estructura que es almacenada por el cliente. A partir de aquí el cliente deberá incluir el JWT en cada petición al servidor.

Las principales ventajas de JSON Web Token es que es un mecanismo de autenticación sin estado y evita la creación de sesiones en el lado del servidor. El token es una estructura autocontenida que sirve para validar cada petición y controlar las sesiones desde el lado del cliente, por lo que el servidor sólo se encarga de generar el token y validarlo en cada petición.

A continuación, en la figura 3, se muestra el ESB y su relación con la aplicación web de gestión y la base de datos MongoDB. Aunque parezca que la aplicación Angular de gestión de servicios es una entidad externa al ESB, realmente está empaquetada y es servida de manera estática desde una ruta del ESB. Esta arquitectura se comentará en detalle más adelante.

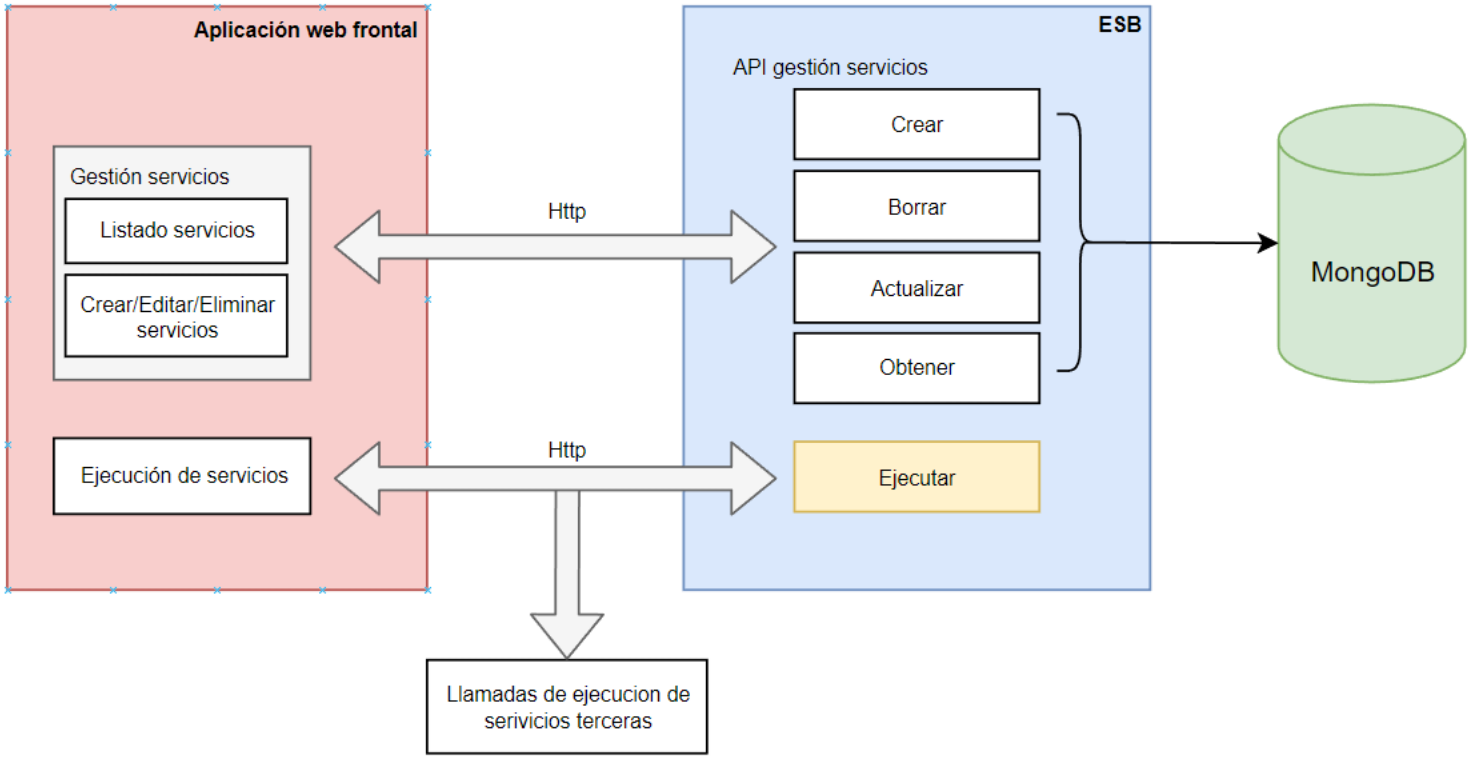

Figura 3. Esquema de las aplicaciones ESB, aplicación web de gestión y base de datos MongoDB (fuente propia).

Ejecución de servicios

Como ya hemos comentado, esta API presenta una llamada para ejecutar servicios de tarea, que se encarga de recibir los parámetros de entrada de la llamada y redirigir las llamadas y resultados a los servicios que componen el servicio de tarea en cuestión. Esta parte del ESB es el engranaje principal que soporta el procesamiento de las llamadas a los servicios de tarea. La llamada del tipo ejecución de servicio de tarea es genérica para todos los servicios, ya que el nombre del servicio a ejecutar viene como parte de la URL. El ESB recoge el nombre del servicio de tarea y lo recupera de la base de datos para obtener el objeto servicio en cuestión.

Tanto los microservicios como los servicios de tarea utilizan el mismo tipo de datos. Con este objeto servicio, se realiza una serie de comprobaciones, como el tipo de datos de entrada, de salida, el tipo de servicio etc. La figura 4 muestra la estructura de la clase Servicio, que anteriormente hemos visto que se almacenará en la base de datos MongoDB.

Figura 4. Estructura de la clase Servicio.

En este punto el ESB accede a la propiedad “body” del servicio, que contiene la lista de servicios que componen este servicio de tarea. Estos servicios internos se recuperan de la base de datos para obtener su dirección, tipo de datos de entrada, de salida… y se comienza a realizar llamadas a estos servicios en orden, de manera síncrona, utilizando primero la entrada proporcionada en la llamada a este servicio de tarea principal, y las salidas sucesivas de los servicios que se van llamando. El primer servicio que compone este servicio de tarea recibe la entrada del mismo servicio de tarea, y cuando se llama y recibe una respuesta, esta respuesta se utiliza para llamar al siguiente servicio en la cadena.

En las siguientes figuras se muestran ejemplos de objetos servicio y microservicio. Vemos las propiedades que definen estos objetos, así como el tipo de datos de estas. Como se ha comentado en apartados anteriores, los objetos de tipo servicio tienen la propiedad “body” rellenada con la cadena de servicios que encapsula este servicio tarea:

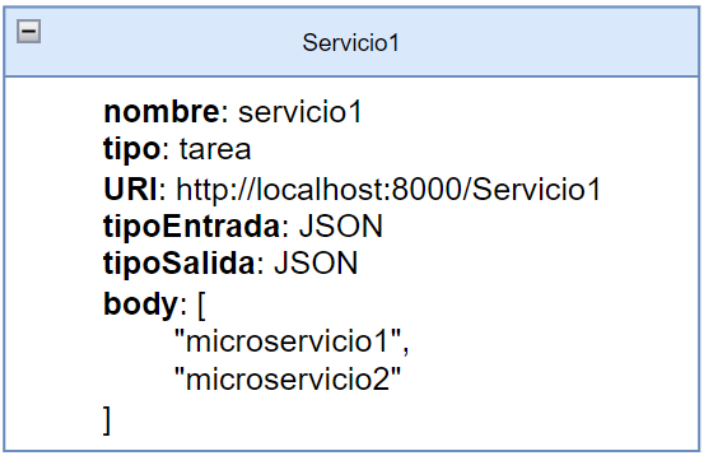

Figura 5. Ejemplo de Servicio (fuente propia).

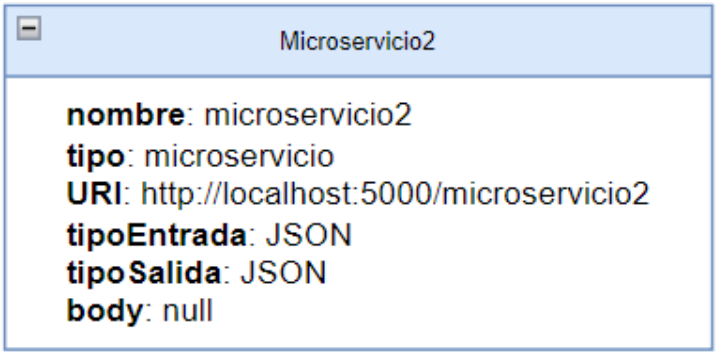

Figura 6. Ejemplo de Microservicio 1 (fuente propia).

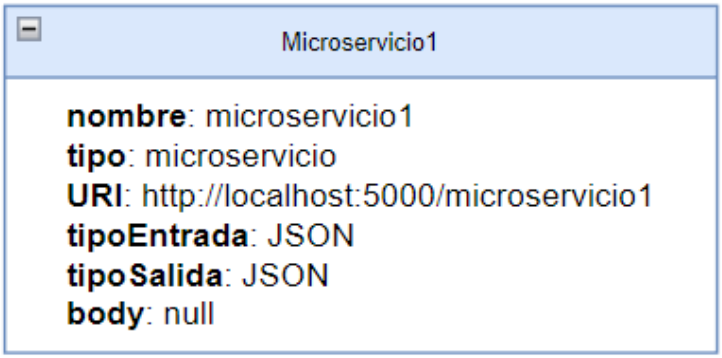

Figura 7. Ejemplo de Microservicio 2 (fuente propia).

En la figura 8 se muestra un esquema del flujo de ejecución cuando se realiza una llamada a al servicio tarea mostrado en el grupo de imágenes anterior. Las operaciones que publica el ESB son Crear, Borrar, Actualizar, Obtener y Ejecutar. Esta última se ha desglosado en subtareas para mostrar la lógica interna de esta operación:

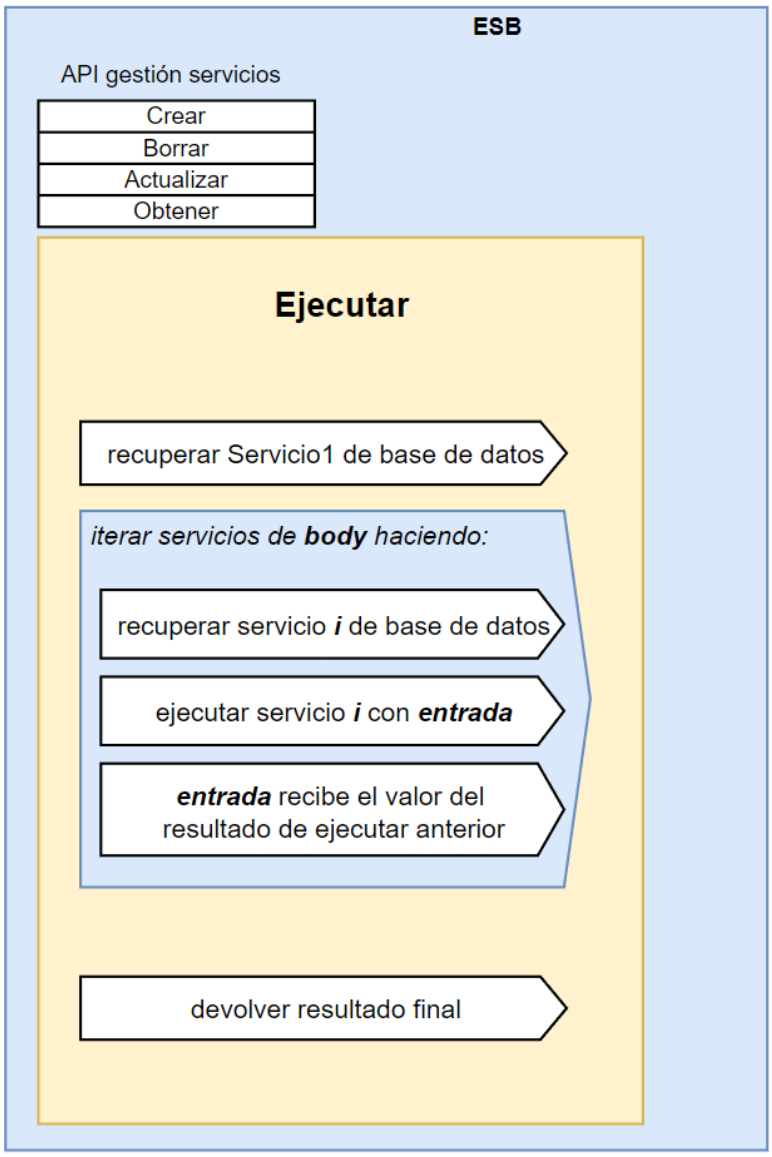

Figura 8. Esquema de acciones publicadas por el ESB y desglose de acción "ejecutar" (fuente propia).

