Máster Universitario de Investigación en Ingeniería de Software y Sistemas de Información

Código (31105151) Trabajo Fin de Máster Universitario en Investigación de Software y Sistemas de Información

Integración y evolución hacia una arquitectura orientada a Microservicios dirigida por eventos en la difusión de Información en los mercados financieros

Estudiante: Mario Antonio Vergés Rodríguez

Directora de tesis: Elena Ruiz Larrocha

Curso académico: 2020-2021. Convocatoria de Junio

Máster Universitario de Investigación en Ingeniería de Software y Sistemas de Información

Código (31105151). Asignatura: Trabajo de Fin de Máster en Ingeniería de Software y Sistemas Informáticos

Título del trabajo:

Integración y evolución hacia una arquitectura orientada a Microservicios dirigida por eventos en la difusión de Información en los mercados financieros.

Tipo de trabajo al que pertenece:

Tipo de trabajo específico propuesto por el alumno.

Nombre del estudiante:

Mario Antonio Vergés Rodríguez

Nombre de la tutora o directora:

Elena Ruiz Larrocha

DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO CIENTÍFICO, PARA LA DEFENSA DEL TRABAJO FIN DE MASTER

Fecha: 15/03/2021

Quién suscribe:

Autor(a): Mario Antonio Vergés Rodríguez
D.N.I/N.I.E/Pasaporte.: 21497939T

Hace constar que es la autor(a) del trabajo:

Título completo del trabajo.
Integración y evolución hacia una arquitectura orientada a microservicios dirigido por eventos en la difusión de información en los mercados financieros.

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores, se han referenciado debidamente en el texto de dicho trabajo.

DECLARACIÓN:

- ✓ Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.
- ✓ Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- ✓ No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Fdo.

IMPRESO TFDM05_AUTORPBL
AUTORIZACIÓN DE PUBLICACIÓN
CON FINES ACADÉMICOS

UNED ETS de Ingeniería Informática

Impreso TFdM05_AutorPbl. Autorización de publicación y difusión del TFM para fines académicos

Autorización

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los Autor/es

Juan del Rosal, 16
28040, Madrid

Tel: 91 398 89 10
Fax: 91 398 89 09
www.issi.uned.es

Resumen

La mayoría de las organizaciones trabajan solo en unos pocos dominios. Construyen repetidamente sistemas similares dentro de un dominio determinado con variaciones para satisfacer las diferentes necesidades de los clientes. Se pueden lograr ahorros significativos reutilizando partes de sistemas anteriores en el dominio al construir nuevos.

La ***arquitectura de software*** es el resultado de un esfuerzo importante y su desarrollo puede representar una parte considerable del trabajo que se realiza en un proyecto de desarrollo.

Las ***líneas de productos de software*** buscan justamente promover ***la reutilización sistemática de artefactos*** de los cuales la arquitectura es uno de los más importantes. Este enfoque busca tener distintos beneficios asociados a la reutilización como pueden ser la reducción del tiempo de desarrollo (pues ya no se tienen que desarrollar ciertas partes del sistema), y la mejora de la calidad (pues se incorporan partes que ya han sido verificadas previamente).

En objetivo de este trabajo de ***Ingeniería de línea de productos software***[1], es analizar el **dominio**[2] **‘***del problema’* de la **difusión de información financiera en el mercado bursátil**, con productos de interés donde se determina qué características son comunes a todos los productos y, por lo tanto, pueden estar ***bajo una misma arquitectura*** de una manera global.

Es por ello por lo que se aborda una fase inicial de análisis y posterior propuesta **arquitectural de software**, donde por otro lado se busca hacer una reutilización intensiva. Una idea clave en la **reutilización sistemática de software** es la de dominio.

(1) La ***Ingeniería de dominio***, también llamada ***ingeniería de línea de productos*** es todo el proceso de reutilización (conocimiento del dominio) en la producción de nuevos sistemas de software.

(2) **El *análisis de dominio*** es el proceso de identificar dominios, delimitarlos y descubrir similitudes y variabilidades entre los sistemas en el dominio. Esta información se captura en modelos que se utilizan en la fase de implementación del dominio para crear artefactos como componentes reutilizables, un idioma específico del dominio, o generadores de aplicaciones que se pueden utilizar para construir nuevos sistemas en el dominio.

Abstract

Most organizations work on only a few domains. They repeatedly build similar systems within a given domain with variations to meet different customer needs. Significant savings can be achieved by reusing parts of previous systems in the domain when building new ones.

Software architecture is the result of a significant effort and its development can represent a considerable part of the work that goes into a development project.

The **software product lines** precisely seek to promote the systematic reuse of artifacts of which architecture is one of the most important. This approach seeks to have different benefits associated with reuse, such as reduction of development time (since certain parts of the system no longer have to be developed), and quality improvement (since parts that have already been previously verified are incorporated.).

The objective of this **software product line engineering work**[1] is to analyze the **domain**[2] of the problem of the **diffusion of financial information in the stock market**, with products of interest where it is determined which characteristics are common to all products, and therefore , they can be ***under the same architecture*** in a global way.

That is why an analysis and design phase is approached where a common software architecture is proposed where intensive reuse is made. A key idea in systematic software reuse is mastery.

(1) **Domain Engineering**, also called **Product Line Engineering**, is the entire process of reuse (domain knowledge) in the production of new software systems.

(2) **Domain analysis** the process of identifying domains, delimiting them, and discovering similarities and variabilities between systems in the domain. This information is captured in models that are used in the domain deployment phase to create artifacts such as reusable components, a domain-specific language, or application builders that can be used to build new systems in the domain.

Keywords: Microservicios, Arquitecturas Orientadas a Servicios, Arquitecturas monolíticas, EDA, SOA, MQTT, Arquitectura de software, Ingeniería de línea de productos, mercados financieros, Reutilización, ETL, Mejora de procesos.

Contenido

Lista de figuras

Capítulo 1: Introducción.

Este documento contiene la memoria de la **Tesis Final del Máster de Investigación en Ingeniería de Software y Sistemas de Información**.

Este trabajo está relacionado directamente con las **temáticas** de ***mejora del desarrollo a través de la arquitectura del software***, ***gestión y mejora de procesos software***, ***desarrollo de líneas de producto software*** y ***arquitecturas orientadas a servicios***.

El trabajo consiste en el análisis del ***dominio del problema*** relativo a la **difusión de información financiera en el mercado bursátil**, con productos de interés donde se determina qué características son comunes a todos los productos; donde posteriormente se propone una arquitectura que integre dichos productos y permita iniciar el camino hacia la ***reutilización sistémica intensiva*** del software.

El análisis del dominio del problema se enmarca en el ámbito de negocio de la filial **Market Data** del **Grupo SIX-BME**, especializada ésta en la generación de flujos de información primaria en tiempo real, fin de día e históricos de información financiera del mercado español, donde desarrollo una parte de mis funciones en Ingeniería de Software, en el grupo de trabajo de **Reporting Tools (CIT-MER)**, ubicado en **Markets IT ES (CIT-ME) del Grupo SIX-BME.**

En la actualidad existen un conjunto de aplicativos de arquitectura monolítica de n capas que están construidos como silos aislados, codificados por diferentes grupos de trabajo de manera estanca y aislada, con diferente tecnología en algunos casos y sin la adopción de criterios unificados de diseño ni análisis previo del conjunto.

Cada producto necesita un tratamiento específico, ya que los procesos de recepción y procesamiento, enriquecimiento de la información -en algunos casos- y envío de ficheros tienen características específicas para cada uno de ellos.

Son procesos considerados críticos, puesto que forman parte de la cadena de procesos de los clientes, por lo que cuentan con un plan de contingencia bien definido; es decir, en caso de caída del servicio, se cuenta con un sistema alternativo de ejecución, métodos de comunicación a clientes y a TI de no disponibilidad del servicio, y un plan detallado de recuperación del servicio.

Se ha estimado ya un número elevado de ellos que en su mayoría son servicios maduros. Se han detectado también otros proyectos en marcha, los cuales ya están prestando servicio también.

Compartiendo todo el aspecto común de la difusión de información financiera bursátil, a continuación, se detallan los aspectos de la variabilidad analizados. Se ha estimado hasta tres subconjuntos categorizados acorde a las características propias de arquitectura, grado de especialización, recogida de información, procesamiento de esta, salida de información y tamaño, que podrían estimarse en tres grupos:

Un **primer grupo** lo compondrían pequeños servicios-productos muy especializados.

Un **segundo grupo** lo compondrían productos que contienen mayor complejidad, tanto técnica como de negocio.

Para este primer y segundo grupo de procesos, previamente se han realizado las funciones de transformación de la información en otras áreas de Tecnología; es por lo que únicamente la recogen y la tratan según algunas lógicas de negocio específicas para difundirlas a través de medios securizados a buzones de clientes finales.

En estos dos primeros grupos los equipos de desarrollo habitualmente son formados por un Ingeniero de software responsable del análisis, diseño, desarrollo y mantenimiento del servicio. La formación de los ficheros a transferir y el proceso que lo involucra es una caja negra en el sentido más literal del término. Finalmente, estos se ejecutan aisladamente en distintas mesas de operaciones.

Existe un **tercer grupo** que cumple casi totalmente la definición de un **proceso ETL (ver en Anexo, sección 14.2);** donde se trata el ciclo completo de *extracción, transformación y carga* en una solución de uso intensivo de tecnología avanzada, en el que se ha llevado a cabo como un proyecto tecnológico que ha contado con equipo multidisciplinar formado por un equipo completo de negocio, y un equipo completo técnico formado por analistas/desarrolladores, un arquitecto de software y un ingeniero de calidad, donde se siguen metodologías ágiles de desarrollo, entre otras características de ciclo de vida de software avanzadas.

En secciones posteriores se hace una descripción detallada del modelo funcional de este proyecto, que se usará como base para evolucionar la arquitectura, el modelo de gobernanza en IT e introducir los conceptos clave de ***Ingeniería de dominio***.

Se podrían enumerar una combinación de factores que pudieran dar explicación a esta situación de dispersión en este tipo de aplicativos. El que se hayan desarrollado para distintas áreas de negocio, que la compañía está inmersa desde hace unos años en un profundo proceso de transformación digital y madurez en la que el desarrollo de este ha podido producir algunos desequilibrios y que al ser procesos críticos también se ha buscado en algunos departamentos tener la menor complejidad técnica posible, apuntan a ser las causas más probables.

1.2 Estructura del documento

La estructura de este ***trabajo de especialización e iniciación a la investigación*** es la siguiente:

- El **capítulo 2**, en sus cinco secciones, ofrece el ***estado del arte*** actual de los distintos estilos arquitecturales hasta la fecha en uso o tendencia. Así se describe desde la arquitectura monolítica, pasando por SOA y Microservicios hasta la EDA.
- Los **cuatro capítulos posteriores** describen el **modelo funcional de la solución ETL**, el **servicio web**, el **proceso de flujo** y **estados de diagramas** de alto nivel.
    - El ***capítulo 3,*** ofrece la descripción funcional del sistema.
    - El ***capítulo 4,*** detalla las operaciones del servicio web que conectará en un sentido y otro con el portal web de la solución.
    - Los ***capítulos 5 y 6,*** ofrecen los flujos de procesos y los diagramas de estado de alto nivel.
- **Cinco capítulos** dedicados a la **fase de diseño**:
    - ***Capítulo 7***. Enfoques de diseño clave.
    - ***Capítulo 8***. Detalles del estilo arquitectural.
    - ***Capítulo 9***. Detalle del modelo de Arquitectura Técnica.
    - ***Capítulo 10***. Patrones de diseño claves en la propuesta.
- Un capítulo dedicado a la **gobernanza IT**.
    - ***Capítulo 11***. La descentralización de gobierno.
- El ***capítulo 12***, recoge las ***conclusiones de este trabajo***.
- El **capítulo 13**, proporciona algunas posibles líneas de trabajo futuras.

- Tras la bibliografía se utiliza un anexo con algunas definiciones técnicas especializadas.

1.3 Objetivo y Alcance

El **objetivo y alcance** de este trabajo es ***valorar el sistema actual en su conjunto*** y dotarlo de un ***modelo de arquitectura que mejore el actual***, ***en términos de eficiencia técnica, reutilización sistémica, alineación de TI con negocio y de mejora en la gestión de procesos***.

En concreto, se propone un **modelo de arquitectura dirigida por eventos que complemente la arquitectura orientada a servicios** a presentar (Microservicios en esta ocasión) para que los servicios puedan ser activados por disparadores que se encuentran en eventos entrantes.

A continuación, se enumeran los objetivos:

1. Presentar y justificar el modelo de arquitectura de microservicios dirigida por eventos; y describir la mejora con respecto al actual modelo y cómo lo hace.

2. Demostrar cómo dicho modelo mejora el objetivo de alineación estratégica de TI con la de negocio.

3. Realizar una breve exposición de mejoras en el modelo de administración de procesos adaptado al software que se presenta, demostrando así los mínimos ajustes gerenciales necesarios al modelo técnico.

1.4 Servicios y grado de madurez en TI

El término **servicio** en **ITIL** que utiliza frecuentemente se define como "***un medio para entregar valor a los clientes facilitándoles resultados que estos desean lograr sin la propiedad de costos y riesgos específicos".***

Cada una de estas aplicaciones aisladas atendería a esa definición, puesto que cada una aporta un valor, facilita resultados, y no asumen riesgos específicos ni propiedad de costos,

y en general parece que tienen el suficiente grueso como para considerarlos servicios en términos ITIL.

Por lo tanto, podríamos concluir que todo es correcto y que pasaríamos con nota la **gestión ITIL** que hace la empresa de los servicios.

Tampoco se plantea analizar si la empresa ha adquirido el grado de madurez x de un modelo **CMMI**, o similar, porque esto se da por sentado; y aunque todo es mejorable y los niveles de madurez son distintos entre equipos, la empresa se encuentra en un alto nivel de madurez en prácticamente todas sus filas de TI.

Parece obvio que el camino no es cuestionar pues, si la empresa ha adoptado estándares de gestión de TI idóneos o no, como ITIL, ni siquiera si la adopción de ciertos servicios responde de manera exacta a la definición que estos hacen de servicio.

No se podría afirmar categóricamente que NO se responde a la necesidad de alinear las necesidades de negocio con la estrategia IT en un sentido más ITIL de gestión de los servicios.

Además, éste sería un debate excesivo por el resultado. El calado de los cambios que tendrían que producirse, con unos procesos que no están clasificados en función del servicio de negocio que explotan, sino más bien, en función de su propia naturaleza de ejecución, a bote pronto parece indicar una empresa de difícil consecución.

Estaríamos hablando de una completa integración de estos procesos, para conformar un servicio general que albergue todas las formas de recepción y envío de información a los clientes finales. Ello implicaría, sin duda, la presentación de la idea al comité de dirección, para que esta acepte este cambio, donde numerosas partes implicadas se verían afectadas.

Llevar a cabo esta tarea en una empresa con cientos de temas urgentes encima de la mesa y con un tema que pone el foco directamente a la gestión, apunta a que estaría condenado al cajón de los olvidos.

Por lo tanto, desde la perspectiva de la gestión de los servicios, el debate de que ‘esos servicios’ no son tal, se antoja una batalla complicada que, en la mayoría de los casos, parece que no convencería para motivar un cambio que se supone importante en cuanto a movilización de recursos.

1.4 Administración de procesos

Por un lado, los profesionales de TI solemos ver las cosas como 'cajas'; por ejemplo, para el caso que estamos exponiendo, una persona administra los *sistemas*, otra administra los servicios de transferencia, otra los almacenes de datos, otra administra el software de ejecución, e incluso alguna otra persona se responsabiliza del servicio de esa aplicación en concreto, otra de la mesa de operaciones, etc.... Por otra parte, los usuarios lo que quieren es hacer uso de ciertos servicios de TI.

Es decir, mientras que los técnicos se enfocan en "administrar cajas", los usuarios lo que ven son servicios, y la empresa desea alinear sus estrategias, lo que nos lleva a preguntarnos con talante crítico si toda esa conjunción es armónica, es decir, se cumple.

Precisamente en esa interconexión radica el asunto; en que un servicio de TI es entregado a los usuarios mediante un conjunto de cajas (tecnología), personas que manejan las cajas (gente) y relaciones entre ellos (procesos). Es por ello por lo que en el área de TI debemos entender que lo que se administra y se da a los usuarios son servicios de TI, y **no únicamente dispositivos**.

Las organizaciones se estructuran en base a departamentos funcionales que dificultan la orientación hacia el cliente (por ejemplo, Finanzas, Producción, Ventas y TI), con organigramas muy jerárquicos divididos en múltiples niveles.

Normalmente este tipo de organizaciones adolece de dos grandes problemas:

La comunicación "oficial" fluye verticalmente y puede ser muy lenta. Incluso a veces propicia el conocido juego del "teléfono descompuesto". Esto es, dos operarios de áreas distintas que requieren de un acuerdo oficial deben "subir" por el organigrama hasta la rama común, pasando por todos los niveles de este (operador A –> supervisor de A–> coordinador de A –> gerente de A -> director de A -> director de B -> gerente de B -> coordinador de B -> supervisor de B -> operador B).

Se crean "cotos de poder" e "islas de información estancas" que nada tienen que ver con la satisfacción de las necesidades del cliente final. Estas y otras causas, derivadas de la cultura empresarial en la que trabajamos, son las razones por las que hay **procesos de idénticas características en distintos departamentos** sin comunicación horizontal ni conocimiento compartido.

Por contra, la organización está atravesada por **procesos** que **fluyen horizontalmente**, que fluyen por los organigramas jerárquicos.

Es por lo anterior, por lo que surge la ***administración por procesos***, que percibe la organización como un sistema interrelacionado de procesos que contribuyen conjuntamente a incrementar la satisfacción del cliente. La administración por procesos coexiste con la administración funcional, asignando «propietarios» a los procesos clave, haciendo posible una gestión interfuncional generadora de valor para el cliente y que, por tanto, procura su satisfacción; determina qué procesos necesitan ser mejorados o rediseñados, establece prioridades y provee de un contexto para iniciar y mantener planes de mejora que permitan alcanzar objetivos; y hace posible la comprensión del modo en que están configurados los procesos de negocio, de sus fortalezas y de sus debilidades.

Así mismo da la forma en que la organización administra y mejora continuamente los procesos de negocio para lograr sus objetivos y crear valor para sus accionistas, clientes y colaboradores. Al final, lo que la administración de procesos intenta cambiar es: ir de una organización orientada a productos (en la que existen procesos no coordinados ni administrados), por una organización que administra sus procesos en ciclos de mejora continua mediante el famoso ciclo de *"plan-do-check-act"* o PDCA, por sus siglas en inglés.

La clave es entonces lograr la integración eficiente de ***gente, procesos y tecnología*** para una mejor administración de los servicios de TI, optimizando el uso de los recursos y mejorando constantemente los niveles de servicio.

Se trataría entonces de mejorar la administración de procesos aplicado desde la óptica técnica, presentando un diseño que mejore la administración de procesos. Es decir, partiendo del actual diseño, se trata de rediseñar el modelo de manera que NO impacte en la organización actual de explotación de los mismos y que, por otro lado, en caso de algún ajuste en la administración de procesos, este permita ese cambio sin sufrir el modelo por ello.

Para ello se tendrán algunas líneas maestras de diseño como, por ejemplo, adaptativo al cambio, seguro, robusto, confiable y en términos de calidad.

Capítulo 2: Estado del arte

La **arquitectura de software** describe un sistema de software en términos de componentes principales, las relaciones existentes entre ellos y la información que se transfiere entre los componentes. Es decir, es un plan para construir sistemas con unos requisitos específicos, y sistemas que poseen las características necesarias para cumplir dichos requisitos.

No existe una definición única para arquitectura de software. En www.sei.cmu.edu/architectura/definitions.html [4] se incluyen algunas de ellas, por citar alguna fuente entre las muchas definiciones que hay.

La **arquitectura de software** no solo se preocupa por la estructura y el comportamiento, sino también por el uso, la funcionalidad, el rendimiento, la resiliencia, la **reutilización**, la comprensibilidad, las limitaciones y **compensaciones económicas y tecnológicas**, y la estética. [3]

Sin embargo, dos empresas podrían tener aplicaciones similares para la solución de un mismo dominio de aplicación, por lo que esto implicaría que existiría una diferencia entre la arquitectura específica y el tipo de solución definida. Esto último se define **estilo arquitectónico**.

En una solución empresarial la elección de un estilo arquitectónico generalmente se realiza como resultado de compensaciones de ingeniería en respuesta a un conjunto específico de requisitos.

Por ejemplo, el estilo arquitectónico de *n* niveles, profusamente implementado en nuestra industria hasta hoy, está diseñado para cumplir con los requisitos de distribución, escalabilidad, flexibilidad de la interfaz, independencia del dispositivo, reutilización de servicios comerciales, integración de aplicaciones, etc.

