Sistema de automatización de consultas en sistemas Datawarehouse

Lluc Vicente del Cerro

Resumen— Este trabajo presenta la migración de procesos ETL basados en XSLT que transformaban datos XML hacia una nueva arquitectura capaz de procesar datos JSON provenientes de la API EGRETA, utilizada en el Data Warehouse de la Universidad Autónoma de Barcelona. El proyecto evalúa diferentes alternativas tecnológicas (Saxon, Python con Pandas y Polars, Apache NiFi) para garantizar la continuidad operativa y mejorar la eficiencia del sistema. Además, se incorpora un caso de uso centrado en el análisis de género en la comunidad investigadora.

Palabras clave— ETL, Saxon, Datawarehouse, XSLT, integración, migración, ODI, dataframe, integración, análisis de género, transformación, calidad.

Abstract— This project addresses the migration of XSLT-based ETL processes that previously handled XML data, towards a new architecture that supports JSON data from the EGRETA API, integrated within the Data Warehouse at the Universitat Autònoma de Barcelona. The study evaluates different technological alternatives (Saxon, Python with Pandas and Polars, Apache NiFi) to ensure operational continuity and improve system efficiency. A gender analysis use case has also been developed for the research community.

Index Terms— ETL, Saxon, Data Warehouse, XSLT, Data Integration, Migration, Oracle Data Integrator (ODI), dataframes, Gender Analysis, Data Transformation, Data Quality.

1 Introducción - Contexto del trabajo

En el contexto actual los sistemas Datawarehouse desempeñan un papel crucial en la gestión y análisis de grandes volúmenes de datos. Un Datawarehouse [1] es una infraestructura diseñada para recopilar, almacenar y gestionar datos provenientes de diversas fuentes con el objetivo de facilitar la toma de decisiones estratégicas. Estas plataformas permiten consolidar información de diferentes sistemas en un repositorio centralizado, donde se pueden realizar consultas eficientes y generar informes complejos; un aspecto clave en estos entornos es la capacidad de integrar datos de múltiples fuentes de forma automática, fiable y eficiente.

La Universidad Autónoma de Barcelona (UAB) cuenta con un datawarehouse corporativo en el que para integrar los datos, se emplean procesos ETL (Extract, Transform, Load) mediante la herramienta Oracle Data Integrator (ODI), que permite extraer información, procesarla y almacenarla de manera estructurada.

Una de las bases de datos clave para la investigación en la UAB es EGRETA[2] (Entorno para la gestión de la investigación y la transferencia), que almacena información sobre proyectos de investigación, publicaciones y colaboraciones académicas.

Actualmente la integración de datos se lleva a cabo a partir de llamadas a webservices con una API REST [3], que devuelve respuestas en un formato XML (Extensible Markup Language).

Este es un producto de Elsevier, una empresa líder en la gestión y análisis de información científica. Estas respuestas son transformadas a CSV (Comma-Separated Values) mediante SAXON y se cargan en tablas temporales de ODI donde se realiza su procesamiento y almacenamiento definitivo en el datamart de investigación, una sección especializada del datawarehouse destinada al análisis de datos científicos y académicos.

El problema radica en que Elsevier ha anunciado que, en la próxima versión de su API, dejará de responder en formato XML y solo proporcionará datos en JSON (JavaScript Object Notation). Por lo tanto, es de vital importancia adaptar los procesos ETL existentes para garantizar la continuidad de la ingesta de datos sin pérdidas de información ni errores en la conversión.

1.1 Objetivos

El objetivo principal del trabajo es migrar los procesos de transformación de datos actuales, que utilizan XSLT para procesar XML, a nuevos XSLT que permitan parsear y transformar datos en formato JSON, manteniendo la misma funcionalidad y garantizando la equivalencia en

- *E-mail de contacto: Lluc.Vicente@autonoma.cat*
- *Trabajo tutorizado por: Antonio Espinosa Morales*
- *Curso 2024/25*

los resultados. Además, se buscará poner en práctica estos datos mediante un caso de uso aplicado a la analítica y visualización de datos, concretamente en el análisis de género en la comunidad investigadora perteneciente a la Universidad Autónoma de Barcelona. Para ello, se establecen los siguientes objetivos específicos:

1. Diseñar un nuevo sistema de ingesta de datos basado en JSON y verificar que los resultados obtenidos sean equivalentes a los obtenidos previamente con XML.
2. Evaluar la capacidad de ODI para procesar JSON y determinar si es necesario utilizar herramientas adicionales.
3. Investigar y comparar otras herramientas o métodos para optimizar el proceso ETL y mejorar la eficiencia.
4. Implementar y probar el sistema migrado en un entorno de desarrollo antes de su integración final.
5. Desarrollar un caso de uso centrado en el análisis de género de la comunidad investigadora
6. Incorporar y/o documentar técnicas de desambiguación de datos especialmente en casos de colisiones por nombres repetidos o identificadores inconsistentes.
7. Documentar el proceso de migración y las pruebas realizadas.