En este caso el ESB recibirá una llamada al servicio con nombre “Servicio1” y lo buscará en la base de datos. Con este objeto servicio recorrerá el contenido de “**body”** recuperando el servicio “microservicio1” de base de datos y haciendo una llamada a su **URI** del host de microservicios.

A continuación, la respuesta de este microservicio se utilizará para la segunda iteración del bucle, en la que se recuperará el servicio “microservicio2” de base de datos y se hará una llamada a su **URI** de la misma manera que se hizo con el servicio “microservicio1”, pero esta vez utilizando como entrada la respuesta del servicio anterior (“microservicio1”).

Finalmente, el resultado de este segundo microservicio se devolverá como respuesta del servicio de tarea primero “Servicio1”, creando así una composición de servicios encapsulada en un servicio de tarea.

El servicio de tarea que se llama puede componer tanto microservicios, que reciben una entrada y devuelven un resultado, como servicios de tarea, que en esencia se comportan de la misma manera: reciben una entrada y devuelven un resultado. La diferencia es que un microservicio computa un resultado y lo devuelve, mientras que una llamada a un servicio de tarea (desde el “body” de otro servicio de tarea) llamaría de nuevo a la llamada de ejecución de servicios del ESB con este nuevo servicio, repitiendo el proceso explicado anteriormente. Este proceso recursivo permite la composición de cualquier listado variado de servicios encapsulado como un servicio de tarea.

A continuación, en la figura 9 se muestra un esquema de esta lógica de ejecución que contempla la posibilidad de que uno de los servicios que formen la propiedad "**body"** sea un servicio de tarea.

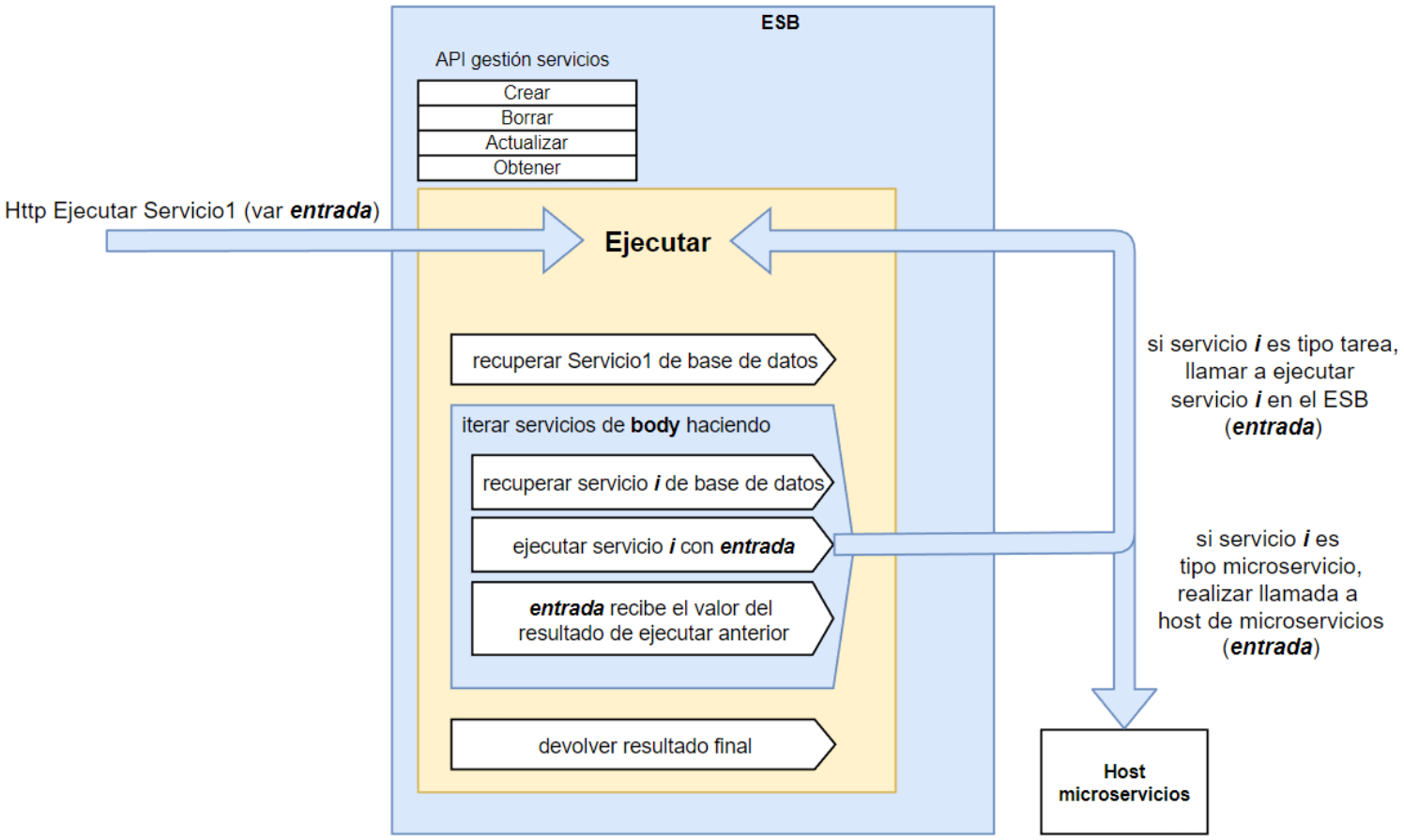

Figura 9. Lógica de la operación "ejecutar" que publica la API del ESB (fuente propia).

Un aspecto importante del ESB son las comprobaciones que realiza el ESB a la hora de crear servicios en cuanto a redundancia de servicios de tarea, ya que no está permitido por razones obvias la composición de servicios cíclicos porque causarían bucles infinitos de llamadas redundantes.

Otra comprobación importante a la hora de definir servicios es que el tipo de datos de los servicios agrupados sea compatible. Esto significa que la entrada del servicio de tarea tiene que coincidir con la entrada del primer servicio definido en el "body" de este servicio de tarea. Este primer servicio compuesto, por su parte, debe devolver una salida compatible con la entrada del siguiente servicio de la lista, y así sucesivamente para hacer posible el flujo de datos entre los servicios.

Esta aplicación Node también hospeda la aplicación web de gestión de servicios y la sirve como aplicación frontal de manera estática. El hecho de que esta aplicación frontal utilice esta aplicación Node como backend, simplifica en gran medida su complejidad de desarrollo y configuración. La aplicación web de gestión es una aplicación Angular, que, una vez compilada, se despliega en una ruta de este backend Node Express.

La arquitectura de una aplicación Angular se caracteriza por estar orientada al consumo de servicios web. Esta arquitectura cliente-servidor entre frontend (Angular) y backend (Node Express) es ideal para una aplicación web modular, escalable y mantenible en el tiempo por estar desacoplada de la implementación de backend.

En secciones posteriores se explicará con más detalle la arquitectura tanto de la aplicación web de gestión de servicios como de la relación entre esta aplicación y el ESB. En la figura 10 se muestra cómo el ESB sirve de manera estática la aplicación frontal Angular mientras esta hace uso de la funcionalidad que publica la API del ESB:

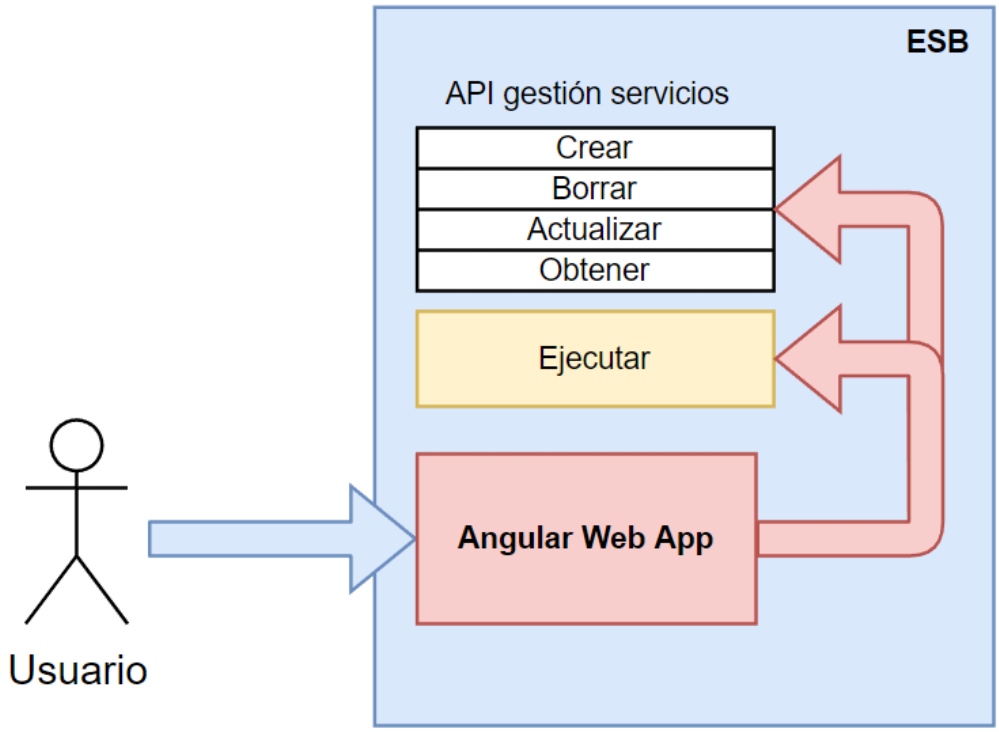

Figura 10. Relación del ESB y la aplicación web de gestión (fuente propia).

3.6.2. Host de Microservicios

El host de microservicios es en esencia una API web Flask que presenta llamadas de dos tipos: entrenamiento de modelos e inferencia. Las primeras actúan como instrucción de inicio para comenzar el entrenamiento de un modelo a partir de uno o varios sets de datos de entrenamiento ya presentes en la máquina.

Estas llamadas de entrenamiento lanzan procesos muy demandantes en cuanto a hardware por lo que el host de microservicios se hospeda en una máquina flexible en la nube, que soportará más o menos carga de procesamiento en función de la demanda. Esta flexibilidad optimiza la potencia de procesado para ahorrar costes de infraestructura a la organización.