La **arquitectura *n* -tier** está diseñada para proporcionar acceso de cliente basado en web o no, que interactúa con las capas de negocios (servicios de negocio) de la empresa, y que incluso podrían venir de una migración de una aplicación heredada.

Algunos principios y prácticas arquitectónicas importantes son:

- Separación de preocupaciones.
- Vistas arquitectónicas.
- Acomodación del cambio.
- Abstracción.
- Consistencia.
- Derivación empresarial.
- Patrones.
- Facilitación.
- Comunicaciones.

La **separación de preocupaciones**. Este es el principio más fundamental de la arquitectura. En teoría de orientación de objetos sería una clase, una responsabilidad. El beneficio es que un cambio en una parte del sistema no afecta negativamente a otras partes. Un ejemplo familiar de este principio es la separación de la interfaz de la implementación.

Las **vistas arquitectónicas** proporcionan otra separación importante de preocupaciones mediante la inclusión o exclusión de detalles específicos y la presentación de información a diferentes partes interesadas. Las vistas o perspectivas arquitectónicas están diseñadas para abordar preocupaciones específicas del desarrollo de software o los grupos y organizaciones empresariales importantes que juegan un papel en el ciclo de vida completo de las soluciones empresariales. Las vistas típicas de software son lógica, implementación, proceso y red. Las vistas (preocupaciones) típicas de la arquitectura empresarial son negocios, información, aplicaciones, tecnología e implementación. Casualmente, la implementación y el diseño de SOA se adaptan bien a estos conjuntos de preocupaciones arquitectónicas.

La **acomodación del cambio**. Este es un aspecto realmente importante que se liga con aspectos de **gobernanza de IT**. Aplicaciones diseñadas para el cambio, se refiere a que la arquitectura debe proporcionar flexibilidad, de modo que los requisitos de las aplicaciones futuras puedan satisfacerse más fácilmente. Una arquitectura flexible identifica tanto los requisitos de las aplicaciones futuras como las áreas que probablemente cambiarán. El seguimiento de las tendencias de la industria ayuda a identificar algunas áreas de cambio

potencial. Estas áreas deben abordarse explícitamente en arquitectura. Si los detalles de flexibilidad e independencia no se incluyen en el diseño original, es muy probable que la arquitectura contenga acoplamientos implícitos, que son mucho más difíciles de manejar cuando ocurre un cambio inevitable.

La **abstracción** es una herramienta arquitectónica clave que se utiliza para desacoplar, adaptar el cambio y separar las preocupaciones. Hay un dicho en la industria que dice que "cualquier problema en ciencias de la computación se puede resolver agregando una capa de abstracción". Una capa de abstracción proporciona una indirecta entre dos capas, lo que permite una mayor flexibilidad. Normalmente, la abstracción también proporciona un mayor nivel de interacción. Por ejemplo, en lugar de escribir directamente en una base de datos, escribe SQL, que proporciona un modelo de interacción de nivel superior, así como una capa de abstracción e indirección por encima de las interfaces de base de datos de nivel inferior. La abstracción proporciona una mayor productividad (en el sentido de que una instrucción SQL corresponde a muchas invocaciones de las interfaces de base de datos de nivel inferior) y también admite varios almacenes de datos diferentes.

Otro de los principales objetivos de la arquitectura es la **promoción de la coherencia** y la **reutilización**.

La **derivación empresarial** es quizás el principio arquitectónico más importante, que reconoce que la razón de ser de una arquitectura (y la de la propia TI) es respaldar el negocio de la empresa, es decir, las estrategias y objetivos de la organización. ***Cadena de valor de Michael Porter*** **[5].**

Un **patrón** es una plantilla para una solución a un conjunto específico de requisitos y, como tal, es una herramienta para describir la arquitectura. Generalmente se reconoce que el padre del movimiento de patrones es Christopher Alexander, profesor de arquitectura (como en edificios, no en software) en Berkeley. Lo expresó de esta manera: “Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, y luego describe el núcleo de la solución a ese problema, de tal manera que puede usar la misma solución un millón de veces, sin haciéndolo dos veces de la misma manera".

Facilitación significa que una arquitectura debe facilitar la creación de soluciones que se ajusten a la arquitectura. Por tanto, la arquitectura no se trata solo de describir lo que hace

el sistema; también debe proporcionar los medios para construir el sistema y sus componentes.

Las **comunicaciones** tienen que ver con el hecho de que una arquitectura proporciona el mecanismo para que las personas se comuniquen y alcancen un entendimiento común de los sistemas de TI. Estos principios arquitectónicos le permiten describir la arquitectura de manera inequívoca, en diferentes niveles de abstracción y en un contexto comprensible para cada parte interesada.

2.1 Arquitectura monolítica

El **estilo arquitectónico monolítico** es usado ampliamente en la industria, y básicamente es una pieza de software independiente; un silo, cuya funcionalidad no depende de otras piezas.

Tradicionalmente, su división en componentes puede describirse según la siguiente imagen:

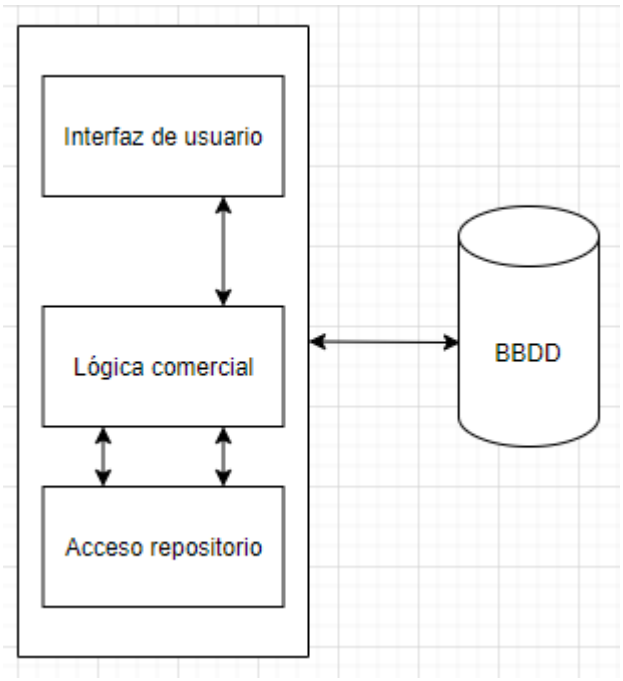

Figura 1. Representación gráfica del estilo monolítico.

El estilo monolítico: **una aplicación monolítica construida como una sola unidad**. Las aplicaciones empresariales se componen de tres partes principales: una **interfaz de usuario** del lado del cliente (que consta de páginas HTML y javascript que se ejecutan en un navegador en la máquina del usuario) una **base de datos** (que consta de muchas tablas insertadas en una gestión de base de datos común y generalmente relacionales) y una **aplicación del lado del servidor.**

La aplicación del lado del servidor maneja las solicitudes HTTP, se encarga de la lógica del dominio, recupera y actualiza los datos de la base de datos y selecciona y completa las vistas HTML para enviarlas al navegador. Esta aplicación del lado del servidor es un *monolito*: un

único ejecutable lógico. Cualquier cambio en el sistema implica la creación e implementación de una nueva versión de la aplicación del lado del servidor.

Este estilo arquitectural se emplea masivamente en la industria. Toda la lógica se ejecuta en un solo proceso, lo que permite usar las características básicas de un lenguaje de dividir el dominio de aplicación en clases, funciones y espacios de nombres. La escalabilidad del sistema reside en el crecimiento horizontal a través de muchas instancias de este tras un equilibrador de la carga.

La arquitectura monolítica define sus componentes de manera interconectada e interdependiente, con sus elementos, *estrechamente acoplados*, y aunque es cierto que puede contener varios ensamblajes en tiempo de ejecución, todas las capas se unen como un solo proceso.

La arquitectura monolítica obligaría a enfrentarse a los siguientes desafíos:

- Complejidad, número de módulos comerciales elevados y repetitividad en el código.
- Fallos individuales impactarían en el despliegue de todo el sistema.
- La escalabilidad impacta a todo el sistema, en vez de únicamente a aquellas piezas de software indicadas.
- Dependencia entre módulos. Acoplamiento estrecho.
- Tiempo de desarrollo en espiral, debido a la complejidad e interdependencia del código.
- Incapacidad de adaptarse fácilmente a una nueva tecnología.

2.2 Arquitectura Orientada a Servicios (SOA)

Este es un ***estilo arquitectónico*** que permite diseñar las aplicaciones empresariales como una colección de servicios que se ejecutan como ***procesos propios y separados***.

El servicio es el concepto clave de SOA. Básicamente podría definirse como una pieza de software que proporciona funcionalidad a otros componentes del sistema. Esta pieza de software puede interactuar con una BBDD directamente o a través de otro servicio, por ejemplo.

Un hecho diferencial de SOA con respecto a la arquitectura de software es el alcance. SOA se preocupa por brindar servicios consistentes en toda la empresa, de modo que puedan ser utilizados por muchas familias de soluciones diferentes. SOA debe promover el desarrollo de capacidades comerciales de tal manera que sean fácilmente reutilizadas por los diferentes procesos comerciales.

El valor de SOA aparece cuando estos servicios reutilizables se combinan para conformar procesos ágiles y flexibles, por lo tanto, parte de la arquitectura SOA es responsable de crear el entorno necesario para esto.

Dicho de otra manera, la arquitectura permite que distintos grupos o áreas desarrollen individualmente sus necesidades inmediatas, pero al mismo tiempo se define 'una plataforma' donde se puedan combinar en procesos comerciales de nivel superior o soluciones empresariales.

Para ello los servicios han de comunicarse a nivel técnico y semántico y no superponerse en las responsabilidades.

Hoy en día, la corriente principal de los servicios del sitio web Web 2.0 se encuentra en el estilo REST llamado servicio web **RESTful**. Estos servicios han sido ampliamente aceptados por el público debido a su facilidad de uso y simplicidad. Mientras tanto, las tecnologías de servicios web realizan una arquitectura orientada a servicios (SOA) con éxito y se explotan tanto en la industria como en el mundo académico.

En particular, la composición de servicios que puede agregar servicios existentes en uno nuevo ofrece la mayor cantidad de beneficios de SOA a través del estándar BPEL. Sin embargo, BPEL no puede admitir servicios RESTful e integrar recursos web 2.0 y basados

en SOA directamente en un servicio compuesto. Significa que es costoso o requiere mucho tiempo utilizar servicios RESTful en una aplicación basada en SOA.

2.3 Microservicios

Hoy en día no existe una definición precisa de este ***estilo arquitectónico***, aunque existen ciertas características comunes en torno a la organización y a la capacidad empresarial, la implementación automatizada, la inteligencia en los terminales y el control descentralizado de idiomas y datos.

No todas las arquitecturas de microservicios tienen todas las características comunes, aunque sí se espera que exhiban al menos un gran número de ellas.

Los microservicios son un ***enfoque arquitectónico*** y una especialización y un **estilo de SOA** que está compuesto por pequeños servicios que construyen sistemas de software flexibles e independientes.

Este estilo arquitectónico es un enfoque para desarrollar una sola aplicación como un conjunto de pequeños servicios, de manera que se ejecuta en su propio proceso y se comunican entre sí con mecanismos livianos.

Estos servicios se basan en las necesidades empresariales y se implementan de forma independiente. Pueden estar escritos en diferentes lenguajes de programación y utilizar diferentes tecnologías de almacenamiento de datos.

Así ***Lewis*** **y** ***Fowler*** [3] definieron algunas características principales de los microservicios como son su autonomía y la especialización; así como los beneficios que aporta este estilo, entre los que se pueden destacar la agilidad, el escalado flexible, la implementación sencilla, la libertad tecnológica, el código reutilizable y la resistencia.

Entre algunas de las ventajas destacan:

- **Modularidad:** Desarrollo y despliegue independiente, por ser servicios autónomos. Tolerancia a errores más acotada por ser componentes específicos de funcionalidad.
- **Escalabilidad:** al ser modular el escalado se produce horizontalmente.
- **Versatilidad:** elección de la tecnología en cada momento, pudiendo entonces elegir los grupos de desarrollo independientemente de su especialidad tecnológica.

- **Rapidez de actuación:** al tratarse de pequeñas piezas de código, los tiempos de desarrollo y resolución de incidencias se reduce notablemente.
- **Mantenimiento simple y barato:** el mantenimiento es más sencillo y barato que en otras arquitecturas por tratarse de piezas específicas aisladas del resto.
- **Agilidad:** una de las claves de los microservicios, es la reutilización de piezas, ya sean de terceros externas o desarrolladas dentro de la compañía por otros equipos.

Y teóricamente puede presentar una serie de desventajas:

- **Inversión de tiempo inicial:** al crear la arquitectura, se necesita más tiempo para poder fragmentar los distintos microservicios e implementar la comunicación entre ellos.
- **Complejidad en la gestión:** si contamos con un gran número de microservicios, será más complicado controlar la gestión e integración entre ellos. Es necesario disponer de una centralización de trazas y herramientas avanzadas de procesamiento de información que permitan tener una visión general de todos los microservicios y orquesten el sistema.
- **Perfil de desarrollador:** en no todos los ámbitos profesionales se cuenta con desarrolladores experimentados con un nivel muy alto de experiencia y un control exhaustivo de las versiones.
- **No uniformidad:** al disponer de un equipo tecnológico, ha de ponerse el foco en la gestión, si no todo ello podría conducir a un conjunto de aplicación poco uniforme.
- **Dificultad en la realización de pruebas:** en algunos casos los componentes de la aplicación al estar distribuidos, las pruebas y test globales pueden ser más complicados de realizar.
- **Coste de implantación alto:** una arquitectura de microservicios puede suponer un alto coste debido a determinadas piezas de software altamente especializadas.

2.4 Arquitectura dirigida por eventos (EDA)

Según la Wikipedia [6] La **Arquitectura dirigida por eventos**, *Event-driven architecture* o EDA, es un ***patrón de arquitectura*** **software** que promueve la producción, detección, consumo de, y reacción a eventos.

La arquitectura dirigida por eventos, a partir de ahora EDA, presenta una excelente solución para una variedad de desafíos importantes para TI organizacional. Aunque años atrás su logro ha estado frustrado, esto cambia con la llegada de los estilos arquitectónicos como SOA o el estilo arquitectural de los microservicios.

SOA 2.0 engloba las implicaciones de las arquitecturas SOA y EDA proporcionando a un más rico y más robusto nivel, creando un nuevo patrón de eventos. Este nuevo concepto de disparadores de patrones de inteligencia promueve a humanos autónomos o procesamiento automático que añade valor exponencial al negocio. Esto se debe a que se inyecta información de valor añadido en patrón reconocido que no podía haber sido obtenido previamente.

Un evento podría definirse de forma simple como “un cambio significativo en un estado". Por ejemplo, si excede su asignación de minutos de tiempo de teléfono celular, su proveedor de servicios inalámbricos le factura el excedente. En términos de EDA, el estado de sus minutos va de "Menos" a "Más", y ese cambio de estado activa la facturación del cargo por exceso. El valor del saldo de su cuenta cambia a medida que excede la asignación de minutos. El cambio de minutos = menos a minutos = más, es un evento.

Un evento tiene tres niveles de detalle:

- Ha ocurrido el evento.
- Definición del evento. Especificación propia del evento.
- El detalle del evento. Detalle que he generado ese evento.

Si nos vamos al ejemplo anterior, el evento ocurre o no ocurre. Posteriormente, es necesario la definición misma del evento (cambio de minutos) y por último se necesita saber el tiempo excedido, es decir, el detalle.

Ha de comprenderse que eventos no sólo hay en el ámbito computacional, por ejemplo, en verano con el calor, a partir de cierta temperatura, nuestro organismo genera eventos que

producen la sudoración, o a la inversa, en invierno con frío extremo nuestro cuerpo responde a eventos que provocan tiritar.

Y por último hay eventos que no siéndolo, podrían pasar a serlo; como por ejemplo, la contratación de valores en una plaza financiera. Si existiera algún interés en ello, podrían ser una fuente de datos que generasen ciertos eventos.

Así es que a partir de la definición de evento como '*cualquier cosa destacable'* que ocurre, y ahora sí dentro del ámbito comercial/empresarial una EDA es la gama completa de elementos arquitectónicos, incluidos el diseño, la planificación, la tecnología, organización, etc., lo que habilita la capacidad de difundir el evento de forma inmediata a todas las partes interesadas, humanas o automatizadas. [15]

Este tipo de arquitectura a menudo va muy ligada a tecnologías emergentes en la actualidad como la IoT, o el uso masivo de dispositivos móviles que actuarían como sensores, actuadores y/o controladores. Estos pueden detectar cambios de estado de objetos o condiciones y crear eventos que pueden ser procesados por un servicio o un sistema. Los disparadores de eventos son condiciones que tienen como resultado la creación de un evento.

Este **patrón arquitectónico** puede ser aplicado por el diseño e implementación de aplicaciones y sistemas que transmitan eventos entre componentes software que estén emparejados libremente y servicios. Un sistema dirigido por eventos está compuesto típicamente de emisores de eventos y consumidores de eventos.

Los consumidores tienen la responsabilidad de llevar a cabo una reacción tan pronto como el evento esté presente. La reacción puede o no ser completamente proporcionada por el consumidor en sí mismo. Por ejemplo, el consumidor debe tener solamente la responsabilidad de filtrar, transformar y reenviar el evento a otro componente o debe proporcionar una reacción propia a algún evento.

Construir aplicaciones y sistemas alrededor de una **arquitectura dirigida por eventos** permite a estas aplicaciones y sistemas ser construidos de una manera que facilita un **mayor grado de reacción**, debido a que los sistemas dirigidos por eventos están, por el diseño, más normalizados para entornos no predecibles y **asíncronos**.

La arquitectura dirigida por eventos puede complementar la arquitectura orientada a servicios (SOA) porque los servicios pueden ser activados por disparadores que se encuentran en eventos entrantes. Este paradigma es particularmente útil cuando el consumidor no proporciona algún contenedor ejecutivo propio.

Patrones básicos de procesamiento de evento [6]

Hay tres estilos generales de procesamiento de eventos: simple, flujo, y complejos. Los tres estilos se utilizan a menudo juntos en una arquitectura orientada a eventos madura.

Procesamiento de eventos simple

El procesamiento de eventos simples se refiere a los acontecimientos que están directamente relacionados con los cambios, específicos y medibles de la condición. En el procesamiento de evento simple, un evento notable sucede que inicia una acción directa. Se utiliza comúnmente para conducir el flujo en tiempo real de trabajo, reduciendo así el tiempo de retardo y el costo.

Como ejemplo, sensores de presión de los neumáticos o la temperatura ambiente. Este tipo de estilos es muy usado en aplicaciones en tiempo real, debido a su gran utilización por los cortos tiempos de respuesta que necesita.

Procesamiento por flujo de eventos

En el procesamiento de flujos de eventos (PFE), los procesadores de eventos solo reaccionan ante **una serie de criterios establecidos que han de cumplirse** para que se produzca el evento notable en el sistema. Se establece así una secuencia de procesamiento de eventos que se utiliza comúnmente para conducir el flujo de la información en tiempo real dentro y alrededor de la empresa, mejorando así la toma de decisiones a tiempo.

Procesamiento de eventos complejos

El procesamiento de eventos complejos (PEC) lleva el EDA a otro nivel, lo que le permite reaccionar a múltiples eventos bajo múltiples condiciones lógicas. El procesamiento de eventos complejos evalúa una confluencia de eventos y luego entra en acción.

El CEP escucha múltiples flujos de eventos y sabe cómo correlacionarlos de manera lógica de acuerdo a los objetivos del EDA. CEP requiere el empleo de sofisticadas intérpretes de eventos, la definición del modelo de eventos y correspondencia, y las técnicas de correlación.

Es decir, el CEP comienza a 'pensar', tomando fuentes de datos no relacionadas y diferenciando entre información útil o no.

En este último patrón de procesamiento de eventos, se ve claramente la coincidencia natural entre el CEP y la IA. Estos últimos tienen la capacidad 'natural' de reconocer patrones de información y sugerir reacciones a patrones de eventos.

Acoplamiento débil extremo y bien distribuidas

Una arquitectura orientada a eventos está **débilmente acoplada** y **bien distribuida**. Existe una gran distribución de esta arquitectura, ya que un evento puede ser casi cualquier cosa y existen en casi cualquier lugar.

La arquitectura se acopla muy vagamente porque el evento en sí mismo que no sabe nada de las consecuencias de su causa. Por ejemplo, si tenemos un sistema de alarma que registra la información cuando se abre la puerta, la puerta en sí no sabe que el sistema de alarma se sumará la información cuando se abre la puerta, solo que la puerta se ha abierto.

Las arquitecturas basadas en eventos se sueltan del acoplamiento en el espacio, el tiempo y la sincronización, proporcionando una infraestructura escalable para el intercambio de información y flujos de trabajo distribuidos.