2 ESTADO DEL ARTE

Este apartado se centra en las herramientas clave utilizadas para poder llevar a cabo este proyecto. Las herramientas han sido seleccionadas en función de su utilidad en las diferentes etapas del proceso: comunicación con la API, transformación de datos, integración con el Data warehouse y visualización de resultados.

Una API (Application Programming Interface) es un conjunto de reglas y definiciones que permiten que diferentes aplicaciones se comuniquen entre sí. Dentro de este contexto, una API REST (Representational State Transfer) es un tipo de API que sigue principios arquitectónicos específicos para permitir una comunicación eficiente a través del protocolo HTTP. Las API RESTful utilizan métodos estándar como GET, POST, PUT y DELETE para interactuar con recursos en un servidor.

PureAPI es la API utilizada en este proyecto para recuperar datos desde el sistema de Elsevier.
El flujo de trabajo con PureAPI en el proyecto es el siguiente:
Realizar peticiones a la API REST de Elsevier para recuperar datos en formato JSON.
Validar la estructura de las respuestas JSON para garantizar que contienen la información esperada.
Convertir los datos JSON a un formato adecuado para su transformación mediante XSLT o alguna alternativa.
Para llevar a cabo estas tareas, se utilizará Postman [4], una aplicación que permite probar, depurar y automatizar solicitudes a APIs REST.

Otra de las herramientas para ello será Saxon[5], un procesador de XSLT que ya ha sido usado y probado por el equipo de gobierno de datos de la UAB para manejar las transformaciones de XML a CSV. Actualmente, con la migración a JSON, será necesario adaptar los XSLT existentes o desarrollar nuevos procesos de transformación.

Para la integración en el Data Warehouse se hará uso de Oracle Data Integrator (ODI).
ODI[6] es la herramienta utilizada en la UAB para la integración de datos dentro del Data Warehouse; se encarga de la fase final del proceso ETL (Extract, Transform, Load), permitiendo la carga de los datos transformados en tablas estructuradas.

Para el seguimiento y visualización del caso de uso, se utilizará Power BI[7], una herramienta de Business Intelligence (BI) que permite la visualización de gráficos personalizados, informes, cuadros de mando interactivos.

De esta manera se podrá visualizar los datos fácilmente y gracias a ello obtener información crucial a la hora de encarar decisiones.

3 METODOLOGÍA Y PLANIFICACIÓN

Se seguirá un enfoque iterativo permitiendo realizar ajustes y mejoras progresivas en cada fase del desarrollo y facilitando la detección a tiempo de problemas y permitiendo una implementación ágil a la vez que flexible. Se estructurará en las siguientes fases:

- Análisis de la documentación y procesos: Revisar la estructuración, proceso y diseño de flujo ETL actual y analizar la nueva versión de la API para comprender su estructura y diferencias con XML.
- Desarrollo del sistema: Implementar un nuevo mecanismo para poder realizar la migración de XML a JSON.
- Evaluación de ODI: Analizar si ODI permite la ingesta de JSON y, en caso contrario, definir una estrategia alternativa.
- Exploración de herramientas: Se evaluarán diversas herramientas comparando sus métricas y eficiencia buscando la más óptima para llevar a cabo la carga de datos.
- Pruebas y validación: Batería de pruebas y validaciones para comprobar la equivalencia entre los datos obtenidos en formato XML y JSON.
- Construcción del caso de uso: Se llevará a cabo de manera escalonada: empezando por perfiles investigadores individuales, luego colecciones de personas y finalmente con la integración completa.
- Visualizaciones principalmente de indicadores relevantes mediante herramientas de visualización de datos como PowerBI.
- Implementación final: Integrar la solución en el entorno final.

3.1 Planificación

La planificación del proyecto se ha representado mediante un diagrama de Gantt[8] (Apéndice 1), una herramienta visual que permite organizar las actividades en función del tiempo y facilitar su seguimiento. En el diagrama, cada fase del proyecto está distribuida a lo largo de las semanas, asegurando una ejecución estructurada. Se puede observar que el análisis de documentación y el desarrollo del sistema ocupan las primeras semanas, seguidos por la evaluación de ODI y la exploración de herramie
tas.
Posteriormente, se realizan pruebas y validaciones, junto a unas visualizaciones de indicadores de rendimiento para culminar con la implementación final. Gracias a este enfoque se tendrá una visión más global y distribuida que permitirá gestionar mejor los tiempos y los recursos, asegurando que cada tarea se complete dentro de los plazos establecidos.

4. DESAROLLO

En esta sección se describe detalladamente el proceso de migración del proyecto. El trabajo se ha dividido en diferentes fases con el fin de facilitar su implementación y garantizar un seguimiento ordenado de las tareas.

La base de datos sobre la que estamos operando está compuesta por tablas como: Persons, Awards, ResearchOutputs, Organisations, Projects, ExternalOrganisations, Publishers, Activities, StudentTheses, etc.
A continuación, en la figura 1, se muestra una parte del esquema de las tablas de la UAB.