En la figura 11 vemos un ejemplo de llamada a un servicio de entrenamiento, que inicia el proceso de generación de un modelo. Estos modelos serán identificables para ser relacionados con un servicio de inferencia en concreto.

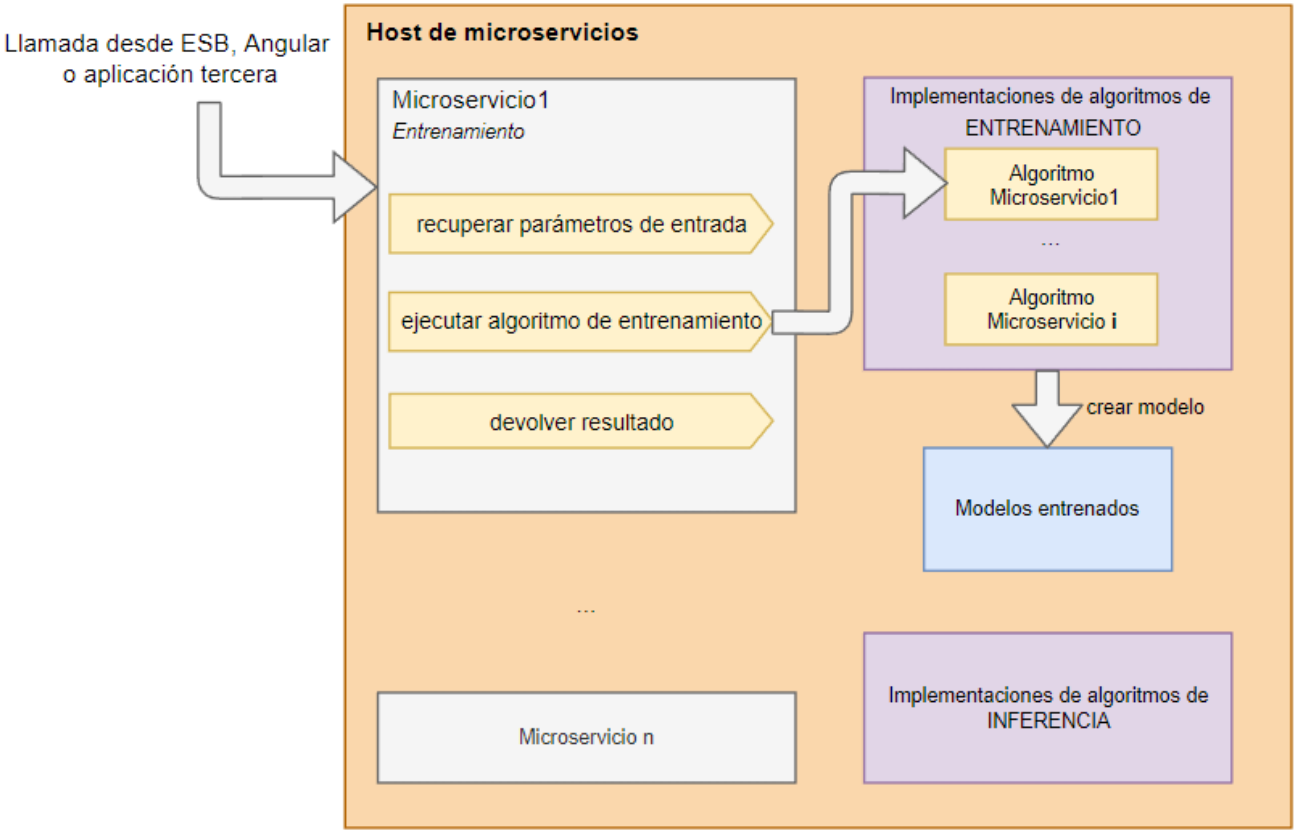

Figura 11. Lógica de las llamadas a servicios de entrenamiento (fuente propia).

Las segundas son llamadas de inferencia, que, a partir de unos datos de entrada (normalmente imágenes), computan el resultado utilizando los modelos entrenados previamente. Estos datos de entrada se envían codificados como cuerpo de la petición HTTP a la API, se reciben y decodifican, y comienza la lógica de inferencia implementada por el equipo de I+D.

En la figura 12 se muestra una llamada a un servicio de inferencia, que utiliza la colección de modelos creada anteriormente por servicios de entrenamiento para computar el resultado. Como comentaremos posteriormente, estas llamadas a servicios de inferencia tendrán un sistema de cacheado para peticiones recurrentes.

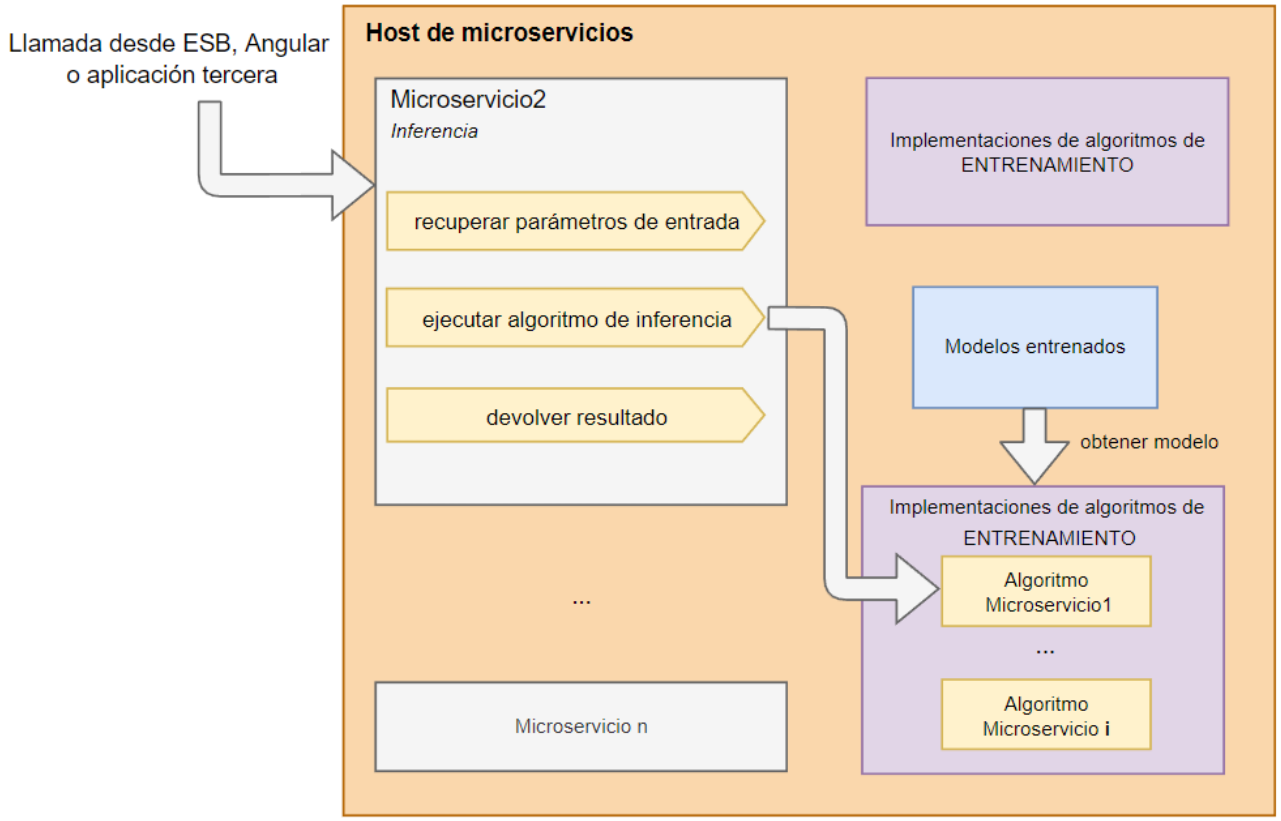

Figura 12. Lógica de las llamadas a servicios de inferencia (fuente propia).

Cacheado de microservicios

Como ya hemos comentado, las llamadas a la API REST Flask que presenta el host de microservicios pueden ser de dos tipos: de inferencia y de entrenamiento. Para esta API cachearemos las llamadas del tipo inferencia durante un tiempo determinado, porque es muy probable que se repitan llamadas idénticas a microservicios que implementan lógica de inferencia.

Aunque son las más exigentes en cuanto a potencia de procesamiento, las llamadas de tipo entrenamiento no tiene sentido que se guarden en la memoria cache porque cuando se hace una llamada de este estilo es porque se necesita entrenar el modelo de nuevo, muy probablemente con datos de entrenamiento diferentes.

Para las llamadas a microservicios de inferencia, esta política de cacheado guardará un registro de llamadas a la API con sus respectivos parámetros de entrada, y el resultado del microservicio al que llaman. Estos parámetros de entrada no suelen ser muy pesados por lo que no resultará un problema almacenar esta información temporalmente.

Este registro se usará para analizar cada llamada posterior en busca de llamadas ya registradas con parámetros de entrada idénticos a una llamada anterior. En caso de encontrarla, se devolverá el resultado que ya se computó para esa llamada, pues el resultado de la inferencia será similar y no tiene sentido ejecutar el microservicio de nuevo.

Se utilizará la librería Flask-Caching para implementar esta funcionalidad de cacheado de llamadas.

Comentaremos también que para las llamadas a la API del ESB no se guardará ningún tipo de cache ya que para las operaciones que publica no tiene sentido guardar llamadas anteriores.

3.6.3. Plataforma de gestión

Finalmente, explicaremos la plataforma web de gestión de servicios. Como ya hemos comentado, esta aplicación es el principal punto de entrada al sistema, tanto para perfiles desarrolladores de servicios como para consumidores. Es una aplicación web frontal Angular que sigue su característica arquitectura orientada a componentes y al consumo de servicios web.

Como característica a destacar, durante el desarrollo se ha utilizado el lenguaje de programación TypeScript para facilitar la creación de clases, interfaces y tipos de datos, aportando robustez y mantenibilidad al código.

Los diferentes componentes UI de la aplicación Angular actúan de vista y controlador desde la parte frontal, mientras que el acceso a la capa de datos se realiza a través de servicios Angular, que se encargan de gestionar la comunicación con los servicios de gestión que publica el backend del ESB. Con esta arquitectura típica de Angular se consigue desacoplar la visualización de los datos de las operaciones en la capa de persistencia, mejorando la gestión de errores y los tiempos de carga y experiencia de usuario.

Estos servicios están divididos en servicios de gestión de datos (operaciones sobre el recurso Servicio) y servicios relacionados con autenticación y autorización, que detallaremos más adelante.

Esta aplicación se prepara para desplegar en la aplicación Node del ESB mediante un comando Angular que crea un paquete JavaScript, HTML y CSS minificado.

Como ya hemos mencionado, se hace uso de la librería de componentes Angular Material para aportar componentes de UI y crear una experiencia de usuario fluida, intuitiva y agradable, a la vez que se agiliza el desarrollo y mantenimiento de la aplicación.

Esta parte frontal incluye control de autenticación y autorización mediante una página de inicio de sesión y una estructura de roles, creando los roles de administrador, desarrollador de servicios y consumidor de servicios. El acceso está protegido mediante autenticación JWT, como viene siendo costumbre últimamente en las aplicaciones web. En la figura 12 vemos la página de inicio de sesión de la aplicación.