Sin embargo, los eventos están estrechamente unidos a la arquitectura, a través de suscripciones de eventos y patrones, a la semántica del esquema de eventos y valores subyacentes.

En soluciones donde exista un alto grado de heterogeneidad semántica de los eventos en implementaciones, como las ciudades inteligentes, o como podría ser en búsqueda de información financiera en internet, donde existen grandes redes de sensores o potenciales fuentes de información actuando como tales, respectivamente, hace que sea difícil de desarrollar y mantener sistemas basados en eventos.

Ventajas y Desventajas

Ventajas

1. Simplicidad
2. Evolución: se pueden reemplazar componentes suscriptores
3. Modularidad: una sola modalidad para eventos diversos
4. Puede mejorar la eficiencia, eliminando la necesidad de polling por ocurrencia de evento

Desventajas

1. Posibilidad de desborde
2. Potencial imprevisión de escalabilidad
3. Pobre comprensibilidad: Puede ser difícil prever qué pasará en respuesta a una acción
4. No hay garantía del lado del publicador, que el suscriptor responderá al evento
5. No hay mucho soporte de recuperación en caso de falla parcial

2.5 Microservicios dirigidos por eventos

En la sección anterior se ha relatado la interconexión existente entre la EDA y SOA y la IoT o la comunicación móvil; incluso en sistemas CEP se ha referido de manera natural a la coincidencia natural con la IA, cuando se refiere al empleo de sofisticadas intérpretes de eventos, la definición del modelo de eventos y correspondencia, y las técnicas de correlación.

En esta sección se tratará de dar definición precisa de un sistema dirigido por microservicios impulsados por eventos.

Un microservicio impulsado por eventos es una pequeña aplicación creada para cumplir con un contexto delimitado específico. Los *microservicios de consumidor* consumen y procesan eventos de uno o más flujos de eventos de entrada, mientras que los microservicios de *productor* producen eventos en flujos de eventos para que otros servicios los consuman.

Es común que un microservicio controlado por eventos sea consumidor de un conjunto de flujos de eventos de entrada y productor de otro conjunto de flujos de eventos de

salida. Estos servicios pueden ser sin estado o con estado y también pueden contener API síncronas de solicitud-respuesta.

Todos estos servicios comparten la funcionalidad común de consumir eventos o producir eventos para el intermediario de eventos. La comunicación entre microservicios controlados por eventos es completamente asincrónica. [16]

Los flujos de eventos son atendidos por un *agente de eventos*. La ejecución de microservicios a cualquier escala significativa a menudo requiere el uso de canalizaciones de implementación y sistemas de administración de contenedores.

En este caso todas las definiciones anteriores respecto a los eventos son útiles para esta definición, aunque sí se hará una definición específica más en detalle de la composición de este.

Un evento se representa normalmente mediante formato *clave, valor.* La clave se usa para fines de identificación, enrutamiento y operaciones de agregación con eventos con la misma clave, mientras que el valor almacena los detalles completos del evento.

Los datos de eventos sirven como medio de almacenamiento de datos independiente de la implementación y a largo plazo, así como como mecanismo de comunicación entre servicios. Por lo tanto, es importante que tanto los productores como los consumidores de eventos tengan un entendimiento común del significado de los datos.

Idealmente, el consumidor debe poder interpretar el contenido y el significado de un evento sin tener que consultar con el propietario del servicio productor. Esto requiere un lenguaje común para la comunicación entre productores y consumidores de manera análoga a, por ejemplo, una definición de API entre servicios de solicitud-respuesta síncronos.

En el corazón de cada plataforma de microservicio impulsada por eventos para producción, se encuentra el **intermediario de eventos**. Este es un sistema que recibe eventos, los almacena en una cola o secuencia de eventos particionada y los proporciona para el consumo de otros procesos. Los eventos generalmente se publican en diferentes flujos según su significado lógico subyacente, similar a cómo una base de datos tendrá muchas tablas, cada una separada lógicamente para contener un tipo específico de datos.

Los sistemas de intermediarios de eventos adecuados para empresas a gran escala generalmente siguen el mismo modelo. Varios agentes de eventos distribuidos trabajan juntos en un clúster para proporcionar una plataforma para la producción y el consumo de flujos de eventos. Este modelo proporciona varias características esenciales que se requieren para ejecutar un ecosistema impulsado por eventos a escala:

Escalabilidad

Se pueden agregar instancias de intermediarios de eventos adicionales para aumentar la producción, el consumo y la capacidad de almacenamiento de datos del clúster.

Durabilidad

Los datos de eventos se replican entre nodos. Esto permite que un grupo de intermediarios conserve y continúe proporcionando datos cuando falla un intermediario.

Alta disponibilidad

Un clúster de nodos de intermediarios de eventos permite a los clientes conectarse a otros nodos en caso de falla del intermediario. Esto permite a los clientes mantener un tiempo de actividad completo.

Alto rendimiento

Varios nodos de intermediarios comparten la carga de producción y consumo. Además, cada nodo de intermediario debe tener un alto rendimiento para poder manejar cientos de miles de escrituras o lecturas por segundo.

Aunque hay diferentes formas en las que los datos de eventos se pueden almacenar, replicar y acceder entre bastidores de un agente de eventos, todos ellos generalmente brindan los mismos mecanismos de almacenamiento y acceso a sus clientes.

Agentes de eventos versus agentes de mensajes

Hay cierta confusión entre un agente de mensajes y lo que constituye un agente de eventos. Los intermediarios de eventos se pueden utilizar en lugar de un intermediario de

mensajes, pero un intermediario de mensajes no puede cumplir con todas las funciones de un intermediario de eventos.

Los intermediarios de mensajes tienen una larga trayectoria y numerosas organizaciones los han utilizado en arquitecturas de middleware orientadas a mensajes a gran escala. Los agentes de mensajes permiten que los sistemas se comuniquen a través de una red a través de colas de mensajes de publicación / suscripción.

Los productores escriben mensajes en una cola, mientras que un consumidor consume estos mensajes y los procesa en consecuencia. Los mensajes se reconocen como consumidos y se eliminan inmediatamente o poco después. Los intermediarios de mensajes están diseñados para manejar un tipo de problema diferente al de los intermediarios de eventos.

Los agentes de eventos, por otro lado, están diseñados para proporcionar un registro ordenado de hechos. Los intermediarios de eventos satisfacen dos necesidades muy específicas que el intermediario de mensajes no satisface. Por un lado, el intermediario de mensajes solo proporciona colas de mensajes, donde el consumo del mensaje se gestiona por cola.

Las aplicaciones que comparten el consumo de una cola recibirán cada una solo un subconjunto de los registros. Esto hace que sea imposible comunicar correctamente el estado a través de eventos, ya que cada consumidor no puede obtener una copia completa de todos los eventos.

A diferencia del intermediario de mensajes, el intermediario de eventos mantiene un solo libro mayor de registros y administra el acceso individual a través de índices, por lo que cada consumidor independiente puede acceder a todos los eventos requeridos. Además, un agente de mensajes elimina los eventos después del reconocimiento, mientras que un agente de eventos los retiene durante el tiempo que la organización lo necesite.

La eliminación del evento después del consumo hace que un intermediario de mensajes sea insuficiente para proporcionar la fuente única de verdad almacenada indefinidamente, accesible globalmente, reproducible y única para todas las aplicaciones.

Capítulo 3. Descripción funcional de la solución

Las distintas etapas por las que pasa la información desde la recepción hasta su difusión en ficheros a cada cliente, así como el control sobre la recepción, procesamiento, enriquecimiento, almacenaje, difusión de la información y como la definición de clientes, productos, etc. son:

- Recepción de la información
- Control de recepción
- Procesamiento de la información
    - Enriquecimiento de la información
    - Generación y gestión de ficheros
- Difusión de ficheros
- Supervisión del sistema
- Gestión y almacenaje de la información recibida, procesada y enviada
- Gestión de clientes y productos.

El siguiente diagrama de alto nivel describe, a grandes rasgos, los elementos esenciales que intervienen en la recepción de la información:

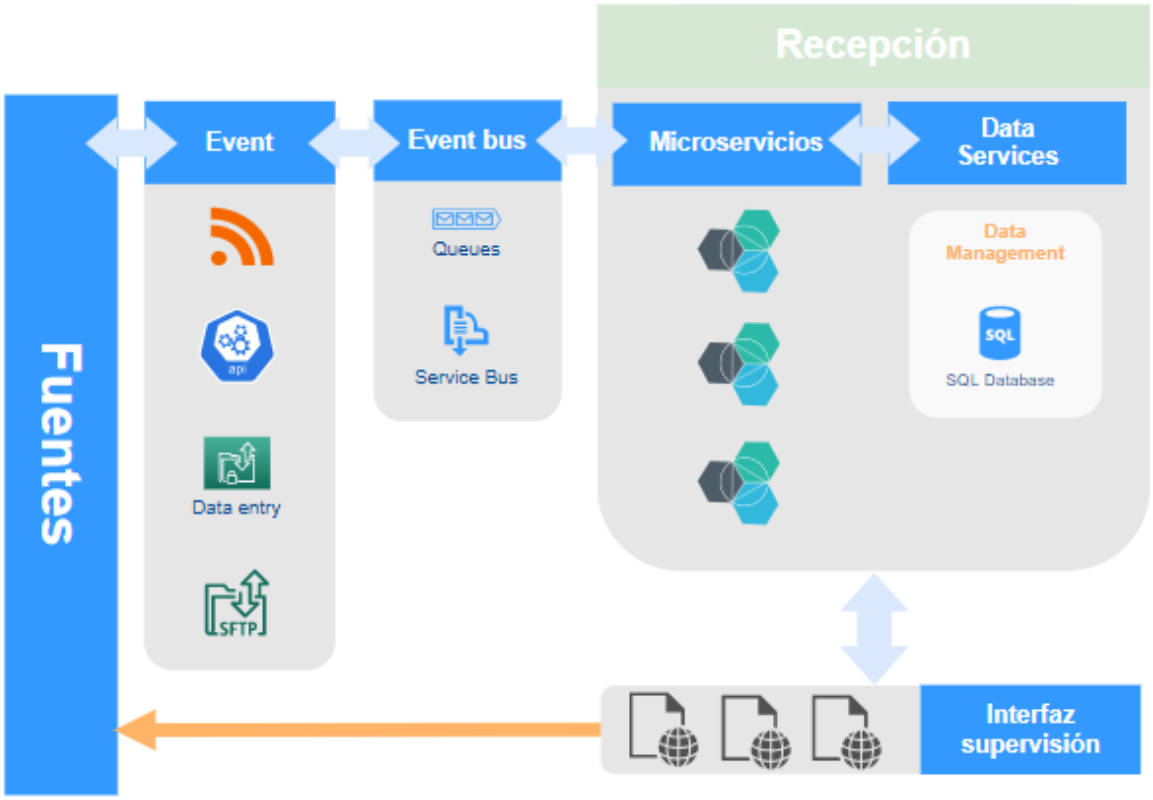

Figura 2: Recepción de información.

3.1 Recepción de la información

3.1.1 Fuentes

Las fuentes de la información a incluir en los productos son las siguientes:

- Bolsas de Valores suscritas al servicio.
- Superintendencias.
- Data Entry.
- Emisores.
- Otras fuentes:
  - Bancos Centrales.
  - Contribuidores de precios.
- Procesos internos de final de día.
- Información de final de día a cierre de mercado.

3.1.2 Canales de recepción

La información se recibirá al menos a través de los siguientes canales:

- **Servicio FTP/SFTP.**
  - Ofrecido por la fuente: La fuente pondrá un servicio FTP/SFTP en el que se le dará acceso al buzón asignado, para recuperar la información requerida. El sistema debe ser capaz de acceder a los buzones al menos con los métodos de acceso más utilizados (activo, pasivo, con Password, a través de claves públicas y privadas, con la información encriptada,).
  - Ofrecido por el sistema: se pondrá a disposición de cada fuente un buzón dentro del servicio SFTP para que se conecte a él y deposite la información requerida utilizando claves públicas y privadas para ello.
- **Feed RSS:** en el caso de que la fuente ponga a disposición de sus clientes *feeds* RSS para la recepción de información, se evaluará la posibilidad de que el sistema se suscriba a dicha *feed* para acudir a los documentos a los que se hagan referencia en los mensajes de la misma.
- **API:** en el caso en el que la fuente de información envíe la información a través de un API, el sistema podrá recibir la información correspondiente a través de la misma.
- **Servicios de almacenamiento en la Red de Internet (Nube).** Las fuentes pueden disponer de servicios de almacenamiento de la información en la Nube (propios o ajenos) donde poder acceder a bajarse la información requerida.
- **Data entry:** se dispondrá de un aplicativo para la introducción manual de información. Este data entry será tratado como una fuente adicional de información.
- **Web de contribución:** se pondrá a disposición de las fuentes una web de contribución a través de la cual podrán enviar la información correspondiente.
- **Repositorios de información en la intranet.** Las fuentes pueden disponer de servidores de datos propios donde poder acceder a la información requerida. Esta información resulta de otros procesos internos que generan información para su distribución al exterior.

La implementación de posibles canales de recepción de información adicionales dependerá de los métodos de envío de información empleados por las fuentes.

Se debe tener la posibilidad de forzar manualmente la carga de la información, incluso de cargar archivos manualmente si fallasen los procesos de entrega de la información de la fuente o los procesos de captura de información.

3.1.3 Formatos

El aplicativo debe ser capaz de tratar y almacenar cualquier tipo de información que esté disponible al menos en los siguientes formatos:

- Fichero de texto plano del tipo CSV
- Ficheros de lenguaje de marcas del tipo XML, JSON, XBRL, etc.
- Ficheros asociados a documentos de texto que pueden contener gráficos e incluso imágenes, del tipo PDF, DOC, Excel, etc.

Las fuentes proveerán la información en diversos formatos. Dependiendo del tipo de información recibida, los formatos más comúnmente utilizados son los siguientes:

- Maestro de Valores y Precios y Volúmenes: fichero de texto plano CSV, JSON o XML entre otros.
- Hechos Relevantes: XML o XBRL con documentos adjuntos (PDF, imágenes, DOC, etc.)
- Índices, Boletines, Tick Data: ficheros PDF, TXT, ZIP.

3.1.4 Almacenamiento

Los sistemas de almacenamiento que se necesitará para registrar toda la información que se reciba de los **contribuidores** y se genere, serán los siguientes:

- Un **servidor de ficheros** que hará de repositorio central de ficheros donde se almacenará la información recibida a través de las distintas fuentes tal y como ha sido entregada.
- Una **base de datos** que registre la información recibida de las fuentes una vez validada y clasificada.

Estos sistemas de almacenamiento deberán contar con un sistema de ***contingencia*** sobre la información almacenada en los servicios de ficheros y base de datos, así como un control de información histórica que permita enviar a un repositorio de información histórica toda aquella información que ya haya sido tratada y sea necesario seguir manteniéndola, pero fuera del sistema principal.

La información se almacenará tal y como se reciba de los clientes en un directorio destinado a tal efecto y se registrará la entrada de la información en base de datos. En el caso de los Hechos Relevantes, se descargarán todos los archivos adjuntos a la comunicación y se almacenarán en un directorio destinado a tal efecto.

3.1.5 Envíos de confirmación de recepción

Se genera el envío a la fuente de una confirmación de recepción correcta de la información. Se generarán alertas a nivel de operador cuando no se reciba la información.

3.1.6 Horarios de recepción

Los horarios de entrega de información para cada una de las fuentes de información están vinculadas al tipo de información que estén difundiendo y a los eventos que provoquen la generación de esta información (por ejemplo, apertura y/o cierre de los mercados, comunicación de hechos relevante, etc).

El sistema tendrá en cuenta ciertas características específicas que pueden presentarse y entre las que se detecta las siguientes:

- la posibilidad de recibir ficheros incrementales además de ficheros full, que incluyen únicamente modificaciones a los archivos enviados con anterioridad,
- o el envío de múltiples ficheros a lo largo de la sesión, que el sistema rastreará periódicamente (cada 5 minutos) el sistema de entrada de información a fin de descargar nuevos contenidos.

3.1.7 Recepción de Información

Se recibirá la información de cada una de las fuentes, siendo capaz de:

- Reprocesar la información recibida de las fuentes tantas veces como la envíe la fuente y si fuera necesario fuera de los horarios establecidos; e incluso, se debe permitir procesar información de otros días, siempre con métodos propios según la naturaleza de cada 'componente'. También será capaz de cargar archivos varias veces al día si la fuente envía archivos modificados resultantes de la revisión de la información remitida en un archivo anterior.
- Clasificar la información recibida por fuente y tipo de información y posteriormente almacenarla en el repositorio central de ficheros recibidos de las fuentes para su posterior procesamiento.

Las fuentes de información consideradas hasta el momento son:

- Maestro de Valores: la información del Maestro de Valores se puede recibir, dependiendo de la fuente, una o varias veces durante el día. Lo habitual será recibir de la fuente un archivo a cierre de la sesión con información válida para la siguiente sesión bursátil (ie. información en D-1 válida para D). No obstante, es posible que las fuentes envíen archivos incrementales con información de valores actualizada a lo largo de la sesión o inmediatamente antes de la apertura.
- Precios y Volúmenes: la información de Precios y Volúmenes se recibirá una vez al día, tras el cierre de los mercados, salvo casos en los que la bolsa envíe archivos con correcciones, en cuyo caso será necesario volver a recoger y cargar el archivo.
- Hechos Relevantes: los Hechos Relevantes se podrán recibir en cualquier momento del día, si bien es previsible que haya más Hechos Relevantes antes y después del horario normal del mercado local. Como se ha indicado con anterioridad, el sistema rastreará los servidores de hechos relevantes cada 5 minutos a fin de cargar contenidos nuevos que vaya creando la fuente.

3.1.8 Incidencias en la recepción de información

Las posibles incidencias que se prevén en la recepción de información estarán clasificadas en dos grandes grupos:

Control de calidad: incidencias asociadas al contenido de la información.

Estas incidencias se solventarán por el operador de la solución. Entre otras, se prevén incidencias relacionadas con:

- Problemas de formato en la información.
- Retrasos en la entrega de la información
- Archivos disponibles, pero parcialmente, o vacíos.
- Problemas de privilegios de acceso a la información.
- Otros problemas a la hora de descargar la información.
- …

Excepciones en Operación/Explotación: incidencias generadas por algún problema técnico

El operador de la solución elevará estas incidencias al ***soporte de Nivel II***. Entre otras, se prevén incidencias relacionadas con:

- Problemas en los servicios FTP/SFTP, Feed RSS etc. que impidan la descarga de la información (por ejemplo: caída del servicio, credenciales no reconocidas, problemas de comunicaciones).
- Problemas técnicos relacionados con el Módulo de Recepción de Información (módulo no arrancado u operativo).
- Problemas con el servidor de ficheros o base de datos (por ejemplo, que no estén accesibles o tengan algún problema).
- Problemas en las comunicaciones hacia el exterior del sistema que impidan acceder a la información.
- …

Cualquier incidencia que se produzca debe estar recogida en un fichero o tabla de base de datos y podrá ser consultada por los usuarios supervisores del **Módulo de Recepción de Información,** así como por el departamento de Explotación si así lo necesitara.

Habrá diferentes niveles de alerta que podrán ser filtradas por los operadores del sistema.

3.2 Procesamiento de la información

El procesamiento de la información se hará en base al tipo de formato de la información a tratar y los clasificaremos en dos:

- **Los ficheros exportables.** Estos son ficheros que no reciben ningún tratamiento. Simplemente se transfieren.
- Los **ficheros** fácilmente **procesables**. Serían aquellos cuyos contenidos son de fácil extracción y almacenaje como el caso de ficheros de texto plano, CSV, XML, …
- **Ficheros con tratamientos especiales** que sería aquellos que para extraer la información necesitan un procesamiento especial y que pueden contener referencias a otros documentos. Estos ficheros serían los disponibles en formato PDF, Word, …

3.2.1 Ficheros exportables

Los ficheros que se obtendrán de las fuentes con información de Maestro de Valores, Precios y Volúmenes y ficheros resumen de Hechos Relevantes, boletines, Ticks Data.

En estos casos, se transfiere el archivo punto a punto sin más tratamiento adicional.

3.2.2 Ficheros procesables

El siguiente diagrama describe de manera somera los elementos esenciales que intervienen en el procesamiento de ficheros de fácil extracción.

Inicialmente los ficheros que se obtendrán de las fuentes en un formato fácil serán los ficheros con información de Maestro de Valores, Precios y Volúmenes y ficheros resumen de Hechos Relevantes, boletines, Ticks Data, estos tres últimos provenientes de entidades terceras.