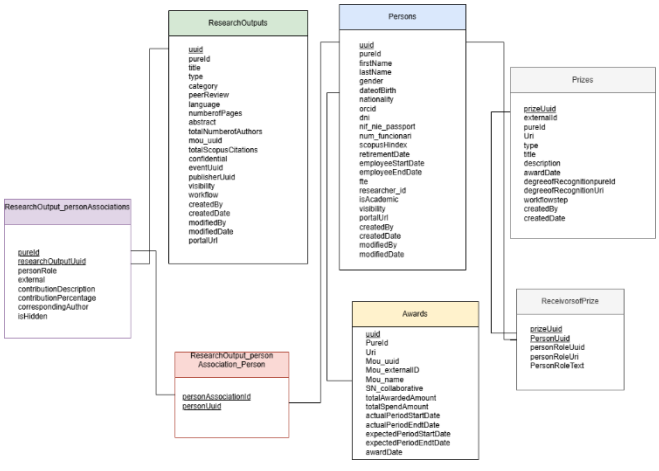

Fig. 1. Esquema parcial de EGRETA

Los datos más relevantes son los registros de personas de la tabla Persons que contiene un total de 24.581 registros con los atributos que se muestran en la figura.
Y los de Awards con un total de 66.946 registros que hace referencia a información relacionada con ayudas, concesiones o financiamientos.
Conjuntamente con ResearchOutputs que hace referencia a las publicaciones.

4.1 Comparativa funcional de las APIs XML y JSON

Durante las primeras fases del proyecto se realizaron pruebas intensivas sobre las APIs REST proporcionadas por el sistema EGRETA. En la siguiente figura se puede ver reflejado el funcionamiento mediante peticiones para poder adquirir los datos más relevantes; ya que no trabajamos con todos los endpoints dentro de EGRETA.

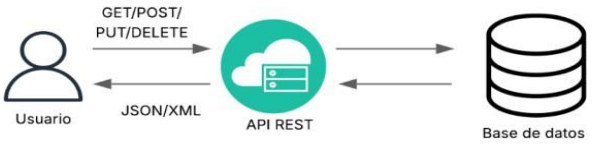

Fig. 2. Diagrama de funcionamiento de API REST

La API REST ofrece dos versiones distintas: una orientada a respuestas en formato XML y otra en formato JSON. Ambas APIs exponen, en teoría, información equivalente, pero tras un análisis detallado se han podido ver diferencias notables en su estructura, disponibilidad de campos, endpoints y comportamiento general.

La API XML se encuentra accesible a través de URLs con el prefijo https://egreta.uab.cat/ws/api/524, mientras que la API moderna en JSON se localiza en https://egreta.uab.cat/ws/api. La exploración y prueba de estas APIs se ha hecho mediante Postman, permitiendo lanzar peticiones tipo GET y POST con cabeceras personalizadas para la autenticación mediante nuestra apikey.

Se descubrió que el conjunto de endpoints disponibles en ambas APIs no es exactamente coincidente. La API JSON incluye un gran número de endpoints de tipo allowed, search y actions, muchos de los cuales no tienen un equivalente directo en la versión XML. En cambio, la API XML dispone de endpoints como pueden ser /persons/active, /persons/former, /persons/{id}/activities, que no se encuentran documentados ni disponibles en la API JSON.

El hecho de que haya varianza en los endpoints no nos debería afectar ya que de forma general lo que se pretende es atacar el endpoint entidad base como podria ser Persons o Awards y a partir de ahí realizar los procesos de filtrados en la ETL, de esta manera se correspondería como ya lo hacían previamente el equipo de Gobierno del Dato.

4.2 Diferencias estructurales en las respuestas de las APIs

La estructura interna de los documentos devueltos también difiere sustancialmente entre los dos formatos:

- En la API XML, la raíz es un nodo <result> que contiene información como <count>, <pageInformation>, <navigationLinks> y un nodo <items> con una lista de entidades (por ejemplo, <person>).
- En la API JSON, la raíz es un objeto con propiedades count, pageInformation, navigationLinks y items, donde items es un array de objetos.

Esto implica que en XML la navegación se hace mediante jerarquías de elementos, mientras que en JSON se realiza a través de mapas (objetos) y arrays. Además:

- Los atributos en XML (por ejemplo, @uuid) pasan a ser campos de objeto en JSON (por ejemplo, "uuid").
- Los campos multilingües, que en XML están modelados como nodos <term locale="ca_ES">, en JSON se representan como mapas { "ca_ES": "...", "en_GB": "..." }.
- Las colecciones repetidas, como staffOrganisationAssociations o identifiers, se representan en XML como listas de nodos <staffOrganisationAssociation>, mientras que en JSON se modelan como arrays.