Figura 13.Interfaz de inicio de sesión de la aplicación web de gestión (fuente propia).

La aplicación se divide en dos partes principales: servicios de tarea y microservicios. Cada sección incluye tanto la visualización como la edición, creado y borrado de ese tipo de servicios. En las siguientes figuras (13 y 14) se muestran las interfaces de estas partes de la aplicación.

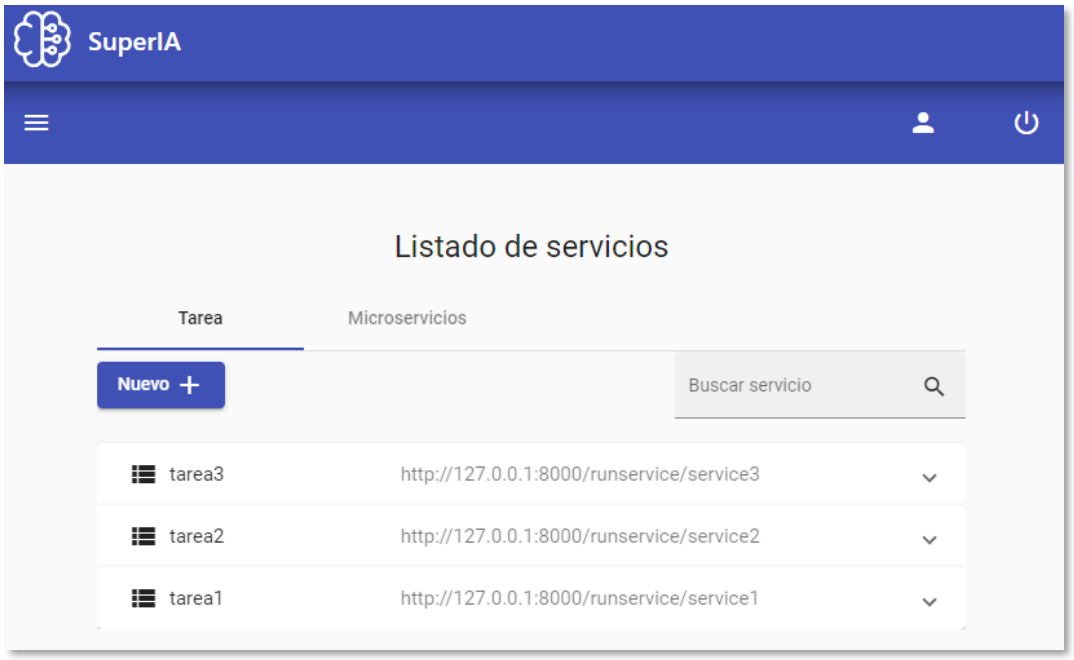

Figura 14. Listado de servicios de tarea (fuente propia).

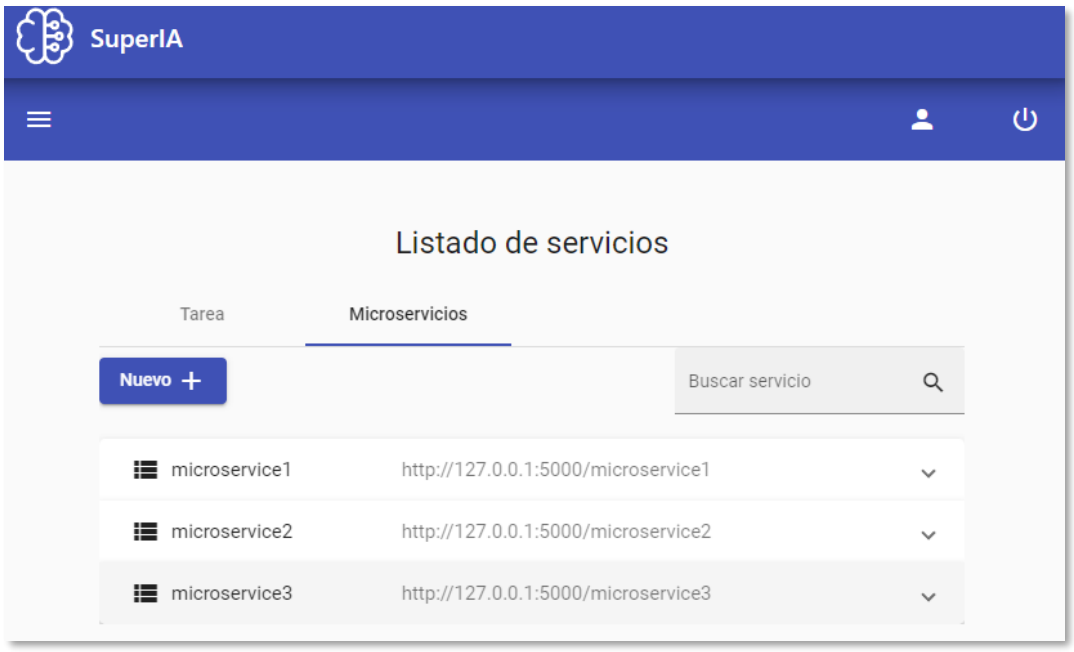

Figura 15 Listado de microservicios (fuente propia).

Dentro de la visualización de un servicio, está la funcionalidad de ejecutar el mismo servicio proporcionando unos datos de entrada desde esta misma web. Esta funcionalidad replica el caso de uso de utilizar un servicio o microservicio desde una aplicación tercera a través de HTTP.

En las siguientes figuras (15 y 16) se muestran los componentes UI de un servicio de tarea y de un microservicio. Como se puede observar, se muestra el contenido del objeto servicio con sus propiedades de manera estructurada, además de los botones y campos de texto para buscar, editar, eliminar y crear servicios.

En el caso del servicio de tarea (figura 15) vemos que la interfaz también muestra el flujo de llamadas a servicios que encapsula su propiedad "**body"**.

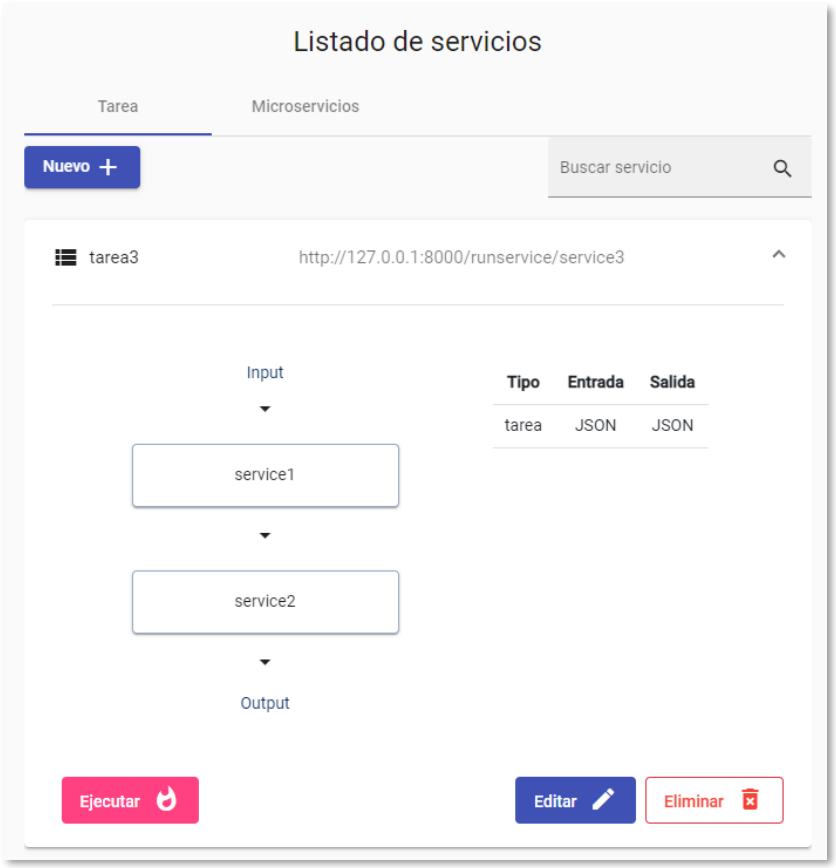

Figura 16. Desglose de servicio de tarea (fuente propia).

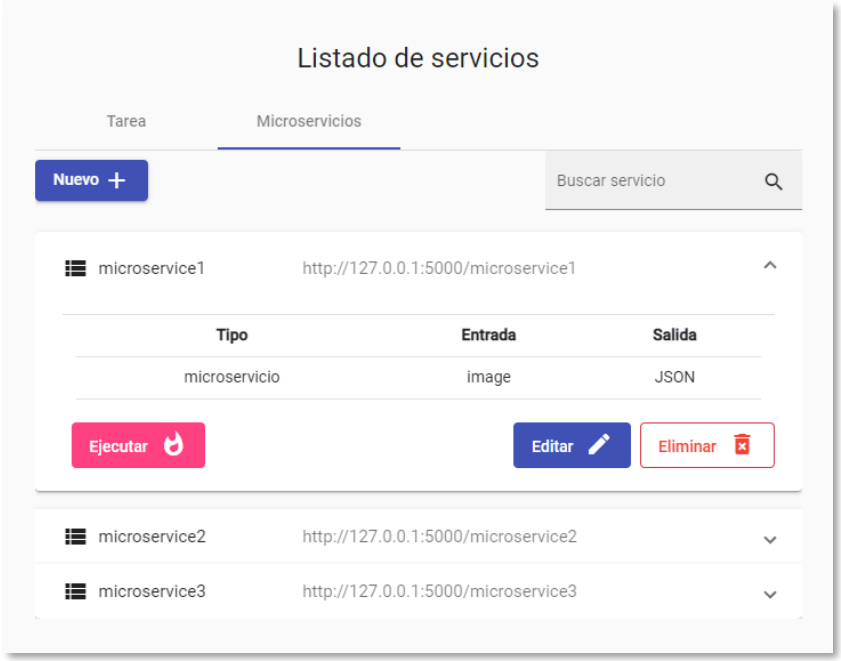

Figura 17. Desglose de microservicio (fuente propia).

Tanto servicios de tipo tarea como microservicios pueden ser ejecutados desde esta interfaz web ya que ambos están implementados como servicios web en sus respectivas APIs.

En las siguientes figuras mostramos la funcionalidad de la aplicación web de gestión que ejecuta un servicio, emulando el caso de uso en el que se realiza una llamada a un servicio desde una página web externa. En este ejemplo veremos cómo se ejecuta un servicio desde la aplicación de gestión. Este servicio recibe una entrada de tipo imagen que es codificada en formato base64 para ser enviada por HTTP al host de microservicios. Más concretamente, el siguiente microservicio recibe una imagen y devuelve una predicción del contenido de esta.