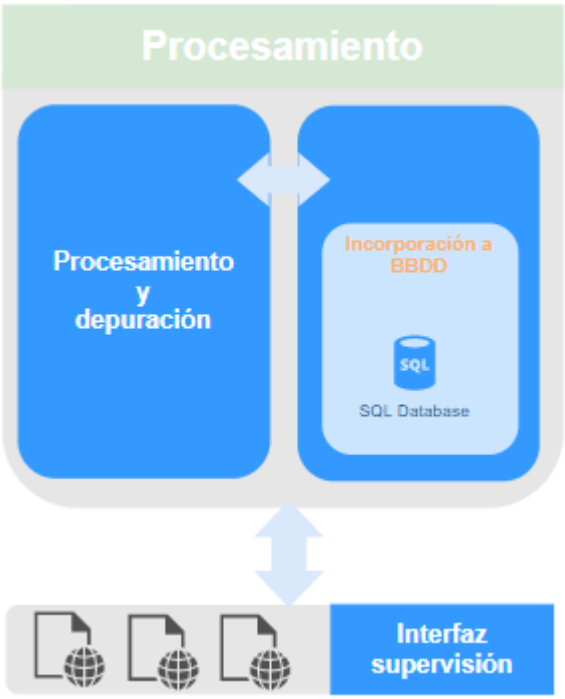

Figura 3: procesamiento de información de Maestro de Valores / Precios y Volúmenes.

En algunos casos habrá que extraer y almacenar la información contenida en este tipo de ficheros para lo que habrá que realizar al menos las siguientes tareas:

- Validación de la información: Se validará que el fichero con la información recibida de la fuente viene con todos los campos informados y cumple con los formatos establecidos. En caso contrario se generará una incidencia. se detectan inconsistencias, se contactará con la fuente que será la responsable de actualizar los archivos con la información correcta.
- Conversión a Estándar: se depurará la información recibida de acuerdo con los parámetros establecidos. Este depurado implicará, por ejemplo, adaptar los contenidos recibidos en determinados campos al Estándar.

- Enriquecimiento de la información: la información recibida de las fuentes será completada por el sistema con contenidos estáticos almacenados en las BBDD y/o recibida de otras fuentes.

Los contenidos con los que se enriquecerá la información almacenada en BBDD, en caso de que no esté incluida en la información recibida de las fuentes, serán, al menos, los siguientes:

- LEI de la emisora (siempre y cuando exista una referencia, por ejemplo, un código local, para el cruce de la información contra la tabla que almacena los códigos LEI)
- NIF nacional de la emisora (siempre y cuando exista una referencia, por ejemplo, un código local, para el cruce de la información contra la tabla que almacena los códigos LEI)
- MIC del mercado donde cotiza el instrumento
- Divisa en la que cotiza el instrumento
- Código CFI del instrumento
- Timestamp con la fecha y la hora de recepción de la información

Para el enriquecimiento de la información, se dispondrá de datos adicionales que se almacenarán de las siguientes fuentes:

i. LEI de las emisoras (ISO 17442)
ii. MIC del mercado (ISO 10383)
iii. Códigos de monedas (ISO 4217)
iv. Códigos ISIN de las emisiones (ISO 6166)
v. Códigos del país (ISO 3166)

En otros casos solamente será necesario transferir los ficheros punto a punto, sin necesidad de validar, ni tratar el contenido de la información en ellos, aunque sí ofrecerán algún tipo de lógica de movimiento previo de los ficheros o de lógica de negocio.

3.2.3 Ficheros con tratamiento especial

El siguiente diagrama describe de manera somera los elementos esenciales que intervienen en el procesamiento especial que requieren los ficheros de información de Hechos Relevantes o Noticias:

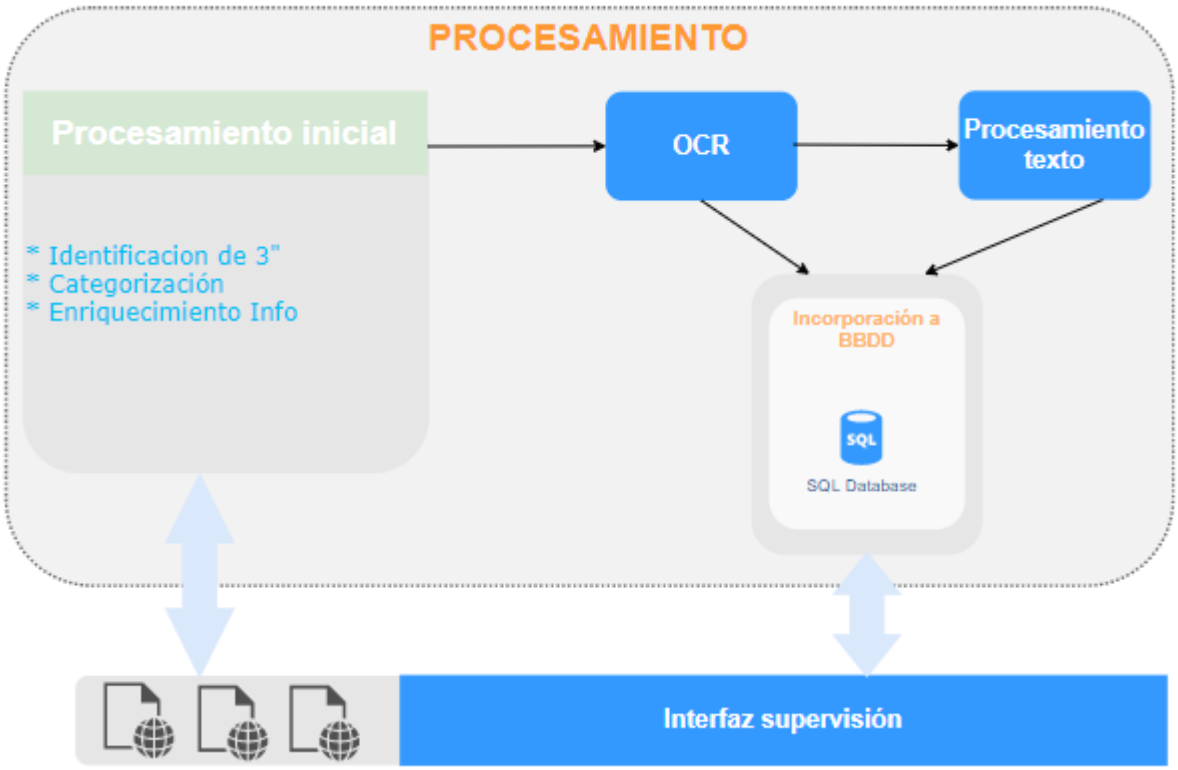

Figura 4: procesamiento de información de Hechos Relevantes.

Procesamiento inicial

En cuanto se reciba un fichero con información de un nuevo Hecho Relevante o Noticia, se realizarán las siguientes tareas:

- Identificación de posibles terceras partes implicadas (p.e. auditores, agencias de calificación crediticia, etc.).
- Categorización del Hecho Relevante según el Estándar: se accederá a una tabla de equivalencias en la que se asigne, para cada tipo de hecho relevante de la fuente, un tipo de hecho relevante de acuerdo con el Estándar.

- Enriquecimiento de la información: la información recibida de las fuentes será completada por el sistema con contenidos estáticos almacenados en las BBDD relativos al emisor, de tal forma que el sistema enriquezca la información del hecho relevante. Los contenidos con los que se enriquecerá la información son los detallados a continuación:
    i. LEI de la emisora
    ii. NIF nacional de la emisora

Procesamiento OCR

La información contenida en los documentos adjuntos de ciertos tipos de Hecho Relevante se procesará mediante herramientas de **análisis de textos u *OCR*** con la finalidad de extraer la información que corresponda para su incorporación a ficheros *machine readable.*

Los tipos de Hechos Relevantes que, en una primera fase, se procesarán con estas herramientas son los siguientes:

- Anuncio de dividendos
- Ampliaciones de capital
- Splits / Contrasplits
- Emisiones de instrumentos de Renta Fija

En un primer paso, las herramientas de procesamiento de texto extraerán la información de todos los documentos adjuntos y la almacenarán en formato de texto plano. Esta información ya puede ser difundida a los clientes.

Posteriormente, se establecerá un procedimiento para que la información pueda ser procesada y de ella se extraigan los contenidos que debe incluirse en los ficheros ***machine readable***.

Los campos de los ficheros *machine readable* vendrán definidos en los documentos de especificaciones técnicas correspondientes.

En BBDD se almacenarán:

a) Los documentos adjuntos en su formato original (ya mencionado en la sección de Recepción de la información)

b) El texto extraído en formato CSV

c) Los contenidos extraídos para su incorporación a los ficheros ***machine readable.***

3.24 Archivo resumen

El sistema creará un archivo resumen que contendrá la información de cada hecho relevante que se procese. Este archivo se sobrescribirá cada vez que se disponga de un hecho relevante nuevo, es decir añadirá al archivo una línea nueva por cada hecho relevante que se procese.

El archivo se irá actualizando durante las 24 horas del día, y se reiniciará a las 00:00 de cada día laborable.

3.2.5 Incidencias en el procesamiento de ficheros

Ficheros procesables

- Incidencias: en el caso de producirse algún tipo de incidencia en el procesamiento de la información a la BBDD, se generará una incidencia que podrá ser visualizada por el operador de la solución y le permitirá actuar en consecuencia. Entre otras incidencias, se prevén las siguientes:
    - Información errónea: en el caso de que la información recibida incluya información errónea:
        - campos vacíos
        - contenidos no coincidentes con lo indicado en las especificaciones técnicas (por ejemplo, campo ISIN con formato incorrecto)

Se generará una incidencia para que la gestione el operador de la solución en comunicación con la fuente. Como se ha indicado con anterioridad, el operador de la solución en ningún caso modificará la información enviada por la fuente.

- Error en la carga de la información a base de datos: el operador contactará con el soporte de Nivel II a fin de comunicar la incidencia

- Error en el enriquecimiento de datos:
    - Emisora no encontrada

- Instrumento no encontrado
- Datos no disponibles

El operador contactará con el soporte de Nivel II a fin de comunicar la incidencia

Las incidencias generadas en el módulo de procesamiento se gestionarán a través del Interfaz de Supervisión y se almacenarán en BBDD. Las potenciales incidencias serán las siguientes:

- Errores a la hora de asociar el HHRR a una emisora
- Errores a la hora de tipificar el HHRR
- Error en el enriquecimiento de la información:
  - Datos de entidad no disponible
  - Datos de terceras partes no disponible
- Error en el procesamiento OCR:
  - Error en la extracción de información
  - Error del aplicativo OCR
  - Error en el almacenamiento de la información

3.3 Difusión

La gestión de ficheros, clientes, contenidos y de canales de difusión se gestionarán a través de un **interfaz de gestión de ficheros** (en adelante "interfaz") diseñado a tal efecto.

El siguiente diagrama de alto nivel describe los elementos esenciales que intervienen en la difusión de información:

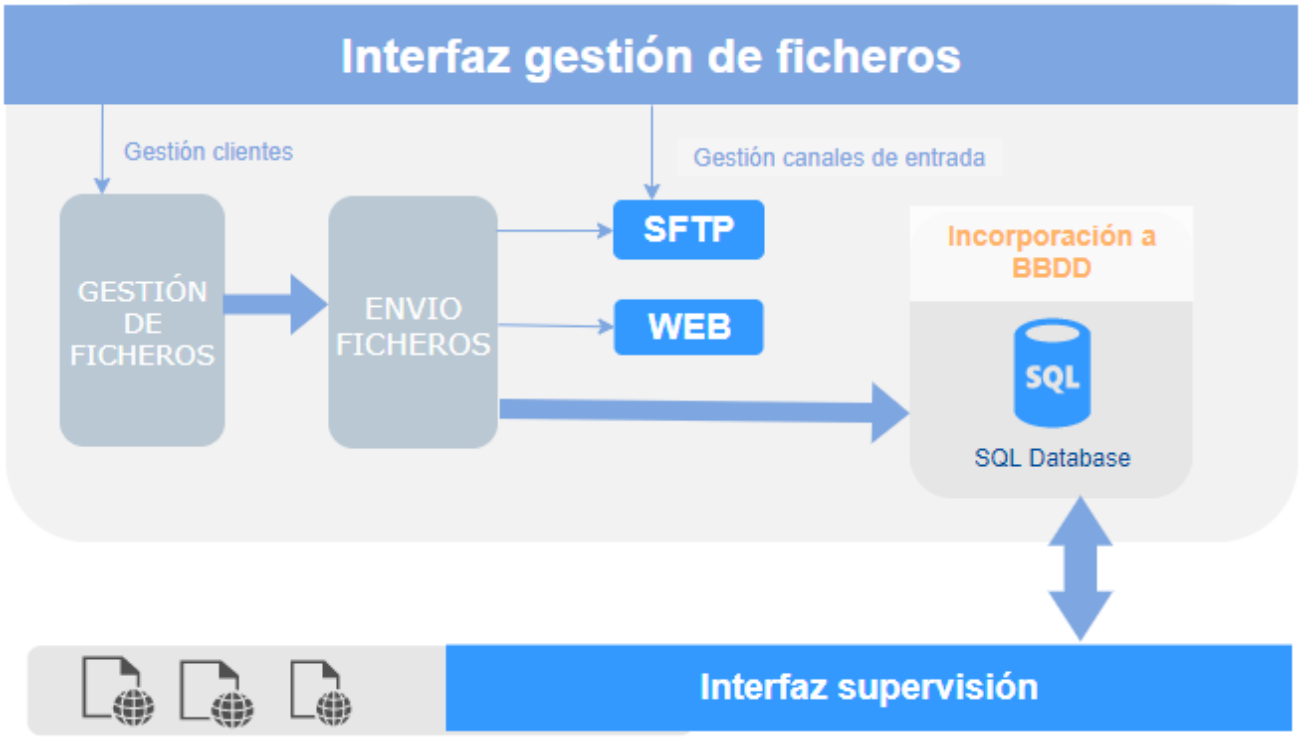

Figura 5: Interfaz de gestión de ficheros.

3.3.1 Gestión de ficheros

El interfaz llevará a cabo las siguientes tareas:

a. Producción de los archivos:

a. Maestro de valores y precios y volúmenes: el interfaz creará los archivos tal y como están definidos. Se estandariza la nomenclatura utilizada para los ficheros, sus contenidos, formas de entrega, etc.

Los archivos se proporcionarán por país, así como por activo (identificado éste último por la primera letra del código CFI del activo al que hace referencia), de tal forma que la entrega de la información podrá alinearse con el cierre de cada mercado en cada país. Se proporcionarán, asimismo, archivos incrementales a lo largo de la sesión cuando la fuente envíe información actualizada fuera del horario de recepción del archivo full.

Se establecen los formatos CSV, JSON y XML para todos los archivos de maestro de valores y precios y volúmenes.

b. Hechos Relevantes: el interfaz creará el Hecho Relevante según el estándar:

i. Texto del Hecho relevante, descripción, tipología y link a documentación adjunta.

ii. Texto del Texto del Hecho relevante, descripción, tipología y documentación adjunta en formato texto.

iii. Hecho relevante en formato *machine readable*. Estos hechos relevantes podrán estar disponibles bien directamente de la fuente, bien a través de data entry, bien a través de los procedimientos de lectura del OCR.

Una vez creados los archivos y los HHRR, se almacenarán en BBDD y se pondrán a disposición de Gestor de Archivos (file dispatcher) tan pronto como estén disponibles.

b. Generación de ficheros ad-hoc/one-off:

El interfaz tendrá un módulo para permitir la creación de ficheros de información histórica. El interfaz permitirá al administrador lo siguiente:

i. Elegir los contenidos según el producto que necesite el cliente, según los siguientes parámetros.
    - Activo
    - Mercado
    - País

ii. Elegir los formatos de salida de la información (XML, csv, JSON…)

iii. Elegir el rango de fechas a incluir.
    - Información de Maestro de Valores: los ficheros incluirán, para la fecha inicial, información de todos los instrumentos vivos a la fecha. Para el resto del periodo, se incluirá únicamente información sobre las altas/bajas/modificaciones acontecidas hasta la fecha final.
    - Información de Hechos Relevantes y Precios y Volúmenes: se incluirá toda la información almacenada para el rango de fechas seleccionado. Para el caso de hechos relevantes, se extraerá la información que se encuentre almacenada en BBDD, no la información adjunta que ha enviado el emisor.

Asimismo, este módulo permitirá al gestor seleccionar los archivos que necesita de manera sencilla y manual para peticiones ad-hoc del cliente (p.e. si un cliente solicita un fichero determinado de una fecha previa).

3.3.2 Distribución de la información (file dispatcher)

Una vez generados los ficheros, y también para los casos en que no haya sido necesario generarlos, el interfaz dispondrá de un sistema (file dispatcher) que gestionará la copia de dichos ficheros en las carpetas de los clientes que hayan contratado el producto.

El interfaz tendrá que contemplar lo siguiente:

- Deberá soportar todos los formatos posibles (txt, json, pdf, imágenes, archivos comprimidos, etc)
- Deberá realizar un control de la entrega efectiva de los ficheros, generando alarmas si no es capaz de entregar los ficheros conforme a lo establecido.

- Deberá soportar la entrega de un mismo archivo varias veces en el mismo día para soportar posibles modificaciones.
- Deberá programar borrados recurrentes de la información para evitar saturar el sistema (por ejemplo, borrar archivos con una antigüedad superior a los 20 días o configurable en algunos casos).

El envío de ficheros se monitorizará a través del Interfaz de Supervisión. Se almacenará un *log* en el que se detalle lo siguiente:

- Nombre o código del fichero.
- Fichero enviado.
- Formato del fichero.
- Fecha y hora de envío.
- Nombre del cliente.
- Canal de difusión.
- Incidencia (S/N).

Las incidencias que deberá registrar el sistema, en una primera etapa, son las siguientes:

- Error en la generación de ficheros (no se han generado o se han generado vacíos).
- Error en el envío de los ficheros.
- Retraso en la generación o en el envío de ficheros.
- Error en el almacenamiento de los ficheros difundidos.
- Error en la entrega de e-mails.
- Error por problemas de tamaño.

3.3.3 Gestión de canales de distribución (envío de ficheros)

El interfaz permitirá al gestor de la solución activar o desactivar canales de distribución de información de acuerdo con las necesidades de los clientes (e.g. activar un directorio SFTP, activar direcciones de email para el envío de notificaciones…).

La estructura de carpetas seguirá la siguiente lógica:

Cada cliente dispondrá de su propia carpeta dentro del servidor SFTP. Sólo tendrá acceso a esa información el cliente que ha contratado la información, mediante la entrega de claves públicas/privadas.

Un mismo cliente podrá tener varias carpetas en el servidor SFTP, bien para distintos productos, bien para distintas filiales que, en su caso, serán tratadas como clientes diferentes. En estos casos, se valorará crear una carpeta "raíz" para el cliente y subcarpetas para productos/filiales, por ejemplo:

- Refinitiv
    - Master Data EQ
    - Master Data IF
    - Corporate Actions machine readable
- Bloomberg
    - Bloomberg Master Data
        - Master Data EQ
        - Master Data FI
    - Bloomberg Corporate Actions
        - Corporate Actions machine readable
    - Bloomberg UK
        - Master Data EQ
        - Coporate Actions machine readable
    - Bloomberg USA
        - Master Data FI

Dentro de cada subcarpeta, la información se clasificará por días, según el formato ISO de fechas. Para todos los casos se tomarán las horas UTC para la determinación del día.
Por ejemplo:

- Refinitiv
    - Master Data EQ
        - 2019-12-01
        - 2019-12-02
        - 2019-12-03
    - Master Data IF

        - 2019-12-01
        - 2019-12-02
        - 2019-12-03
    - Corporate Actions machine readable
        - 2019-12-01
        - 2019-12-02
        - 2019-12-03

Hay otros casos, dependiendo de la naturaleza del componente, tiene su propia configuración de estructura de carpetas de salida.

3.4 Gestión de clientes

El aplicativo dispondrá de un módulo que permita gestionar la información de los clientes, así como los productos que cada cliente tiene contratados.

La entrada de la información tendrá lugar de dos maneras:

a) Carga directa de la información a través del formulario de condiciones particulares del contrato: el cliente firmará un contrato para recibir la información, al que se anexará un formulario en archivo Excel con los datos del cliente y los productos contratados. Este formulario se podrá cargar directamente al sistema. Este formulario vendrá definido en el manual de explotación del sistema.

b) Carga manual de la información: el gestor del aplicativo tendrá acceso a un "data entry" desde el que cargar la información del cliente y gestionar los productos que ha contratado y las formas de entrega.