4.3 Análisis de las diferencias funcionales y contenido

Ciertas llamadas a la API JSON no devolvían datos que sí estaban disponibles en la versión XML. Por ejemplo, peticiones a /persons/{uuid} funcionaban correctamente en XML pero llegaba a devolver errores en JSON, lo que indica una posible falta de sincronización entre ambas versiones del sistema; además, se detectaron diferencias en los campos disponibles: algunos registros en JSON no incluían valores como orcid, nif_nie_passport o employee, mientras que esos campos estaban siempre presentes en las respuestas XML seguramente debido a un tema de navegación hasta el dato o de guardado diferente.

Se comprobó que las estructuras de los endpoints search en JSON devuelven arrays planos que son más fáciles de procesar en bulk, mientras que XML está más orientado al acceso puntual y jerárquico. Esto influye directamente en la manera en la que deben desarrollarse los XSLT para parsear correctamente la información.

4.4 Consideraciones para la migración de procesos ETL

Estas diferencias obligan a cambiar profundamente el enfoque de los estilos XSLT existentes. La migración requiere la adopción de XSLT 3.0 con soporte para funciones como parse-json(), json-doc(), y el acceso a mapas y arrays.

También ha sido necesario definir nuevos mecanismos de control para manejar estructuras ausentes, arrays vacíos y tipos de dato heterogéneos para definir transformaciones robustas que garanticen compatibilidad con el pipeline de datos.

4.5 Flujo de trabajo para la adaptación

Para poder llevar a cabo el proceso de adaptación del sistema de XML a JSON empezaremos por la descarga de todas las salidas o respuestas de la API Rest en formato json, de ahí a adaptar los XSLT para que acepten JSON y consigan navegar y funcionar correctamente para posteriormente poder ejecutarlo a través de una terminal desde Visual Studio Code mediante el motor Saxonica con la nueva versión 11.6, que nos permite trabajar mejor con JSON y así finalmente recibir y poder guardar en los directorios de destino los archivos csv finales.

Fig. 3. Flujo de ejecución

4.6 Implementación de funciones de limpieza y normalización de datos

Se han adaptado las funciones auxiliares utilizadas para el preprocesamiento de la información dado que muchos de los campos que se obtienen de EGRETA, contienen texto multilingüe, caracteres especiales (como saltos de línea, punto y coma), o necesitan ser truncados a una longitud máxima por requerimientos del sistema de destino.

Para mantener la coherencia en los CSV generados, se diseñó un conjunto de funciones reutilizables en XSLT 3.0 orientadas específicamente al tratamiento de datos extraídos desde estructuras JSON. Estas funciones permiten:

- Eliminar caracteres problemáticos mediante expresiones regulares.
- Truncar cadenas de texto a un número configurable de caracteres.
- Extraer términos localizados en diferentes idiomas.
- Unificar los valores multilingües en un único campo delimitado.

4.7 Pruebas de uso: transformación de entidades individuales y colecciones

Una vez definidos los mecanismos de limpieza, se aborda- ron dos casos principales de transformación de datos: uno orientado a registros individuales y otro a la iteración so- bre grandes volúmenes de datos.

En el **caso individual**, se trabajó primero con respuestas del endpoint que devuelve una persona concreta, accediendo a campos simples (uuid, name, gender, dateOfBirth) y a colecciones anidadas como identifiers. Se aplicaron condi- cionales para convertir códigos de URI en valores equiva- lentes y para formatear fechas a un formato estándar DD/MM/YYYY.

Una vez comprobado el funcionamiento se continuó con el caso general.

El **caso general** supuso iterar sobre una colección de personas completa devuelta por la API. Para ello se recorrió el array items del objeto JSON y se aplicaron las mismas reglas que en el caso individual. La construcción de las filas del CSV se realizó utilizando string-join(...) para garantizar una correcta alineación horizontal de todos los campos, evitando el error frecuente de obtener datos en vertical o mal separados.

Una vez validada esta prueba de uso se replicó para todos los otros endpoints más relevantes.

Se diseñaron plantillas reutilizables que permitieran aplicar lógicas comunes en ambos escenarios, facilitando la escalabilidad y el mantenimiento futuro.

Para verificar que ODI puede asumir este cambio a JSON se tuvo que cambiar la llamada general que hace el bucle para conseguir extraer toda la información de la nueva api de Json y que en vez de ejecutar el directorio con las variables xml ejecutar el directorio json.

De esta manera los csv resultantes acaban siendo tablas de destino en Oracle Data Integrator.

5. EXPLORACIÓN DE HERRAMIENTAS

Una vez establecida la base funcional para la migración de procesos del formato XML a JSON, se plantea la exploración de otras vías que pueden ser más eficientes o pertinentes para optimizar el flujo de trabajo.

Esta sección se centrará en el análisis de herramientas y tecnologías complementarias que puedan aportar mejoras ya sea en términos de rendimiento, escalabilidad, facilidad de integración y mantenimiento.