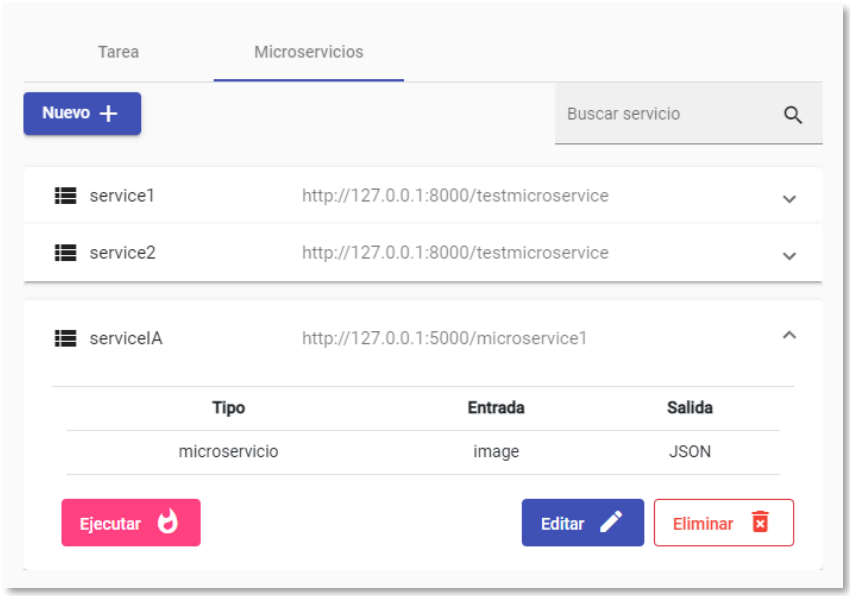

Figura 18. Desglose de microservicio de inferencia que recibe una imagen como parámetro de entrada (fuente propia).

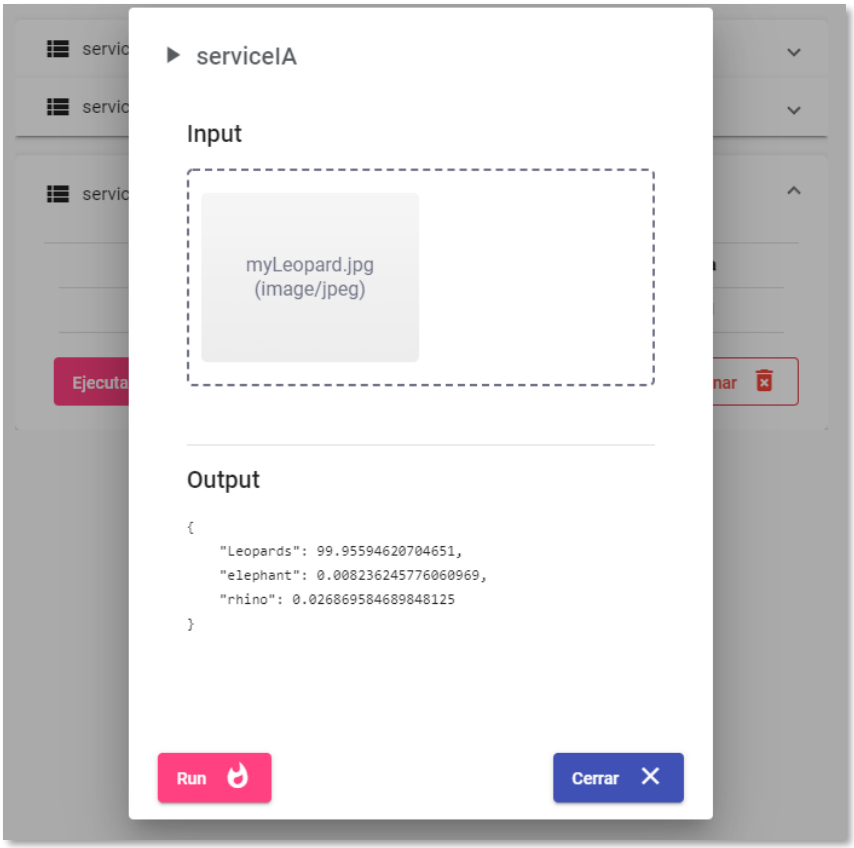

Figura 19. Desglose de microservicio de inferencia que recibe una imagen como parámetro de entrada 2 (fuente propia).

La creación y edición de servicios es similar para los dos tipos de servicios ya que utilizan el mismo tipo de datos.

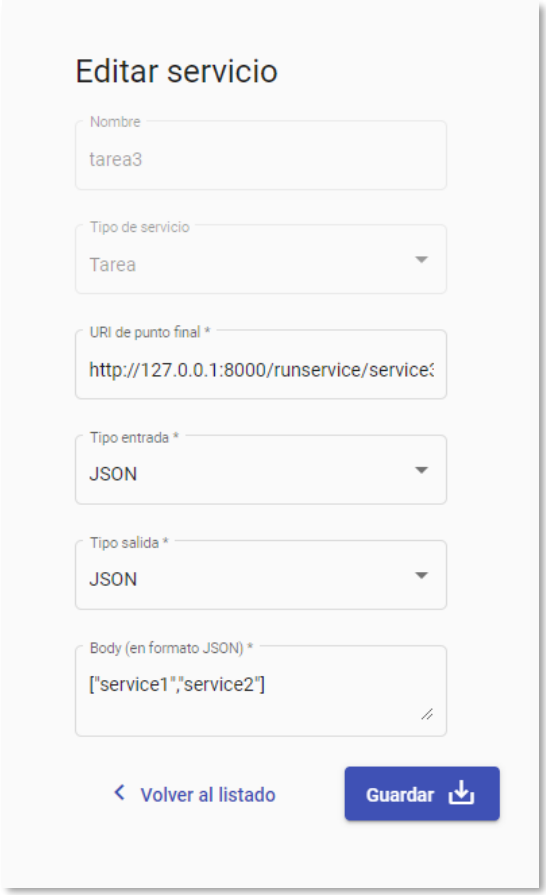

Figura 21. Interfaz de edición de servicios (fuente propia).

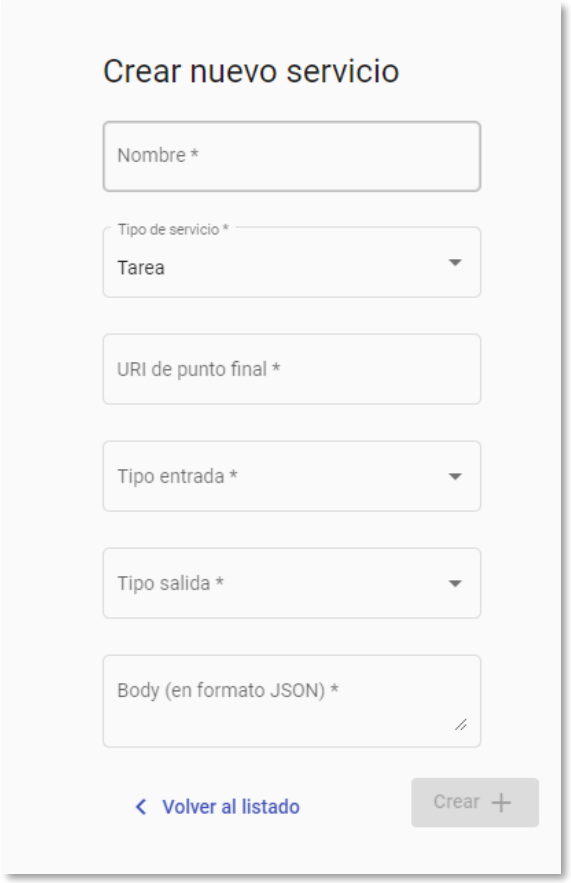

Figura 20. Interfaz de creación de servicios (fuente propia).

Finalmente, en un futuro se añadirá funcionalidad para que el rol de administrador pueda gestionar accesos y crear usuarios, de manera que ya no tendrá que accederse a la base de datos directamente como se venía haciendo hasta ahora de manera provisional.

3.7. Evolución del desarrollo e implementación

En esta sección detallaremos el proceso de diseño, desarrollo e implementación de la solución SOA que hemos presentado. El objetivo del proyecto ha sido la implementación del sistema SOA en una empresa de desarrollo software que posee un equipo orientado al desarrollo de soluciones de machine learning y visión artificial.

3.7.1. Recolección de requisitos

El proceso de recolección de requisitos se llevó a cabo como primera fase del proyecto en conjunto con el equipo de I+D de la empresa en la que se ha implementado la solución. El cliente principal es dicho equipo por lo que con su ayuda se consiguió una alta precisión en la definición de casos de uso. Estos casos de uso se crearon en base al flujo de trabajo habitual de este equipo, que en primera instancia trabajaba implementando los algoritmos en sus propias máquinas y traspasándolos a una máquina compartida más potente para su ejecución.

Otro requisito importante ha sido la flexibilidad de la que se ha dotado al sistema gracias al requisito de la utilización de Docker sobre máquinas en Azure para crear un sistema modular, flexible y escalable. En la misma línea, la diferenciación entre los algoritmos de inferencia y los de entrenamiento descrita en secciones anteriores apoyó la necesidad de aplicar este tipo de implementación.

Por último, comentaremos un requisito que nació principalmente de la naturaleza SOA del sistema, el diseño y construcción del inventario de servicios, que encapsularía toda la funcionalidad implementada.

Estos aspectos técnicos de la plataforma como el rendimiento, la interoperabilidad y la flexibilidad en cuanto a modularización y potencia de procesamiento se han visto enormemente apoyados por la actividad de este equipo de I+D en la recolección de requisitos.

La recolección de requisitos, aunque se desarrolló de manera intensiva antes de comenzar con el diseño y la implementación, se ha ejecutado como un proceso iterativo e incremental, ya que con SCRUM es muy común que se modifiquen o creen requisitos en cada entrega.

3.7.2. Diseño del sistema

Tras completar diferentes fases de recolección de servicios se comenzó con el diseño de las diferentes partes del sistema.

Desde el principio el objetivo fue crear una aplicación encargada del hospedaje, publicación y ejecución de los algoritmos IA, el **Host de microservicios**, y otra aplicación para llevar a cabo la gestión y registro de los servicios de tarea y el registro de los microservicios IA. Como hemos visto en apartados anteriores, esta última implementación se ha dividido en dos, el **ESB** y la **Aplicación Web de Gestión**.

ESB

Esta primera es una aplicación destinada principalmente a la publicación de las operaciones para gestionar el ciclo de vida del servicio, la "orquestación de servicios" y la base de datos.

Dentro del diseño del ESB comentaremos que para el proceso de comunicación entre servicios se determinó que se utilizaría REST sobre HTTP, como se ha explicado en secciones anteriores. En esta fase de diseño se determinó también la estructura que seguirían los mensajes, en formato JSON, para terminar de estandarizar las comunicaciones.

Se diseñaron también las posibles operaciones sobre los servicios. Esta tarea se simplificó gracias a la utilización del patrón REST en la API que presenta el ESB. De esta manera, cada método HTTP se correspondía con una operación sobre el recurso REST Servicio.

Dentro del diseño del ESB se ha incluido el diseño de la base de datos, que al tratarse de una base de datos no relacional el proceso tradicional de diseño y creación de la infraestructura (creación de tablas, claves, relaciones…) se simplifica en gran medida. Resumiendo, se diseñaron las colecciones Servicio y Usuario. La primera define los servicios tanto de tipo tarea como de tipo microservicio, y la segunda se encarga del almacenado de usuarios de la aplicación web de gestión.

Por último, mencionaremos la que es probablemente la funcionalidad más interesante y clave que se diseñó para el ESB. Esta funcionalidad es la que equivaldría a la orquestación de servicios en sistemas distribuidos, pues se encarga de procesar los flujos de información que definen los servicios del tipo tarea y de enrutar unos servicios con los siguientes. La explicación de esta funcionalidad está en la sección 3.6.1. ESB – Ejecución de servicios.

Aplicación web de gestión

Esta aplicación se trata de una interfaz web que encapsula la gestión de servicios implementada en la aplicación anterior para los diferentes tipos de usuario final, por lo que, gracias al diseño anterior, el diseño de la aplicación web de gestión se simplificó enormemente.