A **través del interfaz** el administrador podrá realizar las siguientes tareas:

a. ***Alta de cliente:*** el interfaz permitirá dar de alta clientes según las modalidades definidas en el punto anterior. El administrador introducirá los siguientes datos del cliente:

   i. ***Nombre del cliente***
   ii. ***CIF/VAT number o equivalente*** *(clave para identificar al cliente)*
   iii. ***Personas de contacto*** *(se permiten varias)*
   iv. ***Direcciones de email para las notificaciones*** *(se permiten varias)*
   v. ***Fecha de alta del cliente***

b. Alta de filiales: el interfaz permitirá dar de alta a las distintas filiales que el cliente quiera incluir en el contrato. Las filiales serán tratadas como clientes diferentes, con carpetas propias que incluyan la información contratada.

c. ***Gestión de productos y formas de entrega:*** El interfaz mostrará al gestor el catálogo de productos para que éste pueda asociar el producto contratado al cliente. Mostrará:

   i. *Productos disponibles*
   ii. *Formatos disponibles*
   iii. *Formas de entrega disponibles*
   iv. *Fecha de inicio de provisión de la información (por producto)*

   El gestor asociará al cliente tanto productos contratados como formatos seleccionados, formas de entrega y tarifas aplicables.

   Se permitirá la modificación de los productos contratados (discontinuando o añadiendo productos según solicite el cliente), con la consiguiente actualización en las fechas de provisión/discontinuación de la información.

d. ***Modificación de información de clientes/filiales:*** La modificación de información de clientes se realizará de manera manual. El sistema permitirá modificar cualquier información almacenada relacionada con el cliente (dirección, nombre, CIF/VAT o similar, productos contratados, canales de difusión…), almacenando en BBDD tanto la información anterior como la fecha efectiva del cambio.

e. ***Baja de clientes/filiales:*** La baja de clientes se realizará de manera manual. Se establecerá una fecha de baja desde la cual el cliente ya no podrá recibir más la información que había contratado.

   A partir de esa fecha, el sistema no pondrá a disposición del cliente el producto que tenía contratado y discontinuará la emisión de mensajería por e-mail (notificaciones o productos si esta es la vía seleccionada). No obstante, se mantendrá accesible tanto la carpeta SFTP habilitada para el cliente como los archivos de fechas anteriores a la baja efectiva hasta que pase el proceso de limpia del SFTP.

   Por cliente, se entiende el firmante o la filial. Es decir, un cliente podrá solicitar la baja de un producto que sólo recibía una filial.

f. ***Generación de informes:*** El interfaz permitirá al administrador consultar información sobre el listado de clientes (tanto actuales, como históricos), con el detalle de los productos que tiene contratados.

3.5 Interfaz de Supervisión

El Interfaz de Supervisión monitorizará todos los procesos (entrada, procesamiento, generación de ficheros y difusión), analizará las incidencias generadas en cada instancia y actuará para solventarlas.

Todas las incidencias quedarán registradas en el sistema, tanto las detectadas por el interfaz, por Supervisión o por el cliente a través de los medios puestos a su disposición (hotline, e-mail, etc).

Los operadores de la solución tendrán acceso a las incidencias y a su estatus (abierta, cerrada, en curso, etc).

Para poder solventar las incidencias, el administrador del interfaz de supervisión tendrá acceso:

- A las BBDD donde se almacene la información.
- A los procesos implementados
    - Recepción de archivos
    - Procesamiento de la información
    - Enriquecimiento de la información
    - Creación de ficheros
    - File dispatcher
    - Almacenamiento de ficheros

El interfaz contemplará, al menos, las siguientes alarmas:

Entrada de la información

- Inicio del proceso de recogida de la información.
- Proceso de recepción de la información.
- Errores en la recepción de la información:
    - Archivo no disponible,
    - Archivo no descargable,
    - Error de conexión,
    - Otros problemas.

- Recogida de la información correcta.
- Inicio del proceso de recogida de actualizaciones.
- Error en base de datos.
- Error en SFTP/RSS.
- Caída de las comunicaciones.
- HHRR: no hay HHRR en >60-120 minutos dentro del periodo de recepción habitual establecido.
- HHRR: error en documento adjunto.

Se debería crear una tipificación de errores para poder contactar a la fuente y agilizar los trámites necesarios para que la fuente corrija dichos errores.

Procesamiento de la información

- Error en carga de datos, detallando el tipo de error:
  - Campos vacíos ,
  - Campos con datos no válidos (e.g. formato incorrecto/no válido).,
- Error en los procesos que estandarizan la información (pe. error al cambiar códigos locales por estándares internacionales) .
- Error en los procesos que “enriquecen la información” (e.g. al añadir timestamps, o LEIs, CFI/MIC codes).
- Error de base de datos.
- HHRR: error de OCR
  - OCR inactivo,
  - Incompatibilidad del archivo para su lectura,
  - Error en la lectura del archivo.
- HHRR: error de lector de texto
  - Proceso inactivo,
  - Proceso incapaz de extraer información.

Comunicaciones

- Error en las comunicaciones (entrega y recepción de la información) .
- Error en comunicación con los distintos sites o repositorios.

- Errores en las comunicaciones.

Generación de ficheros

- Errores en la generación del fichero:
  - Por falta de información,
  - Por problemas en los procesos (detallar proceso fallido).
- Error en base de datos

Difusión y entrega

- Error en base de datos de clientes (asignación imposible).
- Errores en la entrega de la información:
  - SFTP:
    - carga fallida en carpeta del cliente,
    - asignación incorrecta de productos en carpetas SFTP.
  - E-mail:
    - E-mail no generado,
    - E-mail no enviado,
    - E-mail devuelto.
- Repositorio de HHRR.
- Otros.

Generación de alertas para el cliente

- Archivo disponible.
- Archivo actualizado disponible.
- Retraso en la entrega del archivo.

El sistema almacenará un log con las actividades realizadas, y clasificará las incidencias según su estatus actual:

- Incidencia abierta,
- Incidencia cerrada.

Se establecerá un protocolo para remitir a la Central las incidencias de segundo nivel que no puedan ser atendidas por los operadores del servicio (Nivel 1). Asimismo, se redactará un SLA que cubra las incidencias que tenga el sistema.

3.6 Almacenamiento

En cada paso de los descritos con anterioridad se deberá almacenar la siguiente información:

1. Recepción de la información
    a. Maestro de Valores y Precios y Volúmenes: se deberán almacenar los ficheros recibidos con su formato y contenido originales (incluyendo actualizaciones si fuese el caso). No se reescribirán los ficheros, de tal forma que se pueda auditar la información recibida de cada fuente en caso necesario.
    b. Hechos Relevantes: se deberán almacenar tanto las notificaciones recibidas como los documentos adjuntos a las mismas con su formato y contenido originales.
    c. Incidencias: se deberán almacenar todas las incidencias generadas en los procesos de esta etapa.

2. Procesamiento (opcional):
    a. Maestro de Valores y Precios y Volúmenes: una vez se depure y enriquezca la información, ésta se almacenará para su posterior tratamiento.
    b. Hechos Relevantes:
        i. Procesamiento inicial: se almacenará la información resultante del procesamiento inicial
        ii. Procesamiento texto / OCR: para los Hechos Relevantes cuyo contenido sea procesado por herramientas de tratamiento de texto, se almacenará tanto el contenido del Hecho Relevante en formato txt como la información extraída para su inclusión en los ficheros *machine readable*.
    c. Enriquecimiento de información: la información que se deberá almacenar para enriquecer la información recibida de las fuentes será, al menos, la siguiente:
        i. NIF o equivalente de las emisoras
        ii. LEI de las emisoras (ISO 17442)
        iii. MIC del mercado (ISO 10383)
        iv. Códigos de monedas (ISO 4217)
        v. Códigos ISIN de las emisiones (ISO 6166)
        vi. Códigos del país (ISO 3166)
        vii. Horarios de negociación del mercado

   d. Incidencias: se deberán almacenar todas las incidencias generadas en los procesos de todas estas etapas.

3. Interfaz de Gestión de Output:
   a. Gestión de ficheros: se almacenará toda la información de los ficheros generados.
   b. Gestión de clientes: se almacenará toda la información de los clientes, tanto la referente a los datos de la entidad como la información sobre los productos contratados.
4. Interfaz de Supervisión:
   a. Recepción: se almacenarán las incidencias generadas.
   b. Procesamiento: se almacenarán las incidencias generadas.
   c. Difusión: se almacenarán las incidencias generadas.

Los ficheros generados se almacenarán según la estructura que decida el equipo técnico.

Toda la información se deberá mantener por un mínimo de 5 años. No obstante, no se borrará información histórica de Hechos Relevantes en formato procesable ni de Precios y Volúmenes, para atender posibles peticiones de información histórica.

Capítulo 4. Descripción de la solución web

En este capítulo se describe la funcionalidad de la aplicación web que servirá tanto para las mesas de operaciones como para los usuarios de negocio, donde se ofrece desde un Dashboard hasta una completa gestión de servicios para la gestión de clientes, productos, canales, puntos de acceso, configuración de ficheros, hechos relevantes, instrumentos financieros, usuarios y roles, definición de procesos y parámetros, así como control de actividad y reprocesos.

4.1 Servicio Web de Operación

Este servicio representa la web de operación, usada para tareas de mantenimiento de maestros y gestión de incidencias. Cuenta con las siguientes secciones y funcionalidades:

Sección	Descripción
Dashboard	En esta sección se muestran los detalles de los ficheros de entrada y de salida para una sesión específica. Debe permitir agrupaciones, esto es, por tipos de buzón o por mercados.
Clientes → Subscripciones	Desde aquí se puede ver los clientes y realizar las siguientes acciones: Habilitar / Deshabilitar el cliente Modificar el Cliente y sus suscripciones Ver un detalle del cliente Borrar el cliente Crear un cliente
Clientes → Estado de envíos	Permite ver todos los envíos realizados hacia los clientes con la posibilidad. Desde la lista se puede re-enviar un fichero a un cliente o ver un detalle del envío.

Monitorización actividad → Control de Actividad	Muestra toda la actividad del sistema emitida por todos los procesos y servicios. Desde aquí se puede navegar en algunos casos hacia los ficheros asociados a la actividad.
Monitorización actividad → Ficheros entrada / salida	En esta sección se pueden ver todos los ficheros planificados. Los ficheros pueden estar en distintos estados y se puede ver un detalle de cada fichero e inclusive re-procesarlo o descargarlo.
Monitorización actividad → Servicios del Sistema	Desde aquí se pueden gestionar los procesos y descargar sus respectivos LOGS (registros).
Mantenimiento CADATA → Hechos Relevantes	Muestra los hechos relevantes y permite corregir a los que han dado error de carga.
Configuración del sistema → Canales	En esta sección se listan los canales del sistema y permite realizar las siguientes acciones sobre los mismos: Modificar un canal Ver el detalle de un canal Borrar un canal Crear un canal
Configuración del sistema → Puntos de Acceso	Se listan todos los puntos de acceso del sistema hacia otras fuentes externas y permite realizar las siguientes acciones: Modificar un punto de acceso y probar la conexión hacia dicho punto Ver detalle del punto de acceso Eliminar un punto de acceso

	Crear un punto de acceso
Configuración del sistema → Configuración Ficheros	En esta sección aparecen las configuraciones de ficheros del sistema, todos los ficheros de entrada y de salida. Desde aquí se pueden realizar las siguientes acciones: Modificar una configuración y planificar su ejecución Ver detalle de la configuración Borrar la configuración Crear una nueva configuración
Mantenimiento → Festivos	Aquí se encuentran los días festivos de cada país del sistema, se pueden realizar las siguientes acciones: Modificar un día festivo Ver en detalle un día festivo Eliminar un día festivo Crear un día festivo
Mantenimiento → Códigos MIC	Se listan todos los mercados de los diferentes países y se pueden realizar las siguientes acciones: Modificar un mercado Ver detalle de un mercado Borrar un mercado Crear un nuevo mercado.
Mantenimiento → Instrumentos Financieros	Se muestran todos los instrumentos financieros cargados en el sistema. Se pueden realizar las siguientes acciones: Modificar un instrumento financiero del canal

	Ver el detalle de un instrumento financiero Crear un nuevo instrumento financiero
Mantenimiento → Tipos Hechos Relevantes	Se muestran los tipos de hechos relevantes los cuales se asocian a los distintos hechos relevantes, se pueden realizar las siguientes acciones: Modificar un tipo de Hecho Relevante Ver en detalle un tipo de Hecho Relevante Crear un nuevo tipo de Hecho Relevante Eliminar un tipo de Hecho Relevante
Mantenimiento → Activos Subyacentes	Se listan todos los activos subyacentes en el sistema. También se pueden realizar las siguientes acciones: Modificar un activo subyacente Ver en detalle un activo subyacente Borrar un activo subyacente Crear un activo subyacente
Usuarios → Gestión de Usuarios	Desde esta sección se pueden gestionar todos los usuarios del sistema, se pueden realizar las siguientes acciones: Cambiar el rol del usuario Habilitar / Deshabilitar el usuario Modificar el usuario Ver un detalle del usuario Borrar el usuario Crear un nuevo usuario

Usuarios → Gestión de roles / secciones	En esta sección se pueden gestionar los roles y las secciones de la web, desde aquí se identifica las distintas secciones a las que cada rol puede acceder o no.
Procesos / Parámetros → Definición de Procesos	Se muestran los procesos en el sistema. Se recomienda que solamente el área técnica utilice esta sección.
Procesos / Parámetros → Definición de Parámetros	Se muestran los parámetros asociados a los procesos del sistema. Se recomienda que solamente el área técnica utilice esta sección.

Capítulo 5. Flujo de procesos

En esta sección se describirá los distintos componentes necesarios para completar un ciclo de ejecución desde el inicio al final del proceso.

En términos generales se describe el proceso completo desde la recepción de un fichero hasta su distribución, donde se incluye la actualización del estado en BBDD y la comunicación asíncrona hacia el siguiente 'servicio' perteneciente al flujo del proceso.

La decisión de establecer una **comunicación asíncrona** es debida a la flexibilidad que aporta tener la ejecución 'troceada', de manera que un error en la cadena de ejecución deja el rastreo de ésta y su recuperación desde la web de operación desde el último mensaje de notificación enviado.

5.1 Recepción

Existen 2 estrategias de recuperación de ficheros desde las fuentes de contribución:

- Recuperación de ficheros planificados al inicio de cada sesión.
- Recuperación continua de datos.
- Recuperación de los datos tras algún tipo de evento externo:
    - Disparado por otra aplicación.
    - Por tarea programada.

Recepción SFTP Planificada.

Durante la sesión, las aperturas y/o cierres de los diferentes mercados, se conoce previamente, a qué hora aproximada el contribuidor va a generar un determinado tipo de fichero que necesitamos recuperar. Además, se conoce de antemano el nombre que la fuente le asignará al fichero, dado que este nombre se compone de un literal de identificación de los datos y una cadena que identifica la sesión (ejemplo fecha Derivados_yyyymmdd.csv → Derivados_{session}.csv).

Recepción datos.

Sin una planificación específica, no se conoce con seguridad a qué hora o cuantas veces al día se genera la información y durante todo el día. Generalmente, en estos casos se trata de recuperar ficheros de manera continuada, o bien obtener los datos por medio de una API o leyendo una página web y ver si se ha actualizado, o bien recibirlos tras un evento de entrada asociado.

5.2 Carga & Normalización (File Importer)

En caso de procesamiento de la información, debido a la heterogeneidad de las fuentes de datos, este proceso es el encargado de ***normalizar datos*** en primera instancia y posteriormente realizar la carga en la base de datos de los registros.

La ejecución se inicia al recibir una notificación del **monitor de importación de ficheros**. Una vez descargada la información en la etapa de recepción, y comprobar que el fichero en el sistema se encuentra en el estado de ‘Recibido’ y "Pdte. de Carga", se realiza la notificación al microservicio oportuno para la carga normalizada en base de datos del contenido de fichero.

Proceso de carga

La notificación del evento dispara este microservicio que contiene la inteligencia necesaria para comenzar la carga del fichero en el sistema.

En caso de no encontrar el fichero, el sistema indica que el fichero no está disponible y actualiza su estado, emitiendo un mensaje al log y **salvaguardando el mensaje** para su reproceso desde la web.

Si el fichero ha sido descargado el sistema procederá a la carga.

Si la ejecución del programa de carga es correcta, el sistema mostrará en la web la cantidad de registros cargados y descartados.

Si la ejecución del programa es incorrecta, mostrará el estado de error en la pantalla adjuntando un detalle del error.

5.3 Generación de ficheros (File Exporter) normalizados

Este micro proceso recibirá un evento desde el **servicio de monitorización Extracción a ficheros**, y es el encargado de exportar datos normalizados a los ficheros que se distribuirán a los clientes finales.

Generará 3 tipos de ficheros, XML, CSV y JSON, en el caso de los ficheros cuayo contenido ha sido normalizado, para los que no lo están, se trata cualquier tipo de fichero que se presente en la entrada. Y completará su ejecución

Proceso de extracción

Una vez llegado el evento, el sistema recopila los datos de las diferentes tablas de la Base de Datos y ejecuta el proceso de extracción.

Si la ejecución del proceso de extracción es correcta, el sistema mostrará en la pantalla la cantidad de registros cargados y pondrá el fichero en estado pendiente de distribución

Si la ejecución del proceso es incorrecta, mostrará el estado de error en la pantalla adjuntando un detalle del error.

5.4 Generación de ficheros (File Exporter)

Este micro proceso recibirá un evento desde el **servicio de monitorización Extracción a ficheros**, y es el encargado de exportar los ficheros recibidos, que se distribuirán a los clientes finales.

Generará ficheros de cualquier tipo que se encuentre en las entradas Y completará su ejecución

Proceso de extracción

Una vez llegado el evento, el sistema recopila los ficheros y ejecuta el proceso de extracción.

Si la ejecución del proceso de extracción es correcta, el sistema mostrará en la pantalla la cantidad de registros cargados y pondrá el fichero en estado pendiente de distribución

Si la ejecución del proceso es incorrecta, mostrará el estado de error en la pantalla adjuntando un detalle del error.

5.5 Distribución de ficheros

El **servicio de monitorización de distribución** se encarga de la verificación continua de ficheros en estado Pendiente de Distribución para poder distribuirlos.

Hay algunos procesos que no requieren previamente pasar por las etapas de enriquecimiento y transformación de la información, si no que proceden de fuentes externas que inician el proceso o el propio servicio de monitorización lanza el evento hacia el microservicio para empezar la difusión punto a punto de los mismos.

Al completarse la generación de ficheros (exportación de datos), el fichero pasa a estado `***Pdte. de distribución'***, en caso de no tratarse la generación de los mismos, estos ficheros presentan un estado de inicio de '***Pdte. de distribución'***. Al completarse la distribución a todos los clientes, el estado se modifica a ***'distribuido'***.

El micro proceso **Distribución a clientes** es despertado desde la monitorización del servicio de Distribución, para realizar los envíos de cada una de las peticiones en estado ***'pendiente de envío'***, a través de un sistema de suscripciones de los clientes a los tipos de archivos; éste verifica a qué clientes debe enviar el archivo y se lo envía al punto de acceso configurado para cada cliente.

Si fallan los intentos de envió, el estado pasará a ser de error, y el operador puede solicitar que se reinicie el envío desde la web de gestión (pasando nuevamente status='Pdte. envio', reiniciando la cantidad de errores a 0).

El envío puede ser vía diferentes protocolos de transmisión, por ejemplo, ***sftp o email***, por indicar algunos de ellos, configurados en BBDD.

Por último, el **servicio de monitorización de distribución** envía eventos al **micro proceso de Backup y mantenimiento** general de los repositorios de los ficheros distribuidos como el borrado recurrente de la información para evitar saturar el sistema (por ejemplo, borrar archivos con una antigüedad superior a los 20 días).