El objetivo es identificar posibles alternativas —ya sea en el ámbito de la transformación de datos, la automatización de procesos o la integración con el Data Warehouse— que permitan reforzar y agilizar el sistema desarrollado, garantizando su sostenibilidad a largo plazo y su capacidad de adaptación frente a futuros cambios tecnológicos.

5.1 Python

El primer método alternativo que se ha probado para gestionar la obtención y transformación de datos desde la nueva API REST ha sido el uso del lenguaje Python; gracias a su capacidad para manejar estructuras complejas y sus librerías.
Python permite realizar todo el flujo desde la extracción hasta la exportación sin necesidad de herramientas externas adicionales. En este enfoque, se han utilizado librerías como requests para lanzar peticiones HTTP autenticadas contra los distintos endpoints, y csv y json para procesar las respuestas en formato estructurado y generar archivos CSV finales de forma controlada y flexible.
Una ventaja clave de este enfoque es que permite diseñar el proceso de forma dinámica: mediante bucles automatizados se puede recorrer una lista de endpoints definidos y generar, para cada uno de ellos, los extractos correspondientes con la lógica de transformación adecuada.

La primera implementación se ha realizado con la librería pandas**[9]**, aprovechando su capacidad para gestionar estructuras tabulares de datos de forma flexible. El proceso comienza con la lectura del archivo JSON y la extracción de las entradas contenidas en el campo items. A continuación, se recorren secuencialmente los objetos y se construyen diccionarios que recogen únicamente los campos relevantes, incluyendo valores anidados.
Estos registros se acumulan en una lista que posteriormente se convierte en un DataFrame que se exporta a un archivo CSV asegurando la codificación adecuada para su uso posterior.

La segunda implementación utiliza la librería polars[10], una alternativa más reciente a pandas diseñada para maximizar el rendimiento en tareas de procesamiento de datos. A diferencia de pandas, polars adopta un modelo de memoria columnar "columnar storage model" , lo que permite un acceso más eficiente y operaciones vectorizadas más rápidas, especialmente en conjuntos de datos medianos o grandes. El flujo del script es equivalente al de la versión con pandes.

Polars destaca por una menor utilización de memoria y tiempos de conversión generalmente más bajos, manteniendo la misma estructura lógica y resultados que su contraparte en pandas.
Ambas soluciones han sido diseñadas de forma modular, lo que facilita su reutilización para diferentes endpoints de la API, adaptando únicamente la lógica de extracción de campos según la estructura de cada entidad.

5.2 Apache NiFi

La siguiente opción que se ha probado ha sido Apache NiFi. Apache NiFi[11] es una herramienta de código abierto para la automatización y gestión de flujos de datos entre sistemas. Permite diseñar, controlar y monitorizar canalizaciones de datos mediante una interfaz gráfica, con soporte para transformaciones, filtrado, enrutamiento y conexión con múltiples fuentes y destinos de forma escalable y fiable.

Esta nueva aproximación busca aportar mayor flexibilidad, escalabilidad y mantenimiento a futuro a la vez que haciendo uso de una herramienta más visual,.
A continuación se muestra el flujo actual de procesos en Apache NiFi:

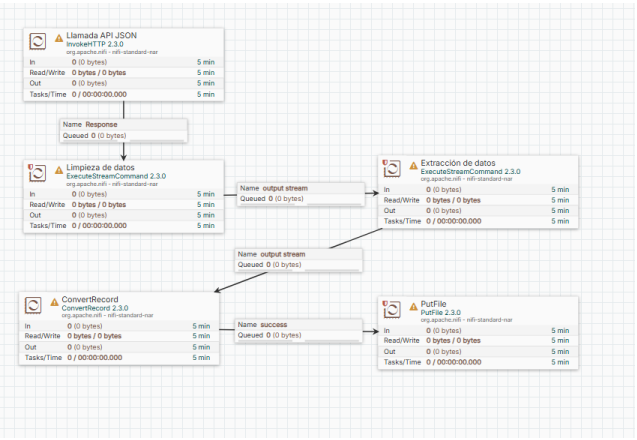

Fig. 4. Flujo de procesos

El flujo implementado comienza con un procesador de tipo InvokeHTTP, que realiza una llamada a uno de los endpoints expuestos por la API. Este procesador está configurado para emitir peticiones GET y recibir respuestas en formato JSON. A partir de ahí, la respuesta se redirige hacia una cadena de procesamiento compuesta por distintos módulos con responsabilidades separadas.

El primer módulo de la cadena es un procesador ExecuteStreamCommand que ejecuta un script externo de Python encargado de aplicar un conjunto de transformaciones genéricas sobre los datos recibidos. Estas transformaciones equivalen a las funciones de limpieza, normalización y estructuración que ya se aplicaban anteriormente mediante funciones definidas en XSLT.