El objetivo principal de esta aplicación es ofrecer al usuario un acceso al inventario de servicios transparente para cumplir con los objetivos de gobernabilidad y visibilidad de un sistema SOA.

Para esta aplicación podemos destacar del diseño de los diferentes componentes UI, como los listados de los dos tipos de servicios, o su funcionalidad de edición, creación y ejecución. Estos diseños se han llevado a cabo priorizando la usabilidad y el diseño funcional e intuitivo sobre lo artístico y estético.

En cuanto a la arquitectura interna angular, como se ha comentado en secciones anteriores, se diseñó una infraestructura únicamente orientada al consumo de los servicios web que publica el ESB. Todos los componentes y servicios Angular tienen esta orientación hacia el ESB.

La aplicación Angular utiliza además el protocolo de autenticación JSON Web Token para implementar la autenticación y autorización entre el usuario, la aplicación web de gestión y el ESB.

Host de microservicios

Como comentamos, esta parte del sistema se encarga de publicar una interfaz a través de la cual acceder a la ejecución de servicios de inteligencia artificial.

Para comenzar, se definieron dos tipos de servicios que funcionarían de manera independiente a la vez que complementaria: servicios de entrenamiento y servicios de inferencia.

Los primeros implementarían la funcionalidad que genera los modelos de datos necesarios para llevar a cabo los procesos de inferencia que implementarían los segundos. Estos modelos se almacenan en forma de ficheros en el propio host de microservicios, y se generan a partir de procesos de entrenamiento utilizando diferentes tipos de datos de entrada.

Los microservicios de inferencia recibirían una serie de parámetros de entrada, con los que ejecutarían el proceso de inferencia haciendo uso de los modelos generados previamente por procesos de entrenamiento.

Por supuesto, estos servicios siguen los mismos patrones de diseño y comunicación que los servicios de tarea gestionados por el ESB.

En cuanto al hardware que soportará estos procesos de inferencia y entrenamiento, es necesario contar con la posibilidad de experimentar picos de demanda de potencia, principalmente para los servicios de entrenamiento. Es por esto que la implementación se ha diseñado pensando en un despliegue en forma de contenedor sobre máquinas en Azure, para proporcionar flexibilidad en estos casos de fluctuación en la demanda de potencia de procesamiento.

Finalmente comentaremos la funcionalidad de cacheado que se diseñó para el host de microservicios, con la que se cachearían las respuestas de los servicios de inferencia siguiendo unas reglas, pues son los más susceptibles a recibir peticiones similares.

Como aspecto transversal del diseño del sistema tenemos la presencia de Swagger para llevar a cabo tareas de documentación y visibilización de las APIs y servicios. Gracias a las diferentes implementaciones de APIs REST es posible generar con Swagger documentación dinámica de las diferentes llamadas mejorando el descubrimiento de servicios de cara a terceros.

3.7.3. Implementación modular, dockerización y despliegue

Resumiendo, el diseño explicado en el apartado anterior, nuestro sistema consta de tres aplicaciones principales: ESB, Aplicación Web de Gestión y Host de Microservicios. La implementación de todas ellas se ha llevado a cabo con el objetivo de crear aplicaciones modulares y fácilmente integrables, mediante el despliegue en contenedores.

En esta sección resumiremos el proceso de implementación realizado para cada aplicación, junto con las tecnologías y arquitectura utilizados, siguiendo los diseños explicados en el apartado anterior.

ESB

El ESB consta de una API REST que implementa diferentes operaciones sobre el recurso Servicio. Se ha implementado como una aplicación Node Express que junto con la base de datos MongoDB, presenta las operaciones para crear, editar, obtener y eliminar servicios. Esta parte de la aplicación sigue una arquitectura API REST tradicional, haciendo uso de los métodos HTTP GET, POST, PUT y DELETE como base.

Por otro lado, tenemos la llamada de ejecución de servicios. Esta llamada puede lanzarse sobre un servicio de tarea o microservicios registrado en el ESB. En el caso de los microservicios, el motor de enrutado redirige la petición hacia el Host de Microservicios junto con la entrada, y devuelve la respuesta recibida al cliente. En el caso de las llamadas de ejecución a servicios de tipo tarea el comportamiento es diferente y ha sido ya explicado en el su apartado de diseño del ESB.

Entre las tecnologías utilizadas para apoyar la funcionalidad de esta aplicación Node Exrpress tenemos body-parser, cors, Morgan o Mongoose.

Aplicación Web de Gestión

Esta aplicación es una aplicación web Angular cuya función principal es la de presentar una interfaz web al usuario que gestiona el inventario de servicios. Esta aplicación es la principal consumidora de las operaciones que publica el ESB.

Esta aplicación Angular utiliza la librería de estilos Angular Material para crear una experiencia intuitiva y agradable en la gestión del ciclo de vida del servicio.

En cuanto a la arquitectura interna de la aplicación Angular remarcaremos la orientación a componentes que propone el propio frameworks, con las que se han construido diferentes componentes UI reutilizables. Comentaremos también la utilización de typescript para crear estructuras de datos robustas para esta parte frontal.

Host de Microservicios

La implementación de la arquitectura del Host de Microservicios se resume en una implementación de una API REST – Python Flask. La API publica las diferentes llamadas tanto a microservicios de inferencia como a microservicios de entrenamiento.

El punto interesante de esta aplicación es el despliegue mediante contenedores Docker sobre máquinas en Azure. Estas máquinas son altamente configurables de manera dinámica lo que las vuelve ideales para sistemas con necesidades de potencia de procesamiento variables.

Capítulo 4 – Conclusiones y líneas futuras

En este capítulo concluiremos este trabajo sobre el diseño e implementación de un sistema SOA orientado a la provisión de servicios y microservicios de inteligencia artificial. Ha sido un largo viaje en el que se han realizado diferentes investigaciones y estudios tanto de la industria actual como de otros trabajos de investigación, con el fin de conseguir un trabajo de la máxima calidad.

Primeramente, comentaremos el proceso de implementación del sistema evaluando diferentes aspectos de sus componentes, como las tecnologías utilizadas o los patrones y arquitecturas elegidos, y se continuará con un resumen de los puntos fuertes y débiles del sistema junto con un plan de acción para continuar mejorando la plataforma. Finalmente se hablará del proceso de despliegue mediante contenedores Docker sobre máquinas Azure.

Seguidamente, se explicará el impacto que tendrá el despliegue en producción de esta herramienta en el entorno empresarial y cómo afectará tanto al equipo de I+D con el que se ha venido trabajando durante el desarrollo del sistema. Se comentará también el impacto de la plataforma en los equipos terceros que hacen uso de estas soluciones IA compuestas ya que son los usuarios finales que utilizarán los servicios de tipo tarea.

Para finalizar, se hablará del futuro del proyecto tanto en el entorno empresarial evolucionando dentro de la organización para mejorar el servicio para los equipos, y las posibilidades de evolución fuera de la empresa aplicando diferentes tipos de monetización a la plataforma, como sistemas de suscripción o alquiler por uso.

4.1. Evaluación de la implementación SOA

Como bien hemos dicho, este proyecto ha consistido en la implementación de una arquitectura SOA para un entorno empresarial, centrado en servir soluciones basadas en microservicios IA.

Durante el proyecto, y tras el estudio e investigación de diferentes trabajos en el campo, se han diseñado e implementado diferentes componentes característicos de las arquitecturas que siguen el patrón SOA, como son, entre otros, el inventario de servicios, el ESB, los protocolos y estándares para el intercambio de mensajes, o la propia unidad de servicio.

Destacaremos que durante el proceso de implementación se ha seguido la metodología ágil SCRUM, con la que se ha trabajado por ciclos creando piezas de código funcionales para mostrar al cliente. Esta metodología involucra en gran medida al cliente, y permite realizar un seguimiento continuo y muy cercano para corregir aspectos de la implementación en cada ciclo y así evitar desviaciones respecto a las necesidades el usuario final.

Se han investigado y estudiado multitud de trabajos de investigación, que se han ido comentando a lo largo del trabajo, relacionados con la implementación de sistemas SOA, diseño de infraestructuras de servicios o metodologías para aplicar gobernanza sobre el ciclo de vida del servicio. Adicionalmente, como se detalló en el apartado 2.3.4., se han estudiado y probado herramientas e implementaciones utilizadas en el diseño y desarrollo de sistemas SOA, como aplicaciones de gestión de servicios y gobernanza, implementaciones software para llevar a cabo el intercambio de mensajes o hasta implementaciones completas del bus de servicios empresariales.

Este trabajo de investigación previo al diseño e implementación de nuestro sistema ha enriquecido enormemente los conocimientos de nuestro equipo, aportando diferentes puntos de vista y ejemplos de implementaciones de componentes y conceptos SOA. La calidad del proyecto se ha visto enormemente propulsada por esta serie de estudios previos.

4.1.1. Aplicación Web de Gestión

Para llevar a cabo el análisis de la implementación de la Aplicación Web de Gestión se han realizado varias pruebas tanto con el equipo de I+D con el que se ha venido trabajando como con otros usuarios en el rol de consumidores de servicios. Tras estas pruebas, podemos concluir que es una plataforma sencilla de utilizar y que muestra claramente las funcionalidades implementadas. Junto con la usabilidad propuesta por la interfaz, otra característica clave es sin duda la transparencia con la que se implementa la gestión del inventario de servicios, presentando claramente el estado de cada servicio y microservicio, las acciones que se pueden realizar sobre ellos y la constante sincronización con la base de datos.

La interfaz es robusta en la gestión de errores y posee mecanismos para informar al usuario de qué ha ocurrido y por qué, creando una experiencia de usuario en la gestión del ciclo de vida del servicio intuitiva, satisfactoria y transparente. La aplicación posee además un diseño de interfaz responsive por lo que se puede utilizar desde cualquier dispositivo.

Esta interfaz permite además el testeo de los servicios registrados para probar el despliegue de nuevos servicios y facilitar las tareas de detección de errores.

4.1.2. ESB

La implementación realizada del ESB cumple con todos los objetivos propuestos al inicio de este proyecto, así como con la mayor parte de la funcionalidad propuesta por las herramientas del mismo tipo que encontramos en el mercado. Si bien es cierto que técnicamente puede considerarse una implementación simple, la lógica interna de enrutado cumple perfectamente su función de enlazar y componer servicios y microservicios heterogéneos. El ESB cumple también su función de mantener la estandarización y la aplicación de protocolos en el sistema, tanto a los mensajes entre servicios como a los servicios y microservicios creados y registrados.

Además, la implementación del ESB permite monitorizar y trazar la ejecución de los flujos de servicios tarea y microservicios de una manera intuitiva para el usuario desarrollador.

El motor de enrutado del ESB es altamente escalable dada la ligereza de las tecnologías que utilizan sus interfaces de comunicación (HTTP y REST), siendo posible incluso aumentar la potencia de procesamiento de peticiones gracias a la orientación al despliegue en contenedores Docker que se ha venido trabajando.

Respecto al servicio, como comentamos en el Capítulo 2, SOA define unas pautas a seguir para su diseño. Nuestros servicios siguen un contrato estandarizado en cuanto a que utilizan el protocolo de comunicación REST sobre HTTP, por un lado, y una estructura similar, organizada y comprensible para el contenido de los mensajes, por otro. Esto aporta abstracción y componibilidad a los servicios, permitiendo la creación de composiciones de servicios.