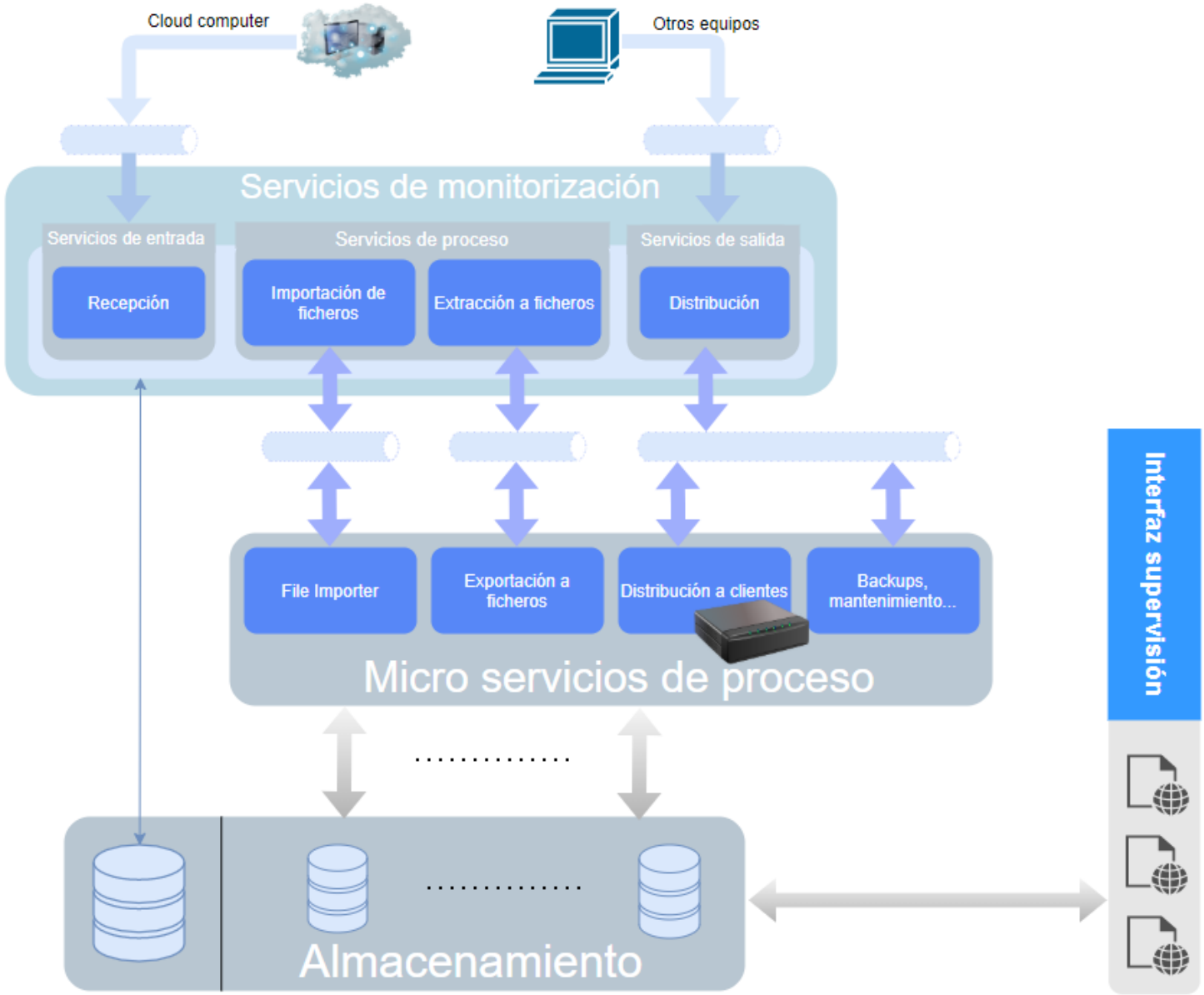

Figura 6. Diagrama de servicios y flujo de procesos

Capítulo 6. Modelo de estados

En el apartado anterior se detalló las etapas de 'ejecución' de la solución. En esta sección se describirá el modelo de estados que sigue el flujo de ejecución de la solución en términos de máquina de estados.

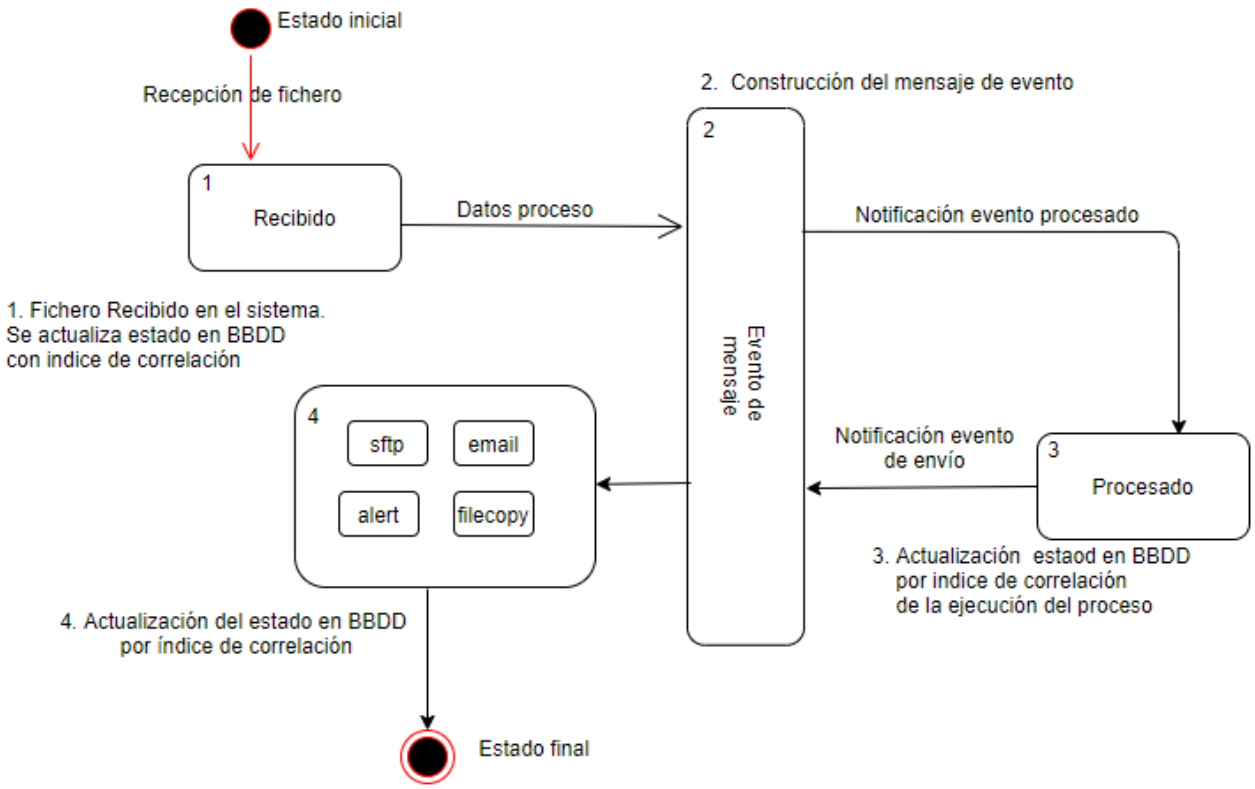

Figura 7. Modelo de estados del flujo de ejecución.

Arreglo al diagrama de servicios y flujo de procesos planteado anteriormente, se describe el diagrama de estados por cada una de las etapas o fases por la que pasa la información en el sistema. Están son:

- Servicios de entrada: Recepción de ficheros
- Servicios de proceso:
  - Carga de ficheros
  - Generación de los ficheros
- Servicios de salida: Distribución de ficheros

6.1 Recepción de ficheros

Diagrama de estados de recepción del fichero en el sistema.

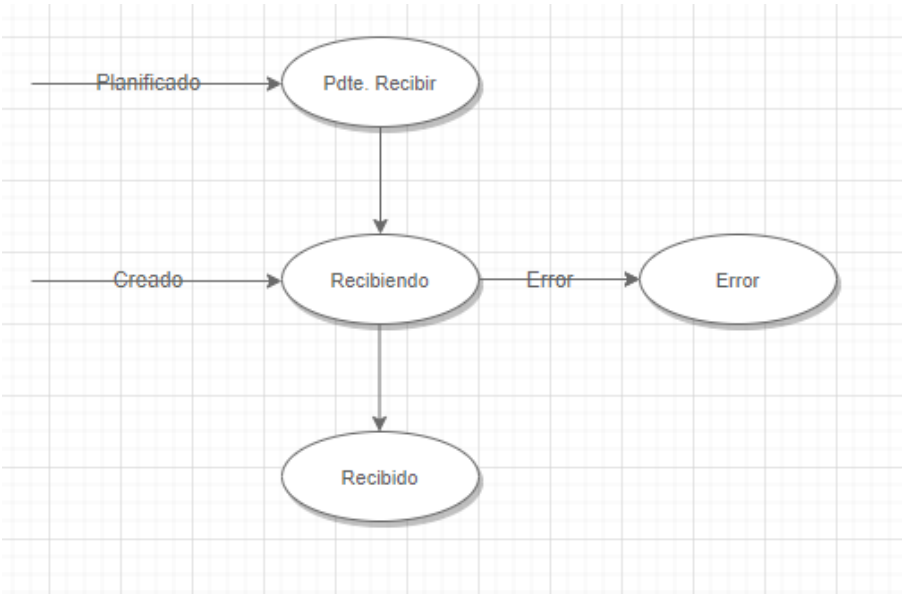

Figura 8. Diagrama de estados de recepción del fichero de entrada

6.2 Carga de ficheros

Diagrama de estados de carga del fichero en el sistema.

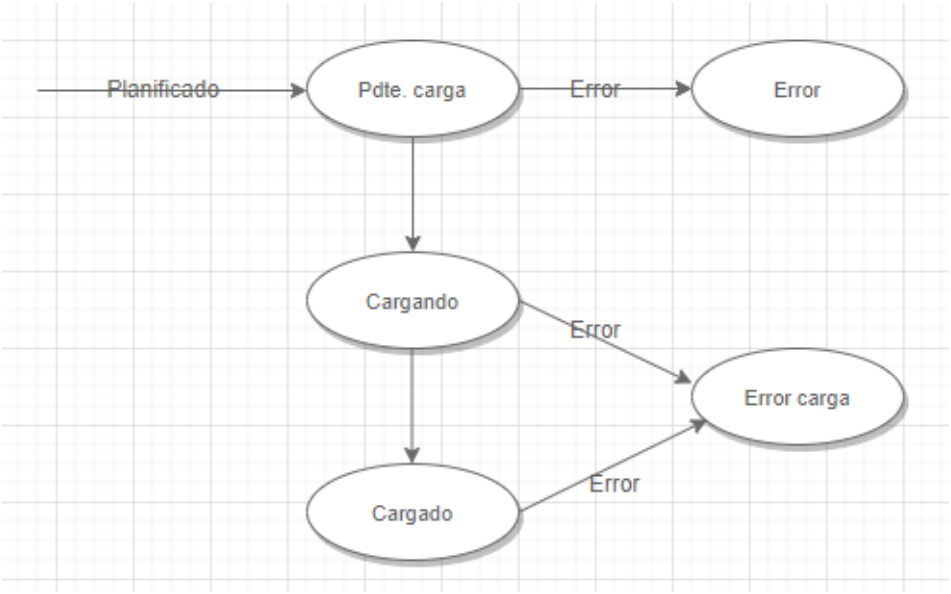

Figura 9. Diagrama de estados de carga de fichero de entrada

Estos dos diagramas de estado anteriores describen la situación ante un fichero de entrada que o bien está planificada su hora de recuperación o se reintenta hasta que se obtiene. Posteriormente, pasa a ser cargado en el sistema. Estos estados indican que un proceso se ha hecho cargo del fichero y éste se está cargando en el sistema.

6.3 Generación de ficheros

Diagrama de estados de extracción del fichero de salida transformado en el sistema.

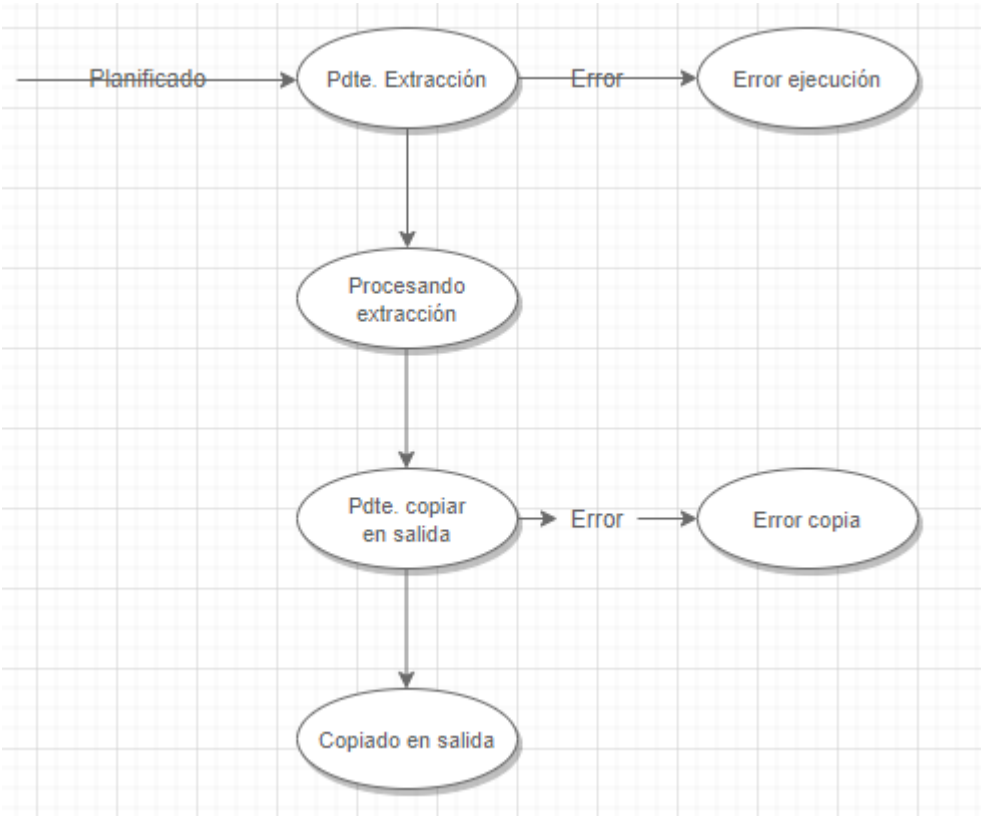

Figura 10. Diagrama de estados de extracción de ficheros de salida

Este diagrama de estados describe la situación ante un fichero dado de alta en el sistema, que bien de manera planificada o de forma recurrente se procesa la extracción a través de la cual se genera la salida posterior. Posteriormente, pasa a quedar como pendiente de ser distribuido.

6.4 Distribución de ficheros

Diagrama de estados de distribución del fichero de salida a los clientes.

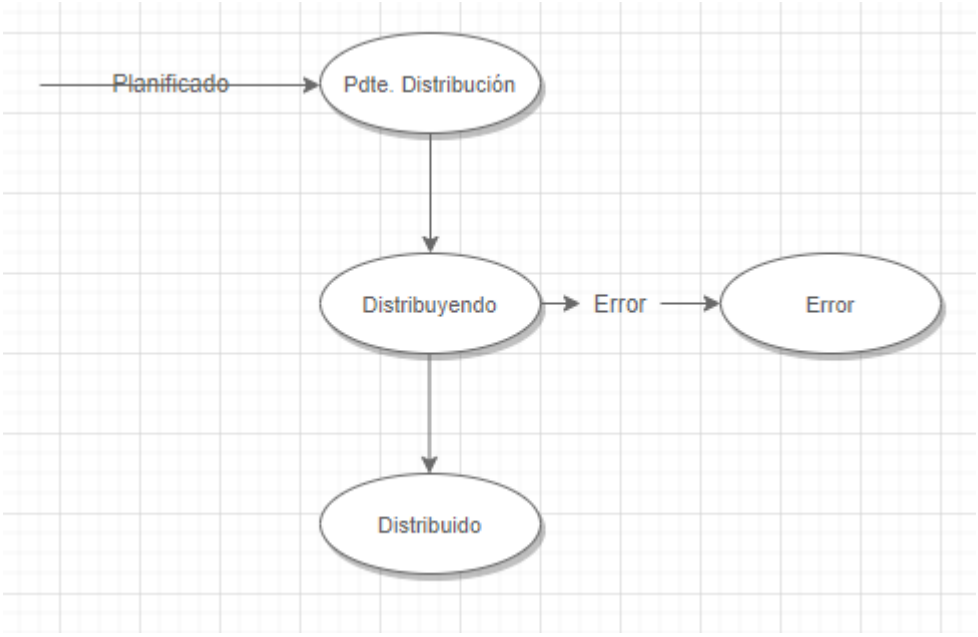

Figura 11. Diagrama de estados de distribución de ficheros

Este diagrama de estados describe la situación de un fichero generado y preparado para ser distribuido a los clientes, que de manera planificada se procesa el envío por medio de configuración en el sistema. Posteriormente, pasa a quedar como distribuido.

Capítulo 7. Enfoques de diseño

En las secciones anteriores ya se ha mostrado algún indicio de la solución arquitectónica a plantear representando en las figuras asociadas de algunos apartados un bus de eventos.

Una vez completada la definición de la solución en bloques funcionales, procesos y modelo de estados, en esta sección se propone el ***diseño arquitectónico orientado a microservicios y dirigidos por eventos*** para el desarrollo de extracción, transformación, carga y distribución de la información.

El objetivo principal es proporcionar un ***servicio integral arquitectónico para la difusión de información financiera al usuario final***. En particular, se usan distintas técnicas para la captación de información. Desde extracción en las webs tradicionales, con distintas técnicas de recuperación de la información tales como ***crawlers*** o ***scrapers*** (ver anexo); sitios sftp, o bien captura de datos de almacenes de datos clásicos relacionales a través de mecanismos de **file dispatcher**.

Este trabajo supone unos retos adicionales entre los cuales destacan [7]:

- Migración de aplicaciones monolíticas a microservicios.
- Desarrollo desde cero de otras soluciones a integrar.
- Integración de las soluciones mixtas.
- Paradigma de gestión: gobernanza descentralizada en microservicios.
- Onboarding del sistema.

La solución constará de estos tipos de componentes:

- ***Componentes de presentación***. Estos componentes son los responsables del control de la interfaz de usuario y el consumo de servicios remotos.
- ***Lógica de integración de aplicaciones***. Este componente incluye un canal de mensajería basado en agentes de mensajes.
- ***Lógica de dominio o de negocios***. Este componente es la lógica de dominio de la aplicación.
- ***Lógica de acceso a bases de datos***. Este componente está formado por componentes de acceso a datos responsables de acceder a las bases de datos (SQL o NoSQL).

Basándonos en reglas de la arquitectura de microservicios, y arreglo al objetivo de este trabajo de desarrollar una evolución de aplicaciones monolíticas a un estilo arquitectural basado en microservicios, varios son los factores que 'empujan' a decantarse por este estilo arquitectural:

- Desde el **punto de vista arquitectural**, este estilo es útil para la generación de un conjunto de servicios pequeños. Cada microservicio implementa una capacidad empresarial en un contexto de dominio limitado, dentro de un dominio global, la difusión de información de mercado. Este principio rector de los microservicios es similar al diseño guiado por el dominio, por sus siglas DDD.
- Cada uno de ellos se **desarrolla y ejecuta** de manera independiente a los demás. Es decir, ocurre una separación, de modo que cada microservicio posee su modelo de datos y su lógica de negocio.
- En el **contexto de los equipos de desarrollos**, podría haber razones suficientes con:
    - El *mestizaje tecnológico*, permite diferentes habilidades en distintas tecnologías, o bien, necesidad de incorporar desarrollos rápidos, market in time.
    - Cambios frecuentes en los requerimientos funcionales, y previsión de migraciones parciales, que permitan aprovechar el desarrollo tecnológico altamente cambiante.
    - Inclusión progresiva de integración continua y metodologías ágiles de desarrollo del software.
- Desde el **punto de vista del escalado**, estos sistemas pueden crecer horizontalmente de manera independiente. De esta forma, se puede escalar a medida cada servicio en función de su demanda, independiente del resto.
- Desde el **punto de vista tecnológico**, los microservicios no requieren de ninguna tecnología específica.
- La necesidad de integrar aplicativos de la naturaleza anteriormente señalada con otros de naturaleza más amplia en su conjunto, pero que obedecen a la misma práctica. Subsistemas integrados en un sistema mayor, que funcionan como componentes independientes.
- Un último aspecto a tener en cuenta es que la **arquitectura lógica** puede coincidir o no con la **arquitectura física**. Es decir, la arquitectura lógica y los límites lógicos de

un sistema no se asignan necesariamente uno a uno a la arquitectura física o de implementación. Esto puede suceder, pero a menudo no es así.

7.1 Arquitectura de comunicación

Los microservicios se comunican entre sí mediante protocolo HTTP (REST), pero también pueden hacerlo de manera asíncrona (mediante AMPG[9], por ejemplo).

Existen dos tipos de comunicación:

Comunicación de cliente a microservicio de HTTP directa Este enfoque se sigue para las consultas y actualizaciones desde el OnBoarding, una herramienta operativa de gestión y control interno de la operativa asociada al back end, y que responderá a un punto de entrada único por microservicio.

Comunicación asincrónica basada en eventos. Esta comunicación se realiza mediante un bus de eventos para propagar las actualizaciones en los microservicios o para la integración con aplicaciones externas.

En un entorno de producción es necesario proteger a los microservicios. Este enfoque de comunicación asíncrona permite aislar a estos de los servicios que recogen la información de puntos externos de la empresa, ambas dos se usarán en la arquitectura que se presenta.

El **bus de eventos** se puede implementar con cualquier tecnología de infraestructura de agente de mensajería como **RabbitMQ**, o bien mediante **Service Bus de nivel superior** (nivel de abstracción) como Azure Service Bus, NServiceBus, MassTransit o Brighter.

En este caso, las necesidades de rendimiento y funcionales de este componente se ajustan a las especificaciones de **RabbitMQ** [10].