Una vez completada esta etapa de limpieza, los datos se trasladan a un segundo procesador ExecuteStreamCommand, el cual también ejecuta un script externo, en este caso encargado de extraer exclusivamente los campos de interés según el modelo de datos que se desea obtener. Esta etapa permite trabajar de forma
específica en función de las necesidades del endpoint atacado, identificando, por ejemplo, los campos clave de identificación, datos personales, agrupaciones temáticas u otras campos relevantes. Esta separación entre la limpieza general y la extracción específica facilita una mayor reutilización de componentes y una mayor facilidad para adaptar el flujo a futuros endpoints sin modificar la lógica común.

Una vez los datos están estructurados de forma precisa en formato JSON plano, se pasa a la etapa de conversión. Para ello, se utiliza el procesador ConvertRecord, configurado con los servicios JsonTreeReader y CSVRecordSetWriter, que permiten convertir automáticamente los datos JSON a formato CSV. La lectura y escritura se configuran para inferir el esquema de forma dinámica y generar un fichero delimitado por comas, con cabecera incluida. Esta etapa sustituye la funcionalidad final que anteriormente se ejecutaba mediante las transformaciones XSLT aplicadas sobre el documento fuente.

Por último, se incorpora un procesador PutFile encargado de almacenar el fichero CSV generado en un directorio determinado del sistema.
Durante la implementación se ha prestado atención a la correcta gestión de las relaciones entre procesadores. Todas las relaciones no utilizadas han sido marcadas como terminadas automáticamente (auto-terminate) para garantizar la validez del flujo, y se ha verificado que cada componente procesa correctamente tanto entradas válidas como posibles errores o salidas vacías.

5.3 Comparativa de métodos

Con el objetivo de valorar las distintas alternativas técnicas implementadas, se ha procedido a una comparativa estructurada de los métodos utilizados. La evaluación se ha centrado en tres ejes principales: tiempo de ejecución, flexibilidad y reutilización de componentes.

5.3.1 Flexibilidad y reutilización

Desde el punto de vista de la flexibilidad, el método basado en XSLT + Saxon presenta ciertas limitaciones al estar fuertemente acoplado al esquema de entrada. Ante cualquier cambio en la estructura del JSON, es necesario modificar manualmente las plantillas, lo que complica la adaptación a nuevos formatos o a entidades con campos variables. Además, la reutilización de código entre plantillas es escasa, ya que cada entidad suele requerir una plantilla específica.

Por el contrario, los enfoques en Python en las 2 versiones permiten definir transformaciones de forma dinámica, lo que facilita su adaptación ante cambios estructurales o nuevos requisitos.
Las funciones de transformación pueden modularizarse fácilmente, reutilizándose entre scripts lo que reduce considerablemente el esfuerzo de mantenimiento y mejora la escalabilidad del sistema a medida que se incorporan nuevos conjuntos de datos.

Apache NiFi se sitúa en un punto intermedio: tiene una cierta flexibilidad pero también se ve un poco limitada para transformaciones complejas y procesos de limpieza de datos por lo que hay que recurrir a scripts externos para mayor configuración
Pero, su interfaz visual facilita la modificación de flujos sin necesidad de cambios estructurales profundos, aunque requiere una correcta organización de los scripts y configuraciones para mantener la coherencia del sistema.

5.3.2 Tiempo de ejecución

A continuación se muestran los tiempos de ejecución según los diferentes métodos o procedimientos realizados aplicados sobre un conjunto de datos especifico.

	Tiempo de ejecución por número de registros		
Métodos	1k	20k	50k
XSLT + Saxon	1.89s	13.12s	21.12s
Apache NiFi	2.1s	9.65s	16.5s
Python (pandas)	0.34s	5.65s	11.6s
Python (polars)	0.3s	5.56s	9.06s

Tabla 1: Tiempo de ejecución

En la primera tabla, los resultados evidencian que tanto pandas como polars ofrecen mejoras significativas en tiempos de ejecución en comparación con el método tradicional. Para un volumen de 50.000 registros, la solución basada en polars reduce el tiempo de procesamiento de 21,12 segundos (XSLT + Saxon) a tan solo 9,06 segundos, lo que representa un *speedup* aproximado de 2,33x. Por su parte, pandas logra completar el mismo volumen en 11,6 segundos, con un *speedup* de 1,82x.

Apache NiFi ofrece también un rendimiento competitivo, aunque su tiempo de arranque inicial es ligeramente superior para volúmenes pequeños. A partir de 20K registros, sus ventajas en automatización y paralelización lo sitúan por encima de XSLT en eficiencia.

Después de un análisis para determinar donde residía el proceso que consumia mas tiempo se llegó a la conclusión que el cuello de botella se originaba en la lectura del json, ya que ocupaba casi un 90% del tiempo de ejecución.

Por tanto se ha querido centrar los esfuerzos en intentar optimizar esta lectura para poder obtener un mejor rendimiento, ya que atacando este problema es donde conseguiremos una gran mejora.
Para ello se han barajado varias opciones, pero se ha optado finalmente por el uso del formato Ndjson (Newline Delimited JSON) exclusivamente en las soluciones implementadas con Python, ya que han demostrado ser las más eficientes en términos de velocidad y consumo de memoria.