El hecho de que todos los servicios se presenten en APIs REST que utilizan HTTP aporta también un alto nivel de visibilidad a cada servicio de manera individual, facilitando la tarea de creación del inventario de servicios. La creación del inventario de servicios, como veremos, está fuertemente apoyada en el diseño del ESB y la aplicación de gestión de servicios.

Los servicios son implementaciones desacopladas de otras funcionalidades o servicios, por lo que cada ejecución puede completarse sin dependencias. Esta autonomía se consigue para los microservicios de inteligencia artificial ya que son implementaciones individuales en una aplicación que permite su ejecución en paralelo.

Podría discutirse que los servicios de tarea no comparten esta autonomía, pero no sería razonable, pues al componer servicios el objetivo es crear dependencias entre los mismos. Los servicios de tarea dependen de la conclusión de los servicios que encapsulan, pues ese es su objetivo, componer diferentes servicios para crear soluciones complejas.

La sinergia de la Aplicación Web de Gestión con el ESB crea una implementación del inventario de servicios escalable, reutilizable y transparente, consiguiendo alcanzar los estándares de gobernanza característicos de implementaciones SOA.

4.1.3. Host de Microservicios

El host de Microservicios implementa los algoritmos IA en forma de microservicios independientes, autónomos y sin estado, y los publica en una API REST accesible mediante HTTP. Esto aporta un alto nivel de accesibilidad y estandarización a estos servicios a pesar de ser implementaciones completamente heterogéneas.

El flujo de trabajo del equipo de I+D a la hora de crear nuevas soluciones mejora pues de ahora en adelante pueden reutilizar código implementado en otros microservicios realizando llamadas a la interfaz del ESB, asegurando además que estas piezas de código siguen estándares de implementación que aseguran rendimiento y seguridad en el algoritmo.

De nuevo, la implementación enfocada al despliegue en forma de contenedores brinda la posibilidad de escalar la potencia de procesamiento para los algoritmos más pesados, por un lado, y para procesar altas cargas de peticiones a la API por otro. De esto se encargará la configuración que se aplique a las máquinas Azure. A esto se le suma la utilización de Swagger para apoyar los procesos de generación y mantenimiento de la documentación de las interfaces de comunicación, cumpliendo los objetivos de visibilidad y estandarización planteados para esta parte del sistema.

Con el diseño y junto a la elección de estándares y tecnologías de implementación podemos decir que se cumplen la mayoría, si no todos, los estándares de una arquitectura SOA y los objetivos planteados al inicio de este trabajo. Todas las partes del sistema trabajan de manera coordinada aportando estandarización, desacoplamiento, reutilización, autonomía, visibilidad y componibilidad y abstracción a la plataforma.

4.1.4. Evaluación global

A continuación, comentaremos diferentes puntos de la implementación que presentaron problemas, y explicaremos las acciones que se tomaron para resolverlos.

Para comenzar, el componente que se esperaba que causara más problemas es el ESB. Al tratarse de la pieza central que coordina y soporta el resto de la infraestructura SOA es necesario que funcione de una manera fluida y robusta.

Del ESB, el motor de enrutado es la pieza más compleja, y, como veremos, tuvimos que resolver una serie de problemas relativos a la lógica de enrutamiento y la visibilidad de los servicios creados.

Motor de enrutado

En un primer diseño de esta parte de la aplicación, al recibir una petición de creación de servicio de tarea, el motor de enrutado creaba un fichero con el código para gestionar la llamada de ejecución al servicio, y editaba el fichero de rutas de la API del propio ESB para añadir la nueva ruta a este nuevo servicio tarea. Esto causaba varios problemas como la necesidad de reiniciar la API del ESB para que se actualizara la interfaz con la nueva ruta.

Otro problema de esta arquitectura del motor de enrutado es que con la creación de un fichero para contener la lógica de ejecución de cada servicio tarea se dificultaría el mantenimiento del sistema.

Esta arquitectura era inviable, por lo que se reinventó el motor de enrutado convirtiéndolo y simplificándolo para evitar crear y actualizar ficheros en tiempo de ejecución. Ahora los servicios de tarea contienen la información del flujo de servicios que encapsulan en base de datos, por lo que el motor del ESB solo necesita conocer su nombre para acceder a ellos y comenzar con el bucle de ejecución síncrona de servicios.

Las llamadas a los servicios de tarea a la API del ESB se gestionan mediante una ruta “comodín” que acepta cualquier llamada a servicio, y en su lógica es donde se distingue a qué servicio se está llamando para recuperar su información de la base de datos. Esta lógica se explica en detalle en el apartado Ejecución de servicios de la sección 3.6.1.

Entrada de datos para los servicios

Otro aspecto que resultó complicado de implementar y afectaba tanto a la Aplicación Web de Gestión como al ESB y al Host de Microservicios fue la gestión de la entrada de datos para las llamadas a los servicios.

Al tratarse de microservicios de inteligencia artificial, en muchos casos los datos de entrada no son simples objetos en formato JSON, sino que son imágenes o archivos en diversos formatos. Por esto, durante la implementación hubo que investigar cómo implementar estos casos de uso, y se resolvió rápidamente ya que es posible codificar archivos (en formato “base64” por ejemplo) y enviarlos a través de HTTP entre aplicaciones.

Vulnerabilidades y soluciones

Aunque la implementación del sistema SOA haya resultado exitosa de manera general, hay ciertos aspectos que pueden perfeccionarse. Remarcaremos también que, al implementar el sistema en un entorno cerrado y controlado, inaccesible desde el exterior de la empresa, se han ignorado algunos aspectos de seguridad en redes que en entornos de producción abiertos a internet son de suma importancia.

En primer lugar, destacaremos la utilización de HTTP y no de HTTPS. Esto se ha realizado así para agilizar las tareas de desarrollo y no tener en cuenta la instalación de certificados en las máquinas que hospedan las aplicaciones del sistema. Para el paso a entornos de producción, obviamente, se implementará la infraestructura necesaria para trabajar con HTTPS y no con HTTP.

Como hemos explicado en secciones anteriores, la Aplicación de Gestión de Servicios y el ESB utilizan el estándar JWT para implementar la autorización y autenticación con la API que publica el ESB. Aunque este estándar esté ampliamente aceptado por la comunidad, hay que tener en cuenta algunos tipos de ataques que pueden romper esta seguridad. Mencionaremos los que tienen más probabilidades de ocurrir: ataque de fuerza bruta contra el secreto del JWT o la confusión de algoritmo de encriptación.

Sobre las posibles vulnerabilidades que puedan presentar las máquinas en Azure que hospedan el sistema, es cierto que estadísticamente estas nubes son con diferencia más seguras que la mayoría de los sistemas propios empresariales, pero siempre hay posibilidades de sufrir ataques y debemos que contar con ello.

A continuación, hablaremos de algunas posibles soluciones a los puntos débiles anteriores. La mayoría de ellos nacen de la implementación en un entorno controlado inaccesible desde el exterior como la utilización de HTTP en vez de HTTPS. Como ya hemos comentado, más adelante se actualizará el sistema para comenzar a utilizar HTTPS en todas sus interfaces de comunicación.

Para mitigar las posibles vulnerabilidades de JWT el equipo desarrollador seguirá investigando y mejorando la infraestructura de autenticación y autorización del ESB y la Aplicación Web de Gestión para aplicar políticas en el código que impidan aprovechar estas vulnerabilidades. Este será un proceso continuado en el tiempo para mejorar y asegurar las comunicaciones con el ESB.

Finalmente, sobre el despliegue en contenedores en la nube de Azure, desde nuestra posición no podemos hacer más que configurar la máquina correctamente para impedir cualquier conexión no deseada. Actualmente con el despliegue controlado en la empresa, evitando el acceso desde internet se evitarán la gran mayoría de vulnerabilidades que puedan presentar las tecnologías o protocolos utilizados.

4.2. En la empresa

Habiendo realizado una evaluación de cada parte del sistema SOA implementado, es hora de comentar del impacto que tendrá nuestro proyecto en la empresa.

Para empezar, recordaremos que este proyecto se ha llevado a cabo con la intención de apoyar un equipo de investigación y desarrollo de soluciones IA, en concreto dentro del campo de la visión artificial. La empresa es una organización mediana dedicada principalmente al desarrollo software, en su mayoría aplicaciones y sistemas web. El equipo de I+D tiene como objetivo aportar soluciones a problemas concretos en los que un desarrollo propio del campo del machine learning sea útil.

Con esto en mente, se ha diseñado e implementado este sistema SOA que pretende crear una plataforma donde registrar los algoritmos IA desarrollados en forma de microservicios web para ser reutilizados y compuestos, creando soluciones complejas a partir de microservicios sencillos.

El equipo de I+D ha estado desde el primer momento involucrado en el diseño y desarrollo del sistema, por lo que el resultado está muy alineado con sus necesidades. Con este sistema, el equipo de I+D creará los algoritmos como venían haciendo hasta ahora, pero además los registrarán en la plataforma SOA en forma de microservicios web. Las soluciones que desarrollen a partir de ahora estarán formadas por composiciones de estos microservicios, que, en vez de estar implementados directamente en el código de la solución, estarán presentes en forma de llamadas a la interfaz de comunicación del ESB.

Esta migración supondrá un cambio en la filosofía de diseño de estas soluciones IA. Estos microservicios deberán seguir pautas de diseño e implementación para aportar la reusabilidad y estandarización que requiere el sistema SOA, permitiendo su reutilización en nuevas composiciones de servicios.

El equipo de I+D será responsable de seguir estas pautas de diseño e implementación de servicios, que, junto con la interfaz de la Aplicación de Gestión de Servicios, dotarán de la gobernanza sobre el ciclo de vida de los servicios que persiguen las arquitecturas SOA.

Además del equipo de I+D, otros equipos de la empresa se verán afectados por este nuevo sistema. Estos equipos utilizaban en ocasiones las soluciones tradicionales IA que venía desarrollando este equipo de I+D y deberán adaptarse a esta nueva forma de trabajar.

Con el despliegue de nuestro sistema, estos equipos terceros seguirán utilizando las soluciones IA, pero en forma de servicios de tarea registrados en la Aplicación Web de Gestión. Esta aplicación web proporcionará un acceso sin permiso de edición de servicios para este tipo de usuarios, que podrán acceder al inventario de servicios para consultar la interfaz del servicio que necesitan.

Este es el valor real de la plataforma, ofrecer la funcionalidad para crear composiciones de servicios que sean utilizadas de manera estandarizada por equipos terceros, que serán, como hemos dicho, usuarios únicamente consumidores de servicios. Es por esto por lo que es importante trabajar aspectos de visibilidad y mejorar el descubrimiento de servicios.

El rendimiento de ejecución de los algoritmos IA implementados ahora en forma de microservicios no se verá afectado, pues las tecnologías y protocolos utilizados son eficientes, seguros y ligeros.

Las máquinas Azure que hospeden los contenedores serán configurables en tiempo de ejecución para ampliar el hardware en función de las necesidades del Host de Microservicios, de manera que frente a cargas de trabajo intensas el cliente no experimentará ningún tipo de irregularidad.

A esto se le suma la implementación del sistema de cacheado de microservicios con el que las llamadas similares a servicios se volverán instantáneas, aligerando la carga de procesamiento de las máquinas Azure.

Esta plataforma brindará agilidad tanto al desarrollo de soluciones IA en forma de servicios como al consumo de estas.