Algunas de las principales características y funcionalidades de RabbitMQ son:

- Software **Open Source**.
- **Gestión de mensajería síncrona o asíncrona**.
- **Gestión asíncrona de mensajes**. Soporta protocolos diversos, colas de mensajes, enrutado flexible a colas, diversos tipos de intercambio de mensajes, etc.
    - ***Protocolos soportados*** por RabbitMQ son AMQP, STOMP, MQTT, HTTP y websockets
    - ***Tipos de intercambio de mensajes***:
        - Intercambio directo e individual por topic.
        - Todos los consumidores conectados a la cola reciben el mensaje.
        - Cada consumidor recibe un mensaje enviado.
- **Se adapta a diversos lenguajes y tecnologías,** los lenguajes más conocidos como Java, .NET, PHP, Python, GO, Ruby, JavaScript, etc.
- **Está preparado para su implementación en sistemas distribuidos**. Puede funcionar en un clúster de servidores para la más alta disponibilidad.
- **Monitorización avanzada**, mediante una interfaz HTTP, la línea de comandos y aplicaciones de interfaz gráfica.
- **Durabilidad y persistencia de los mensajes.**
- Además, **es capaz de extenderse mediante la instalación de plugins diversos**.
- se adapta muy bien a los entornos Cloud, tanto públicas como privadas.

7.2 Componente de presentación

Estas se desarrollarán a través de interfaces de usuario basadas en microservicios de cada microservicio.

Es común que la arquitectura de microservicios se inicie con la lógica del negocio, al final estos suelen implementar necesidades del Back-end. Para ello la interfaz del usuario se controla todavía como si respondiera a un monolito ‘combinado’.

Par el propósito que nos ocupa podríamos tener una interfaz única, común, que a través de accesos por usuario nos defina qué vistas son las que tiene el operador en concreto, y que compartan interfaz; o podemos elegir un enfoque más avanzado, llamado **micro front-end** [11], que consiste en diseñar la interfaz de usuario en función de estos microservicios.

Un enfoque de interfaz de usuario compuesta controlada por microservicios es más compleja de diseñar y desarrollar, y también depende su implementación de si se trata de diseñar una web tradicional o sobre un dispositivo móvil o una SPA (aplicación de página única) al estilo del framework Angular o similares.

Es una decisión que escapa al ámbito de este trabajo, pero que requiere un tiempo de reflexión elegir la más adecuada atendiendo a los diversos factores que intervienen.

Capítulo 8. Detalles del estilo arquitectural

Las ventajas que aporta la migración a una **solución basada en microservicios** son las de **mantener los problemas aislados** en aquellos servicios que ahora aportan valor a la empresa.

La principal característica representativa en la elección de esta arquitectura ha sido la migración de varios aplicativos monolíticos hacia microservicios, integrando estos a través de una capa de comunicación asíncrona y el desarrollo para una gestión integral de un OnBoarding de operaciones.

En la sección de estado del arte se presentan las ventajas generales de la adopción de este tipo de estilo arquitectural, pero conviene en este punto recordar aquellas que sí son aplicables en esta propuesta [3]. Estas son:

- Cada aplicación monolítica migrada a microservicio puede crecer y escalar independientemente.
- Cada servicio (en forma de microservicio) proporciona un límite firme
- Cada solución se mantiene escrita en su lenguaje original.
- Se mantiene la gestión de gobierno descentralizada original.
- Y hay un aspecto destacable, que mientras en una arquitectura monolítica con el paso del tiempo la estructura modular puede llegar a ser difícil de mantener, en el estilo arquitectural de los microservicios, esto se mitiga.
- Existe una reutilización de facto, como son los componentes de software estructurales, que comparte cada microservicio.
- Interfaces de componentes más explícita.

Defintivamente, nos permite **integrar subsistemas que en definitiva son servicios organizados,** como algunos **CRUD**, simples de dos capas, o **subdividir una solución más compleja de 3 capas basada en .Net Core**, manteniendo la independencia ejecutiva y operativa. Es decir, cada equipo puede seguir manteniendo sus sistemas, y un error en la ejecución de un subsistema no afecta al resto. Por lo que se asegura la actual situación de gestión de servicios.

Incluso el despliegue en tiempo de ejecución es posible sin necesidad de detener el sistema completo.

Al tiempo se mantienen los equipos de desarrollo originales, con una mínima inversión en la actualización de un sistema amplio y específico funcionalmente.

Entre las **desventajas** teóricas potencialmente presentes en un estilo de arquitectura de microservicios estarían las ya conocidas: latencia de la comunicación, la comunicación directa entre servicios, la atomicidad de operaciones contra la BBDD, el escalado con requerimientos más elevados de hardware, y por supuesto, la migración en sí de una arquitectura monolítica a un estilo arquitectural de microservicios, entre otros.

Bien, en el análisis preliminar a estas valoraciones se aprecia rápidamente que una solución distribuida aporta complejidad al sistema, y que habrá algún equipo que tendrá que liderar dicha implementación abordando las cuestiones ya citadas de comunicación entre microservicios, atomicidad en las operaciones de BBDD y comunicación asincrónica basada en eventos.

El resto de los factores son despreciables en esta solución y no impactan en el modelo de ejecución, puesto que el cuello de botella en este aplicativo está en el procesamiento de unidades de información relativamente pesadas, y no la comunicación entre componentes.

Es decir, los mecanismos de comunicación, que aportarían esa supuesta latencia al sistema son despreciables en comparación con los tiempos de ejecución de core del aplicativo. Por lo que no es un dato relevante de rendimiento. Los requerimientos iniciales tampoco exigen una respuesta inmediata del proceso ETL.

Debido a determinados puntos de partida, coincidentes con el estilo arquitectural de microservicios, como son que los sistemas monolíticos o la solución ETL, imponen una predominante orientación a los datos y que de origen cada solución tiene su propio modelo de datos (gestión de datos descentralizada); y que incluso diferentes equipos de desarrollo con diferentes velocidades y capacidades tienen el control de sus piezas de código, la evolución del sistema a este modelo arquitectural mantiene estas características.

Quedaría pendiente abordar, llegado el momento de una mayor madurez, las cuestiones más complejas relativas a la comunicación directa entre servicios y la atomicidad de operaciones contra la BBDD; y que inicialmente al menos, cada microservicio contenga los

accesos a BBDD dentro de su unidad de ejecución, a la espera de un análisis más detallado y concreto de cómo abordar dichas complejidades.

Es por ello por lo que no existen transacciones atómicas entre microservicios, queda descartada esta opción, al menos en la primera fase de evolución.

Tampoco se trata de una aplicación con varias aplicaciones clientes, donde sería necesario considerar las puertas de enlace de API en lugar de la comunicación directa de cliente a microservicio expuesta anteriormente. Este es un patrón de servicio similar al patrón fachada del diseño orientado a objetos, pero en este caso forma parte de un sistema distribuido.

De una forma muy sintética y a modo sólo de exposición, este actúa como un proxy inverso, enrutando las solicitudes de los clientes a los servicios, pudiendo aportar también características transversales adicionales, como autenticación, terminación SSL y caché.

Finalmente, habrá un conjunto de microservicios muy especializados que sí desarrollarán una **comunicación entre ellos** y los demás de forma **asíncrona**, a través de eventos. Estos son los **microservicios de infraestructura**, tales como envío por sftp, envío de emails,o traceo, por citar algunos de ellos.

Aunque es cierto que el proceso confiable de estos componentes es crucial, no es menos cierto que no lo es su inmediatez. La entrega del evento y su procesamiento está garantizado a través de colas de mensajes persistentes, y la entrega o no del fichero, tampoco hará que el sistema varíe su ejecución hasta ese momento.

Por otro lado, sí es necesario notificar cualquier excepción que pudiera producirse que no permitiera la entrega de un fichero sftp, o un email. Es por ello que la transversalidad de los componentes de LOG (tanto en su formato de BBDD como de fichero) y la actualización del estado del proceso en sí mismo informarían de cualquier eventualidad para subsanar el error, guardando copia del mensaje a tratar, para poder ser reenviado desde la solución web del operador del sistema.

Ante la eventualidad de un corte de las comunicaciones, de forma que parte del proceso quede incoherente, la solución podría estar en la recuperación del último mensaje no procesado, a partir del cual se proseguiría el proceso.

Capítulo 9. Arquitectura Técnica

Actualmente la arquitectura técnica está basada en el **modelo de virtualización** para los servidores que van a procesar la información. Este modelo es típicamente asociado a **arquitecturas del software de tres capas**, donde el **servidor** es **alojado** en cada **Data Center** replicado de manera que se garantizan los estándares de seguridad, fiabilidad, disponibilidad y contingencia ante fallos. Estos Data Center garantizan los patrones de seguridad, máximo rendimiento y disponibilidad, en un ambiente fácilmente escalable. Los elementos técnicos típicos son un servidor (en caso de que exista replicación) y un servidor para BBDD, o todo en uno.

Desde hace algunos pocos años atrás ya aparecían entre las tendencias Event Driven Architectures y microservicios.

Del primero ya hemos dado habida cuenta de la justificación de su implementación a lo largo del trabajo, pero del segundo, a pesar de haber expuesto ya un amplio abanico de justificaciones a nivel de ingeniería de software quedaría por justificar el uso de microservicios en infraestructuras tecnológicas propias, o lo que es lo mismo, lo que el área de soporte de infraestructura ya se encarga de gestionar el despliegue en forma de dlls pero aplicado a este estilo arquitectural.

Es importante señalar que la **ejecución es interna**, es decir no hay un Cloud de por medio, de forma que la compañía expone los servicios dentro de un gobierno del dato y del proceso completo con la parte de gestión de la infraestructura.

Volviendo al principio y repasando la evidencia, es probado que el modelo de comunicación síncrona en el estilo de arquitectura de los microservicios no da respuesta adecuada y se muestra insuficiente.

Ante ese problema de elección de **arquitectura técnica** hay presentes dos tendencias actuales. Por un lado, en el mundo síncrono el término **service-mesh**, una infraestructura de software dedicada para manejar la comunicación entre microservicios habitualmente con herramientas de gestión, monitorización, etc., generalmente síncrona. Y que por el momento no sería aplicable a esta solución al no contar estos con interdependencia.

Y por otro lado el EDA, que lo haría con las soluciones de naturaleza asíncrona.

Dentro de esta última (el EDA) el concepto **Serverless**, conocido también como FaaS (Function as a Service) ha sido la que mejor ha representado este tipo de arquitectura.

Serverless, podría calificarse como 'sin servidor' y aunque carece de una definición formal, *sin servidor* también puede significar aplicaciones en las que el desarrollador de la aplicación aún escribe la lógica del lado del servidor, pero, **a diferencia de las arquitecturas tradicionales**, **se ejecuta en contenedores de cómputo sin estado que se activan por eventos**, son efímeros (solo pueden durar una invocación) y están completamente administrados por un tercero (en este caso soporte de infraestructura). Una forma de pensar en esto es "Funciones como servicio" o **"FaaS"**. [11]

Este trabajo se centra únicamente en **FaaS**, porque presenta grandes diferencias con respecto a lo que tenemos en mente cuando pensamos en las actuales arquitecturas técnicas.

FaaS está presente en los grandes proveedores de nube, como Google o Amazon desde hace ya algún tiempo.

Para mostrar gráficamente lo expuesto supongamos una arquitectura tradicional de tres niveles:

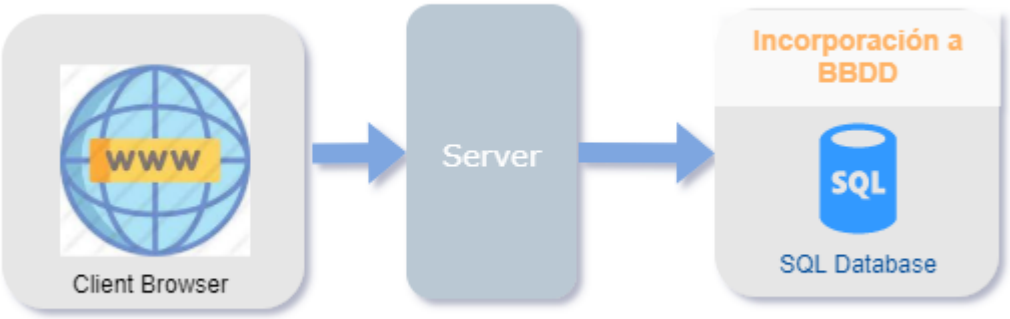

Figura 12. Arquitectura clásica de tres niveles

Con la arquitectura presentada anteriormente, el sistema (servidor) alberga toda la lógica del mismo, desde la autenticación hasta los procesos de negocio, y el cliente es algo poco o nada inteligente.

Supongamos ahora una arquitectura 'sin servidor':

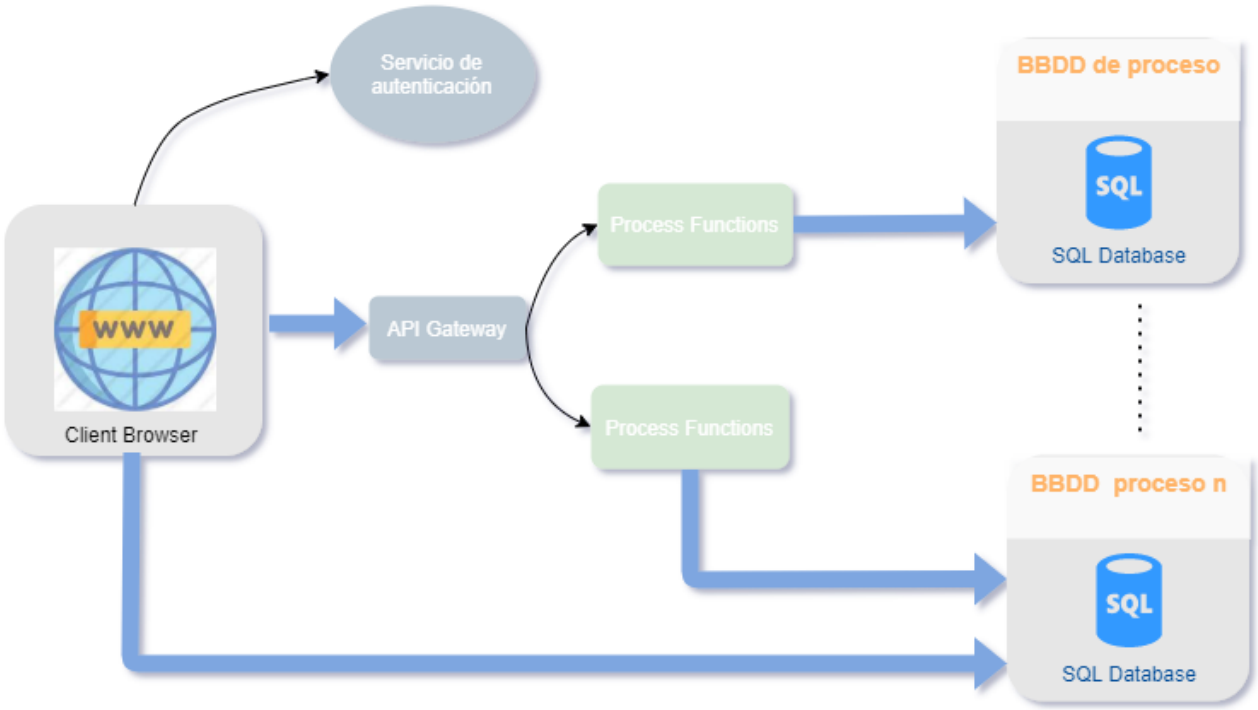

Figura 13. Arquitectura sin servidor.

Entre las principales diferencias entre las arquitectas destacan que, en la primera, el servidor central gestiona todo el flujo, control y la seguridad. Mientras que en la segunda no; se presenta una coreografía que orquesta a cada componente desempeñando un papel, una idea común en los microservicios.

Una de las principales ventajas de usar este tipo de arquitecturas es la flexibilidad ante el cambio, tanto en conjunto como en las actualizaciones independientes entre componentes.

Al final, **Faas** se trata de ejecutar código backend sin administrar sus propios sistemas de servidor o sus propias aplicaciones de servidor de larga duración; es decir, FaaS reemplaza el servidor de procesamiento de clics (posiblemente una máquina física, pero definitivamente una aplicación específica) con algo que no necesita un servidor aprovisionado, ni una aplicación que esté ejecutando todo el tiempo.

Este despliegue además es agnóstico de la tecnología empleada y cualquier microservicio puede ser desarrollado en cualquier lenguaje de uso extendido, así como la gestión de su consumo por parte del software de monitorización.

La implementación es muy diferente de los sistemas tradicionales, ya que no tenemos aplicaciones de servidor para ejecutar. En un entorno **FaaS**, cargamos el código de nuestra función al proveedor **FaaS**, y el proveedor hace todo lo necesario para aprovisionar recursos, instanciar VM, administrar procesos, etc.

Por lo que el **escalado** es **horizontal, automático y gestionado para** infraestructura para dar en cada momento la **necesidad de ejecución** que se necesita**.**

Es por ello que, aunque el paradigma cambia por completo, y la cultura habría de hacerlo, también los beneficios de coste hardware serían inmediatos, por el ajuste que se hace del hardware consumido por cada aplicativo en cada momento.

Sin embargo, los grandes proveedores no se quedan atrás en la fiesta de las herramientas de código abierto. La propia herramienta de implementación de AWS, SAM, el modelo de aplicación sin servidor, también es de código abierto .

Uno de los principales beneficios de FaaS propietario es no tener que preocuparse por la infraestructura informática subyacente (máquinas, VM, incluso contenedores). En el ámbito de este trabajo se toma como premisa tres aspectos. Los datos son propietarios, las necesidades de seguridad son elevadas y la existencia de un Data Center propio no es materia de discusión. O lo que es lo mismo en términos de arquitectura técnica. Es necesario ejecutar una plataforma propia FaaS “Serverful”.

Ha habido una buena cantidad de actividad en esta área. Uno de los líderes iniciales en FaaS de código abierto fue IBM (con OpenWhisk , ahora un proyecto de Apache) y - Microsoft, que abrió gran parte de su plataforma Azure Functions . Muchas otras implementaciones de FaaS autohospedadas hacen uso de una plataforma de contenedor subyacente, frecuentemente Kubernetes, lo que tiene mucho sentido por muchas razones. Hay otros proyectos interesantes como Galactic Fog, Fission y OpenFaaS. [14]

Capítulo 10. Patrones de diseño

Entrando en un aspecto algo más técnico en estas últimas secciones, en este trabajo se seguirán principios, buenas prácticas y la implantación de algunos patrones de diseño.

Principios y buenas prácticas:

SOLID se compone de una serie de **principios y de buenas prácticas** que se deberían tener como base antes de proponer una arquitectura de software para el desarrollo de nuestras aplicaciones.

Los principios SOLID nos permiten administrar la mayoría de los problemas de diseño de software, y si se siguen, se consigue desarrollar un código **más limpio, más mantenible, más escalable a futuro y menos propenso a errores**.

SOLID es un acrónimo, y cada una de las letras que lo componen tiene un significado:

- **S**: se refiere al **Principio de Responsabilidad Única** (Single Responsibility Principle), que en resumidas cuentas lo que dice es que cada módulo de software debe tener una única razón para cambiar.
- **O**: hace referencia al **Principio de Abierto/Cerrado** (Open/Closed Principle), que dice que el código debería estar abierto para extenderlo y para añadirle nuevas funcionalidades, pero en cambio debería estar cerrado a modificaciones, salvo las modificaciones que se deban realizar si se encuentra algún error.
- **L**: viene a referirse al **Principio de Sustitución de Liskov** (Liskov Substitution Principle), que en resumen lo que dice es que una clase derivada debe poder ser sustituida por su clase base.
- **I**: se refiere al **Principio de Segregación de Interfaces** (Interface Segregation Principle), que dice que se han de utilizar interfaces con propósito específicos, o sea que tengan responsabilidades únicas y que se piensen bien y no se hagan interfaces grandes.
- **D**: hace referencia al **Principio de Inversión de la Dependencia** (Dependency Inversion Principle), que para conseguirlo se hace uso de la inyección de dependencias.

Algunos patrones de diseño utilizados:

- De comportamiento
    - **Patrón de mensajería consumidor/suscriptor**.
- Creacional:
    - **Patrón Singleton**.
- Orientado a objetos:
    - **Inyección de dependencias**.

10.1 Patrón consumidor/suscriptor

Este patrón de diseño pertenece a la clasificación de ***patrón de comportamiento***. Este tipo de patrones tratan con algoritmos y la asignación de responsabilidades entre objetos.

En concreto, el patrón de diseño publicador/suscriptor es una variante del patrón Observador, de manera que el suscriptor (observador) se suscribe a un evento del publicados. Este, permite transmitir eventos de forma asíncrona entre objetos desacoplados, incluso a grupos de objetos consumidores.

El publicador notifica todos los suscriptores en el momento en que el primero dispara el evento al que están interesados. Además, como observador (suscriptor) es posible suscribirse a más de un evento.

Este modelo es el que implementa RabbitMQ.