NDJSON[12] es un formato basado en texto en el que cada línea representa un objeto JSON independiente. Esta estructura facilita una lectura secuencial línea a línea sin necesidad de cargar todo el contenido en memoria, lo que permite trabajar de forma más ligera con grandes volúmenes de datos. Además, su simplicidad estructural favorece la reutilización de código y su integración con flujos de transformación ya existentes.
Esta optimización no se ha considerado necesaria en otras herramientas como XSLT o Apache NiFi, dado que Python ha resultado ser la opción más sensible a mejoras en el rendimiento mediante formatos de entrada más eficientes.

En la siguiente tabla se muestra el tiempo de ejecución para estas 2 nuevas implementaciones.

	Tiempo de ejecución por número de registros		
Métodos	1k	20k	50k
Python (pandas)	0.15s	2.52s	3.96s
Python (polars)	0.15s	2.37s	3.8s

Tabla 2: Tiempo de ejecución con nuevo formato

La segunda tabla compara exclusivamente las versiones optimizadas de los scripts Python (pandas y polars) empleando como entrada archivos en formato NDJSON.

En comparación con el método tradicional (21,12s), obtenemos unes mejoras de tiempo para 50k registros muy notables alcanzando un *speedup* de 5,55x (polars) y 5,33x (pandas), destacando la gran eficiencia de este enfoque.
Esta evolución demuestra que el uso de NDJSON junto con polars o pandas es la opción más eficiente para escenarios donde se requiera alta velocidad de transformación manteniendo control del flujo mediante scripts reutilizables y fácilmente adaptables a cambios futuros.

6. Problemas de calidad en los datos

Antes de abordar el caso de uso definido, es necesario analizar y resolver ciertas incidencias relacionadas con la calidad y consistencia de los datos obtenidos.
Se han identificado varios escenarios problemáticos que afectan directamente a la integridad del modelo, tales como duplicidades parciales, registros con diferencias mínimas y colisiones de identidad entre personas con nombres coincidentes. Estas situaciones, si no se abordan adecuadamente, pueden conducir a errores en el análisis y a una interpretación incorrecta de los resultados. Por ello, se han establecido mecanismos de desambiguación y limpieza destinados a garantizar la unicidad, la coherencia y la trazabilidad de los datos consolidados. A continuación, se detallan los principales casos detectados y las soluciones aplicadas.

6.1 Desambiguación de nombre de personas

Es el caso en el que 2 personas comparten exactamente el mismo nombre; puede ser causado por errores de escritura o simplemente los nombres de las personas son homónimos, compartiendo por tanto el mismo nombre y apellidos.
La solución planteada son las comparativas por contexto, mediante información adicional ligada a la persona (atributos únicos) se harán unas comprobaciones de identificadores personales como son el ORCID, DNI o Pasaporte.

6.2 Registros “no exactamente coincidentes” duplicados

En ciertos datos podemos observar que la misma persona o campo a designar aparece registrado múltiples veces con ligeras variaciones.
Se plantea una solución basada en la normalización textual y comparación con tolerancia.

Se aplica una normalización y se definen reglas de comparación como la distancia de Levenshtein[13] para identificar registros que, aunque no son exactamente iguales, presentan una similitud significativa.

Una vez detectados los duplicados, se unifican en un único registro maestro y se mantiene una tabla de equivalencias para conservar la trazabilidad entre los registros originales y el consolidado.

7. Caso de uso

Mediante los datos migrados y varias muestras de datos facilitadas por la oficina del gobierno de los datos se ha planteado un caso de uso que quiere profundizar en un caso real como seria la desigualdad de género en la comunidad investigadora/docente, concretamente de la UAB, para poder mostrar una posible utilización de estos datos.
Se profundizará en un análisis de la desigualdad de género aunque con un conjunto de datos ciertamente limitado y siempre trabajando sobre datos agregados con el fin de proteger y respetar la privacidad de los datos personales y cumplir con las políticas de privacidad de datos según el RGPD[14].

A continuación se muestra la vista de modelo de datos cargada en PowerBI, donde se representan las distintas tablas utilizadas en el análisis, así como las relaciones definidas entre ellas.

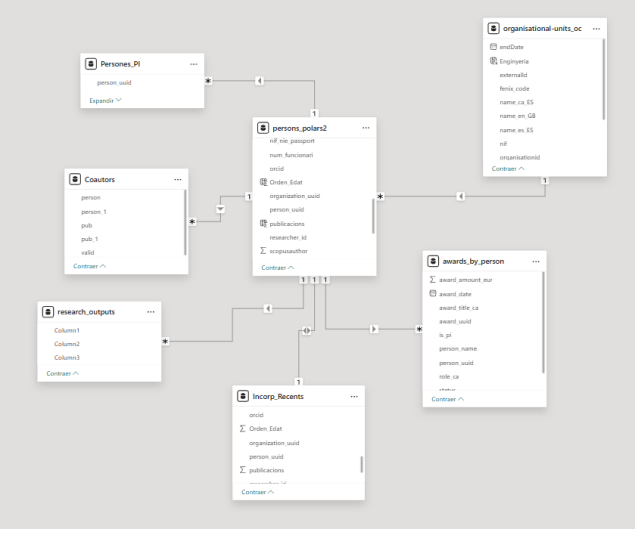

Fig. 5. Vista del modelo de datos

Una vez consolidado el modelo de datos, se procede a la creación de distintas visualizaciones que ayuden a explorar esta problemática de forma más directa y visual.