Este trabajo, como detallaremos en la siguiente sección, no acaba con este primer despliegue. Por supuesto, el equipo de desarrollo seguirá trabajando en conjunto con el equipo de I+D para mejorar la plataforma y satisfacer las necesidades de todos los usuarios de la plataforma ofreciendo soporte para los problemas que surjan y actualizaciones para futuras mejoras.

En la siguiente sección hablaremos también de las diferentes opciones que se presentan para este proyecto, tanto dentro de la empresa en la que se ha creado como fuera de esta.

4.3. Líneas futuras

Comenzaremos este apartado mencionando de nuevo el trabajo de Mamdouh Ibrahim, Brenda Michelson, Kerri Holley, Dave Thomas, Nicolai M. Josuttis y John de Vadoss [4]. En este trabajo realizan diferentes entrevistas a importantes personalidades dentro del mundo de la informática para conocer su punto de vista sobre la trayectoria de SOA y la dirección hacia la que apunta.

Recalcan con buen criterio que SOA es una evolución que requiere un cambio de paradigma, para el cual es necesario afrontar multitud de retos tanto técnicos como de concepción de los sistemas orientados a servicios, como la diferencia entre SOA y los servicios web, y los posibles niveles de adopción de SOA por parte de las empresas. Otros conceptos clave son los de la granularidad, reutilización y gobernanza de los servicios que deben ser interiorizados por los usuarios que implementen SOA en sus sistemas para aprovechar las ventajas de esta arquitectura.

SOA no introduce ningún concepto revolucionario completamente nuevo, si no que consiste en unir diferentes conceptos y prácticas bajo unas condiciones concretas para crear sistemas distribuidos mantenibles y robustos, que presenten componentes y servicios **heterogéneos**, reutilizables y estandarizados. El concepto de heterogeneidad en la implementación de los componentes de un sistema distribuido es clave para la consecución de los objetivos de una arquitectura SOA.

Hablando de aspectos más técnicos, advierten que obviamente hay un intercambio entre rendimiento y reusabilidad, y que las tareas de testeo y monitorización pueden complicarse si no se diseña una infraestructura que haga énfasis en facilitar este tipo de procesos.

Mencionaremos también el trabajo de Geetha Presena Kumari, Balaji Kandan y Asish Kumar Mishra [7] sobre integraciones en sistemas SOA ya que su definición del principal objetivo de SOA es muy acertada y ha marcado el desarrollo de nuestro proyecto. Resumen que el principal objetivo de SOA es exponer la funcionalidad de diferentes aplicaciones de manera estandarizada para poder ser utilizada en otros proyectos. De esta manera se reduce enormemente el esfuerzo y el tiempo para mantener y escalar los sistemas IT para que cumplan con las necesidades de negocio.

Plan de acción

A continuación, hablaremos de la ejecución del plan de mejora para corregir los defectos comentados en el apartado 4.1, y de las posibles mejoras y actualizaciones adicionales para añadir funcionalidad a la plataforma.

El protocolo de comunicación HTTP ha servido como soporte para la creación de flujos de información en las aplicaciones desde el nacimiento de la web. Sin embargo, rápidamente quedó vulnerado ya que la información no viaja encriptada, dando lugar a multitud de ataques informáticos. Este aspecto de nuestro sistema será el primero en sufrir una actualización para comenzar cuanto antes a utilizar comunicaciones cifradas con certificados, ya que, aunque la plataforma trabaje dentro del entorno empresarial el uso de HTTPS es básico para cualquier tipo de implementación web.

Se preparará una serie de certificados para comenzar las pruebas de implementación de HTTPS mientras se lleva a cabo la implementación de la infraestructura necesaria para trabajar con HTTPS en las diferentes interfaces de comunicación, finalizando con la implementación completa de HTTPS.

Seguidamente se dedicará un tiempo al estudio de la implementación actual de JWT, que, a pesar de cumplir con la implementación y estándares básicos de esta tecnología, existen vulnerabilidades para las cuales se deben tomar medidas. Una vez detectadas las posibles vulnerabilidades se procederá a su corrección y documentación.

Por último, hablaremos de la implementación diseñada para la “dockerización” en Azure. El equipo desarrollador de este proyecto no cuenta con experiencia previa en este tipo de despliegues, por lo que previo al despliegue se realizarán una serie de formaciones orientadas al despliegue de aplicaciones web tanto Node como Python con contenedores Docker en máquinas de Azure.

Para su aplicación, se seguirá un procedimiento estándar de despliegue de contenedores en esta nube. A continuación, se realizará un estudio de las diferentes configuraciones para las máquinas de Azure para formar al equipo en este tipo de despliegues. Finalmente se aplicarán las medidas necesarias para crear un sistema de contenedores Docker desplegado en máquinas de Azure seguro y robusto, priorizando la seguridad, pero teniendo siempre en mente la flexibilidad en cuanto a potencia de procesamiento necesaria para llevar a cabo la actividad relacionada con los microservicios de inteligencia artificial.

Durante este periodo de mejora y formación, se continuará trabajando en conjunto con el equipo de I+D para mantener la plataforma actualizada y asegurar que cumple con las necesidades y requisitos de este equipo como ha venido haciendo hasta ahora.

Futuro del proyecto dentro y fuera de la empresa

Finalmente hablaremos del futuro del proyecto tanto dentro como fuera de la empresa, de los posibles escenarios en los que nos vemos en el futuro y de la motivación para seguir trabajando en el proyecto en el futuro.

En primer lugar, comentaremos que, habiendo cumplido con los objetivos transversales y de implementación, y contando con el plan de acción y mejora, el equipo de desarrollo ha quedado altamente satisfecho con el resultado.

Dentro de la empresa

El primer posible escenario en el que vemos este proyecto es cumpliendo con el objetivo primero del mismo, sirviendo como plataforma SOA de microservicios de inteligencia artificial en la empresa para la que fue pensado.

En este caso, el equipo desarrollador llevará a cabo las diferentes tareas recogidas en el plan de mejora. Al mismo tiempo, el equipo desarrollador trabajará con el equipo de I+D para continuar con el proceso de actualización y mejora de la plataforma, al mismo tiempo que se comienza el proceso de apertura de la plataforma para los diferentes departamentos de la organización, pues serán los usuarios finales del sistema.

La naturaleza de estos equipos de la organización es realmente diversa en cuanto a metodología de trabajo y tecnologías que utilizan en sus soluciones. Esto supondrá un reto pues al formar parte del sistema, aumentará la carga de trabajo de soporte al aumentar el tráfico en el sistema. Al igual que con el equipo de I+D, estos equipos presentarán requisitos y necesidades que se deberán tener en cuenta para las actualizaciones de la plataforma SOA.

El coste de las máquinas Azure en las que se llevará a cabo el despliegue corre por supuesto a cuenta de la empresa como un gasto más en infraestructura para la organización.

En este proceso de expansión en el interior de la empresa hará brillar la elección de los diferentes estándares y tecnologías utilizados en el diseño e implementación de la infraestructura. Asimismo, y junto con los procesos de evaluación y monitorización llevados a cabo, se probará realmente el desempeño del sistema.

Fuera de la empresa

Finalizaremos esta memoria explicando el posible escenario en el que el equipo desarrollador de la plataforma termina el desarrollo acordado con esta empresa y continua su actividad fuera de la misma.

La plataforma construida es un sistema funcional preparado para un entorno empresarial cerrado y controlado, que necesitaría pasar por las fases de evolución y mejora propuestos en el plan de acción. Esta primera fase sería similar al proceso explicado en el apartado anterior: el sistema deberá evolucionar principalmente para aportar más seguridad, además de continuar añadiendo funcionalidad a la plataforma.

Una vez conseguido un sistema SOA preparado para un entorno abierto y accesible por cualquier tipo de usuario, el siguiente paso sería el diseño de diferentes planes de acceso y monetización para gestionar el uso del sistema y sacar rentabilidad a la plataforma. El objetivo sería crear, sobre el mecanismo de acceso actual, un sistema de autorización y autenticación seguro y robusto para gestionar el acceso a la plataforma desde el exterior, teniendo en cuenta aspectos de escalabilidad y rendimiento.

Primeramente, se prepararía un plan para empresas y organizaciones con el que acceder a la plataforma SOA de microservicios ya implementada y poblada con diferentes microservicios IA. En este sistema sería necesaria una batería inicial de funcionalidad en forma de servicios y microservicios, así un mecanismo para gestionar las peticiones de los clientes y actualizar el inventario con nuevos servicios de tarea y microservicios.

La manera de monetizar la plataforma se podría llevar a cabo de dos maneras: mediante un sistema de alquiler o mediante pago por consumo. Este plan para organizaciones vendría acompañado de un plan para usuarios más pequeños o individuales en el que el precio y potencia de las máquinas vendría adaptado a las necesidades de estos usuarios.

Finalmente sería necesario un plan de prueba gratuito, limitado en funcionalidad o en tiempo de uso, para dar a conocer la funcionalidad de la plataforma, ofreciendo una versión de la plataforma limitada pero suficiente para vender el producto.

Otra opción completamente diferente a los planes de suscripción o alquiler comentados anteriormente es la preparación de la plataforma para su venta o alquiler como infraestructura software, como código. La forma de monetizar esta opción se llevaría a cabo mediante la venta de licencias de uso temporales, que deberían negociarse y adaptarse a cada cliente concreto. Estas licencias deberán renegociarse cada vez que finaliza el contrato para adaptar el precio al estado actual de la plataforma, ya que se mantendrá un flujo constante de actualizaciones y mejoras.

Referencias

[1] Y. C. Zhou, X. P. Liu, X. N. Wang, L. Xue, C. Tian, y X. X. Liang, «Context model based SOA policy framework», en *ICWS 2010 - 2010 IEEE 8th International Conference on Web Services*, 2010, pp. 608-615. doi: 10.1109/ICWS.2010.115.

[2] L. O'Brien, P. Brebner, y J. Gray, «Business transformation to SOA: Aspects of the migration and performance and QoS issues», en *Proceedings - International Conference on Software Engineering*, 2008, pp. 35-40. doi: 10.1145/1370916.1370925.

[3] S. Kuppuraju y A. Kumar, «Enabling SOA governance for production deployed services», en *Proceedings - 2008 IEEE Congress on Services, SERVICES 2008*, 2008, vol. PART 1, pp. 109-110. doi: 10.1109/SERVICES-1.2008.37.

[4] M. Ibrahim, K. Holley, N. Josuttis, B. Michelson, D. Thomas, y J. deVadoss, *The future of SOA: what worked, what didn't, and where is it going from here?* 2007. doi: 10.1145/1297846.1297975.

[5] S. Kuppuraju y A. Kumar, «Enabling SOA governance for production deployed services», en *Proceedings - 2008 IEEE Congress on Services, SERVICES 2008*, 2008, vol. PART 1, pp. 109-110. doi: 10.1109/SERVICES-1.2008.37.

[6] A. Hsiung, G. Rivelli, y G. Huttenegger, «How to design a global SOA infrastructure: Coping with challenges in a global context», en *Proceedings - 2012 IEEE 19th International Conference on Web Services, ICWS 2012*, 2012, pp. 536-543. doi: 10.1109/ICWS.2012.22.

[7] G. P. Kumari, B. Kandan, y A. K. Mishra, «Experience sharing on SOA based heterogeneous systems integration», en *Proceedings - 2008 IEEE Congress on Services, SERVICES 2008*, 2008, vol. PART 1, pp. 107-108. doi: 10.1109/SERVICES-1.2008.56.