10.2 Patrón Singleton

El patrón Singleton es un viejo conocido en la disciplina de la ingeniería de software. De manera muy breve es un patrón que permite asegurar que una clase solo tenga una instancia, a la vez que proporciona un punto de acceso global a esa instancia.

A pesar de la enorme controversia que genera este patrón. Llegando incluso a definirse como un anti patrón por una parte de la comunidad, lo cierto es que, en este caso, a la hora de aplicar el principio de inversión de dependencias, a través de **contenedores de Inversión de Control** y el **patrón de inyección de dependencias**, y dada la tecnología utilizada .NET core, así como la decisión de diseño de persistir los datos en cada microservicio durante el ciclo de vida del mismo de los objetos instanciados, aplica crearlos de manera Singleton. Es

decir, cada objeto instanciado en el contenedor persiste durante la vida de ejecución del microservicio, solo en el contexto del microservicio y para esos datos.

Esto unido a la gestión multihilo de los microservicios facilita el aislamiento entre de los distintos hilos ejecutados, cada uno con sus datos en el contexto del contenedor de Inversión de Control, que a todos los efectos se comporta como una factoría a la que le solicitamos objetos que internamente se encarga de instanciar y gestionar.

10.3 Inyección de dependencias

Este es el patrón de diseño de software usado en la programación orientada a objetos más utilizado para implementar el principio de inversión de dependencias.

Como cualquier patrón de diseño de software trata de solucionar de una manera elegante un problema recurrente. Como se ha visto en el aparrado anterior, la forma habitual de implementar este patrón es mediante el contenedor IoC, típicamente implementado por un framework.

Este patrón, como muchos otros, ayuda a separar el código por responsabilidades, siendo que en esta ocasión sólo se dedica a organizar el código que tiene que ver con la creación de los objetos.

Como ya sabemos, uno de los principios básicos de la programación, y de las buenas prácticas, es la separación del código por responsabilidades. Pues la inyección de dependencias parte de ahí.

En el código de una aplicación con OOP (Programación Orientada a Objetos) tenemos una posible separación del código en dos partes, una en la que creamos los objetos y otra en la que los usamos.

Existen patrones como las factorías que tratan esa parte, pero la inyección de dependencias va un poco más allá. Lo que dice es que los objetos nunca deben construir aquellos otros objetos que necesitan para funcionar. Esa parte de creación de los objetos se debe hacer en otro lugar diferente a la inicialización de un objeto.

Capítulo 11. Descentralización de gobierno

Este es un aspecto fundamental en este trabajo porque es uno de los objetivos marcados en la definición del alcance del documento. Desde las dos perspectivas de prestación de estos servicios:

- Por un lado, ya se venía de unos componentes aislados donde distintas mesas de operaciones desarrollaban las tareas de supervisión de los flujos de ejecución de estos procesos.
- Por otro lado, distintos equipos de desarrollo con distintos perfiles tecnológicos y profesionales se ocupaban cada uno de sus componentes concretos, de manera aislada.

La única incorporación a esta *reedición* de la gobernanza descentralizada se aplica a ese nuevo grupo de apoyo tecnológico, que ayudaría a definir las nuevas especificaciones para la conversión de los monolitos en la arquitectura a desplegar.

Es por ello, que se consiguen dos objetivos planteados. Unificar un conjunto de aplicativos estancos, transversalizando equipos de desarrollo,

- Unificación de algunos criterios de diseño:
  - Buenas prácticas comunes.
  - Aprendizaje conjunto de arquitecturas emergentes.
  - Cierta transversalización y colaboración de los equipos de desarrollo.

Y los equipos de supervisión de operaciones,

- Herramienta conjunta de supervisión con roles de acceso.
  - Con mayor capacidad de intervención sobre los procesos
    - Autonomía para recuperación de los procesos.
    - Formación de la solución web.

Con todo lo anterior, además se mantiene la gobernanza descentralizada, es decir, donde los aspectos clave de elección de lenguaje de programación, metodología de implementación, etc… las siguen tomando las personas que han venido desarrollando los distintos componentes.

Por otro lado, se asume también, como se indicaba en el punto anterior, la gestión de datos descentralizada, es decir, que cada microservicio se ocupa también de sus propios datos, pudiéndose tener de esa manera desde las distintas tecnologías de las que ya se venía, según el caso.

Y se mantiene la gestión de procesos con mejores niveles de comunicación efectiva, tales como a partir de la creación de ese grupo de trabajo de integración arquitectónico, donde se discute las soluciones a incorporar.

Y por último, aunque las mesas de operaciones sí sufren un cambio de herramienta para la supervisión de los servicios, esta ofrece mejoras significativas operativas respecto a las versiones actuales, como son las de ofrecer una solución web completa que facilita la gestión misma de la supervisión, y ésta a su vez, facilita la tarea de mantenimientos por parte de los equipos de desarrollo que ven una única herramienta que controlar para todo el dominio de aplicación.

Capítulo 12. Conclusiones

[13] Quizás nada sea más complicado y agravante para una organización de TI que lo que llamamos ***muerte por arquitectura***. Esta historia clásica ocurre con demasiada frecuencia. Se parten de supuestos puros, llevados al milímetro y con documentos 'blancos' de arquitectura tras meses de elaboración.

Pero no menos agravante es no tener a nadie al volante, y dejar en manos de cada cual la planificación, el diseño y la evolución de **las líneas de productos software** a desarrollar, donde se acaban produciendo artefactos muy similares con múltiples componentes repetidos, con las deficiencias que ello ocasiona tanto en el momento de la creación, con duplicidades; como en el momento del mantenimiento, con múltiples componentes similares que pueden cambiar en determinado momento, multiplicando la posibilidad de fallo.

En este trabajo se ha pretendido ofrecer una **integración de servicios maduros** bajo el paraguas de una propuesta arquitectural evolucionada respecto a la actual, ***coincidente con la Ingeniería de línea de productos software***, con la que se pretende alcanzar varios objetivos que se detallan a continuación:

- El desarrollo de una propuesta que la dirección acepte bajo premisas de gestión como el coste/rendimiento, oportunidad, Time-To-Market, etc. En definitiva, que demuestre que el fin último de **Ingeniería de línea de productos software** aplicado a este dominio genera beneficios a la empresa.
- Que sea aceptada por otros Stakeholders en conceptos como la satisfacción de los requisitos de madurez de gobernanza TI actuales o la gestión descentralizada.
- Y que los diferentes grupos de trabajo de IT acepten 'a bien' el esfuerzo que supone acoplarse a un cambio sustancial, principalmente cultural, donde un paradigma como este modifica '*las formas de hacer las cosas'*, e implica cambios orgánicos que retocan la fisonomía actual de los equipos de desarrollo.

En definitiva, una propuesta como la presente exige de que se produzcan cambios orgánicos para que se mejore la productividad y el coste en la producción de software. Y, por otro lado, abre el camino a la ***reutilización sistemática***, no oportunista, de la producción de este. Entre las ventajas de la reutilización sistémica se encuentran las siguientes, citadas anteriormente en los objetivos:

- Una **reducción de los costes** de desarrollo.

- Un **aumento de la calidad** de los productos.

- Un **aumento de la Productividad**, mediante la mejora de los tiempos en los que se desarrollan los nuevos proyectos informáticos (Time to Market).

- **Mejoras** en las actividades de **Mantenimiento y soporte de aplicación**.

- **Mejoras en las actividades de control y planificación** por la reducción de desviaciones en los desarrollos.

Pero esta propuesta implica cambios de cierto calado en la compañía, que se encuadrarían dentro do los inconvenientes clásicos derivados de la aplicación de la reutilización sistémica, entre los que se encuentran algunos de los conocidos problemas de coste, carencia de formación inicial, resistencia al cambio y carencia de modelo de gestión; donde dentro del ámbito de este trabajo cabría destacar dos de ellos.

12.1 Resistencia al cambio.

Para obtener los beneficios deseados cabría diferenciar la separación de tareas entre Ingenieros de aplicación e Ingenieros de estructura o arquitecturales, donde ambos tendrían roles completamente diferentes.

El primero de ellos, junto con el departamento de arquitectura software, se encargaría de la realización de frameworks, modelos generativos, análisis de dominios, confección de DSLs (lenguaje específico de domino), diseño de metamodelos, generación de transformaciones, etc.

Todo ello bajo los paraguas arquitecturales a ser utilizados por los ingenieros de aplicación a modo de cajas negras, pero donde se requiere el conocimiento de qué se está utilizando y para qué sirve; por lo que se puede hacer una idea más o menos clara del cambio orgánico que esto implica y la gestión y comunicación que se requiere para que este cambio sea efectivo.

Por otro lado, los Ingenieros de aplicación dejarían de ocuparse de componentes repetitivos que no aportan valor a su desempeño, y que en la mayoría de los casos simplemente meten ruido en la tarea principal que es la resolución de la solución de la lógica de negocio. Pero también han de conocer herramientas relativas a las citadas anteriormente para la comprensión de su desempeño. Es decir, no se trata de una tarea menor, si no de realizar una segmentación del trabajo que redunde en la calidad del software producido, la reducción del coste y por lo tanto la mejora en ellos beneficios, así como la satisfacción del ingeniero que se ocupa de lo realmente necesario y útil en su desempeño.

12.2 El modelo de gestión.

A la hora de aplicar la reutilización de software sistémica aparecen otros cambios significativos a aplicar, como es un **plan de reutilización**, es decir, la definición e incorporación de un modelo de gestión que aborde tanto el problema técnico de un marco de aplicaciones, como la gestión en IT con el resto de los grupos de trabajo para la debida aplicación de componentes reutilizables.

Esto conlleva una fuerte inversión inicial, tras la cual la empresa espera recoger beneficios en el futuro. Es decir, se espera un **retorno de la inversión (ROI).**

Hoy día los grandes constructores de software proporcionan de facto una reducción de costes en la producción de **frameworks** de desarrollo que consisten en una serie de estructuras y tecnología definidas que básicamente facilitan la programación.

En este contexto definido y en el momento actual de evolución tecnológica, los avances en las propuestas de estilos arquitectónicos y sus expresiones prácticas con apoyo de estos frameworks, permiten ya de hecho tener y acelerar el proceso de implantación de soluciones reutilizables.

También se ha definido una serie de **patrones de diseño** en la definición arquitectural que son en realidad y por definición herramientas útiles en la reutilización sistemática en el software.

Por último, cabría introducir modelos de gestión en los que intervengan técnicas de evaluación, de métricas y gestión dinámica de procesos combinados con el desarrollo de modelos de cálculo del **retorno de la inversión** (**ROI)**, para convencer a la dirección a través

de un sistema de ciclo de mejora **PDCA (Plan, DO, Check, Act)** de la idoneidad de avanzar en la reutilización como un activo importante.

La arquitectura de software describe los componentes básicos de un sistema de software y su combinación interna. Representa la decisión de diseño más temprana.

La decisión acerca de su diseño es uno de los puntos más críticos e importantes en el proceso de desarrollo de un software, puesto que una vez llevada a cabo sólo es modificable con mucho esfuerzo, Esta se determina por criterios como la calidad, resistencia al cambio, seguridad y rendimiento, entre otros.

Por lo tanto, la arquitectura no es algo a lo que renunciar, sino al contrario; aunque es difícil hacerlo bien y de manera equilibrada.

Conseguir el equilibrio entre el coste y el beneficio, en términos de retorno de la inversión **(ROI),** a través de sistemas de mejora continua **(PDCA, el círculo de Demming),** incluyendo el objetivo de alinear de TI con la estrategia de negocio **(cadena de valor empresarial,** Michael Porter en su obra, '*Competitive Advantage: Creating and Sustaining Superior Performance (1985)*'**)**, es el camino a seguir para la consecución de los objetivos empresariales.

Capítulo 13. Posibles líneas de trabajo futuras

A lo largo de este documento en sus distintas secciones se ha dado respuesta a los objetivos planteados en origen. En el anterior epígrafe se analizan las conclusiones del trabajo.

En este trabajo se ha presentado una propuesta de diseño arquitectónico para la difusión de información del sistema financiero que utiliza e integra tecnologías emergentes y un diseño arquitectural de tendencia.

El diseño orientado a microservicios y dirigido por eventos mejora en este tipo de dominios interesantes propiedades como son extensibilidad, reutilización, proactividad e inteligencia, aplicado al dominio de la analítica de información bursátil.

Por lo tanto, se podrían abrir **varias vías** de **posibles líneas de trabajo futuras**:

13.1 Cambio de paradigma en la producción de software

Una representa la propuesta arquitectural, objetivo principal de este trabajo como el detonante para un **cambio de paradigma en la producción de software** como es la reutilización sistémica de software, y por lo tanto el **modelo de gestión aplicable** a ese nuevo paradigma, con modelos en los que intervengan técnicas de evaluación, de métricas y gestión dinámica de procesos combinados con el desarrollo de modelos de cálculo del ROI bajo un sistema de ciclo de mejora continua **PDCA (Plan, DO, Check, Act)**.

13.2 Evaluación de la arquitectura de software.

Incluiría una opción más especialista de estudio que tenga que ver únicamente con la evaluación, uso de métricas, y costes asociados a la arquitectura de software, que determinen a través de sus atributos para obtener mejoras continuas de dicha arquitectura. En este ámbito podrían ser varias las opciones a escoger, puesto que es un tema tan extenso como poco investigado en la reciente historia de la arquitectura del software.

13.2 enfoque evolutivo de la EDA.

La evolución tecnológica de los últimos años ha provocado que la web dejara de ser un entorno pasivo en el que mostrar información, para convertirse en un entorno dinámico y vivo que nos permite interactuar a modo de sistema.

En la actualidad la EDA es un 'modelo' aun reciente, con lo cual son aplicaciones simples que manejan algunos eventos y no presentan una complejidad de eventos interactuando. En el futuro se le asocia con la IA; el machine learning y la analítica avanzada.

En este espacio se pretende exponer una solución que tiene que ver con el campo de la analítica, la extracción, transformación y carga de información, en el dominio de aplicación de los mercados financieros. Aunque no se aborda el campo de la analítica avanzada, no es menos cierto que abre el camino para continuar esta línea de trabajo.

La arquitectura presentada utilizando MQTT, usada en IoT por su bajo consumo y la poca sobrecarga de la red que produce entre otras de sus características, permite incorporar más dispositivos *wearables*, móviles, y otros sensores IoT, que pudieran publicar datos en nuevos *topics,* como por ejemplo, servicios de valor añadidos a clientes para valorar el comportamiento, acierto o no de las decisiones de los inversores. En definitiva, la extensibilidad del sistema es uno de los puntos fuertes de este diseño.

La operativa de los mercados financieros a nivel mundial, la interconexión de estos, los unos con los otros, ofrece una perspectiva tecnológica que señala el camino de la EDA como sustrato tecnológico de diseño de software como mejor desarrollo para la industria. Esto unido a la **IA**, **Machine learning** y **analítica avanzada** permitiría crear ecosistemas software que den una solución tecnológica sólida para las demandas empresariales.

Para el despliegue de soluciones de este tipo la **EDA** pasaría a denominarse como **CEP** (Procesamiento de eventos complejos) para aprovechar el poder de eventos a administración automatizada sin comprometer el control de los administradores a través de conceptos como eventos, causalidad, jerarquías de eventos, modelos de evento y reglas.

De forma automática y transparente para los usuarios, el sistema realiza la recogida de datos y analiza éstos en tiempo real aplicando técnicas de rastreo en webs (en algunos casos), sacando partido de las características de las diferentes plataformas utilizadas. A posteriori,

los datos pueden transformarse bajo un motor de reglas de negocio y variables dependientes y anomalías, así como para hacer evaluaciones y clasificaciones.

Las nuevas plataformas *cloud*, móviles y *wearables* disponen recursos (distintos tipos de sensores) y capacidades adicionales (procesamiento y almacenamiento) a las tradicionales, así como características (bajo coste, fáciles de portar, etc.), que aumentan y potencian sus posibilidades en términos de aplicabilidad en diferentes dominios, como también en el dominio de Inntech.

Como trabajo futuro se podría llevar estudios experimentales de manera formal de una forma holística (dimensiones físicas, cognitivas y socioemocionales) con una cantidad significativa de participantes. Así se podrán recoger datos suficientes para realizar con validez análisis estadísticos y basados en técnicas de inteligencia artificial. Los profesionales de los sistemas financieros podrían desarrollar programas de formación de manera personalizada.

Bibliografía

[1] Arquitectura basada en eventos: cómo SOA habilita la empresa en tiempo real

Editorial: Addison-Wesley Professional
Fecha de lanzamiento: febrero de 2009
ISBN: 9780321591388

[2] Creación de microservicios con .NET Core 2.0 - Segunda edición

Editorial: Packt Publishing
Fecha de lanzamiento: diciembre de 2017
ISBN: 9781788393331

[3] Microservicios. Martin Fowler

https://martinfowler.com/microservices/

[4] Software Engineering Institute (SEI)

https://www.sei.cmu.edu/architecture/definitions.html

[5] Cadena de valor de Michael Porter

https://es.wikipedia.org/wiki/Cadena_de_valor

[6] Arquitectura dirigida por eventos

https://es.wikipedia.org/wiki/Arquitectura_dirigida_por_eventos

[7] Arquitectura Orientada a Microservicios y Dirigida por Eventos para el Desarrollo de Sistemas de eSalud Avanzados en ResearchGate

https://www.researchgate.net/publication/346390449_Una_Arquitectura_Orientada_a_Microservicios_y_Dirigida_por_Eventos_para_el_Desarrollo_de_Sistemas_de_eSalud_Avanzados_Caso_de_Evaluacion_de_Fragilidad_en_Mayores/link/5fbf73dca6fdcc6cc669d834/download

[8] SOLID

https://docs.microsoft.com/es-es/archive/blogs/cdndevs/the-solid-principles-explained-with-motivational-posters

[9] Ingeniería de software

https://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software

[10] Creación de microservicios, segunda edición. Autor: Sam Newman.

Editorial: O'Reilly Media, Inc.
Fecha de lanzamiento: agosto de 2021
ISBN: 9781492034025

[11] Micro Front-End. Martin Fowler

https://martinfowler.com/articles/micro-frontends.html

[12] Scraping

https://es.wikipedia.org/wiki/Web_scraping

[13] Muerte por arquitectura. O'reily

https://learning.oreilly.com/library/view/applied-soa-service-oriented/9780470223659/xhtml/Chapter02.html#chapter02

[14] Arquitecturas sin servidor

https://martinfowler.com/articles/serverless.html

[15] Event-Driven Architecture: How SOA Enables the Real-Time Enterprise

Publisher: Addison-Wesley Professional
Release Date: February 2009
ISBN: 9780321591388

[16] Building Event-Driven Microservices

by Adam Bellemare
Publisher: O'Reilly Media, Inc.
Release Date: July 2020
ISBN: 9781492057895

[17] Market Data. Filial de Grupo SIX-BME

https://www.bmemarketdata.es/esp/

[18] Extract, transform y and Load (ETL)

https://es.wikipedia.org/wiki/Extract,_transform_and_load#:~:text=Extract%2C%20Transform%20and%20Load%20(%C2%AB,en%20otro%20sistema%20operacional%20para

Anexo.

Rastreadores (crawlers,scrapers)

Un **crawler** es un recolector de datos. Y el **scraper** es una técnica de *Black HAT SEO* con el objetivo de copiar datos, como el contenido de un sitio web.

Las arañas web o crawlers son bots de Internet. Este método se llama reastreo web, y es el proceso de que **acceso a páginas de la web de una forma metódica y automatizada** a través de herramientas de scraping.

Los web crawlers son utilizados por motores de búsqueda como Google. El *web scraping* se enfoca más en la transformación de datos sin estructura en la web (como el formato HTML) en datos estructurados que pueden ser almacenados y analizados en una base de datos central, en una hoja de cálculo o en alguna otra fuente de almacenamiento. Alguno de los usos del *web scraping* son la comparación de precios en tiendas, la monitorización de datos relacionados con el clima de cierta región, la detección de cambios en sitios webs y la integración de datos en sitios webs. También es utilizado para obtener información relevante de un sitio. [12]

ETL (extraer, transformar y cargar)

Un ETL es el proceso que permite a las organizaciones mover datos desde múltiples fuentes, reformatearlos y limpiarlos, y cargarlos en otra base de datos, data mart, o data warehouse para analizar, o en otro sistema operacional para apoyar un proceso de negocio **[18]**.

Es un proceso que consiste en tres partes, extraer, transformar y cargar.

La parte de extracción consiste en extraer datos de distintos orígenes, bien en forma de bases de datos relacionales, bien en ficheros planos o de formato específico; y posteriormente analizar esos datos y validarlos.

La parte de transformación implica la aplicación de reglas de negocio para convertirlos en datos que serán cargados por el sistema. En él pueden intervienen cálculos, filtrados de la información, unión de distintas fuentes, etc…

Y por último en la fase de carga, es el momento en el que los datos transformados se cargan en el sistema.