El gráfico de barras apiladas de la Figura 6 muestra la distribución de personas por franja de edad y género, se puede apreciar que en las franjas de edad de 55 años en adelante son las más predominantes por hombres y que esta predominancia ha ido cada vez a menos siendo las siguientes franjas mucho más equiparables como la franja de 45-54 que es la más numerosa o la de 25-35.

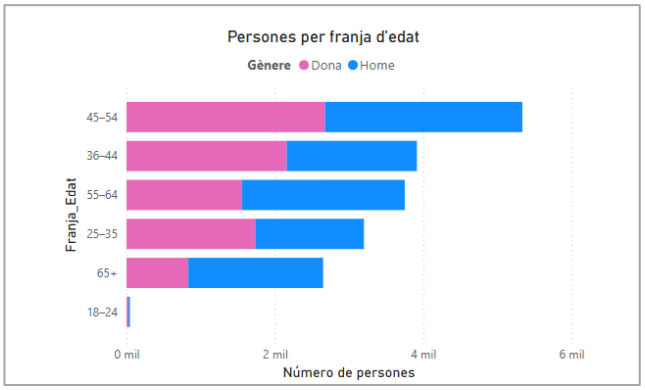

Fig. 6. Gráfico de personas por franja de edad

El gráfico de la izquierda perteneciente a la Figura 7, muestra que los nuevos ingresos registrados mantienen una distribución ciertamente equilibrada entre hombres (52,36%) y mujeres (47,64%), lo cual se traduce en que los procesos de acceso a la comunidad investigadora/docente están ciertamente balanceados.

Sin embargo, el gráfico de la derecha, que representa la distribución de investigadores principales (responsables de proyectos de investigación), revela una desviación significativa: el 58,43% de los proyectos están liderados por hombres frente al 41,57% por mujeres. Esta diferencia sugiere que, a medida que avanza la carrera investigadora, persiste una desigualdad en el acceso a posiciones de mayor responsabilidad, visibilidad o financiación.

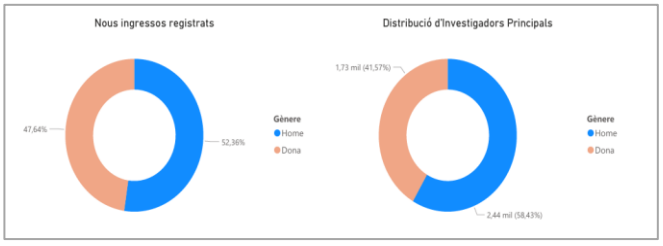

Fig. 7. Gráfico de nuevos ingresos e investigadores principales

El gráfico de la Figura 8 presenta la distribución de género en distintos departamentos de la universidad.
Se observa una gran variabilidad entre departamentos; mientras algunos departamentos como el de Psicología o Enfermería tienen una presencia femenina mayoritaria (superior al 70%), otros como Ciencias de la computación y Física presentan un claro predominio masculino.

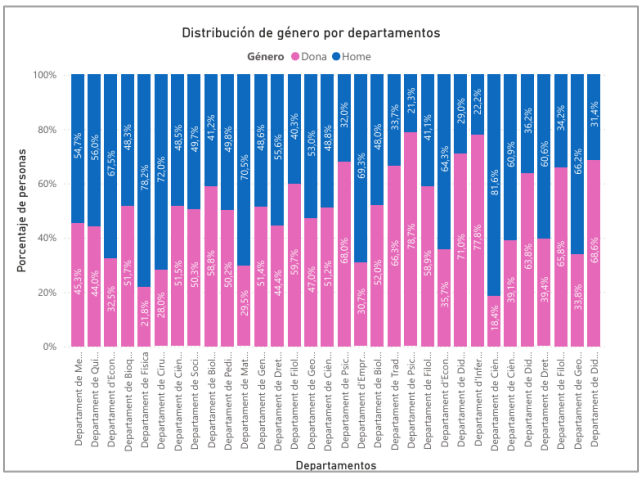

Fig. 8. Gráfico de distribución de género por departamentos

Haciendo especial enfásis en el campo de la ingeniería que es comúnmente conocido como uno de los campos con menos presencia femenina, podemos ver que los datos no engañan.
Alrededor de menos del 40% en varios de los principales departamentos de Ingeniería, encontrando un punto crítico en el Departamento de Ingeniería de la Información y de las Comunicaciones con una presencia del 13,7%.

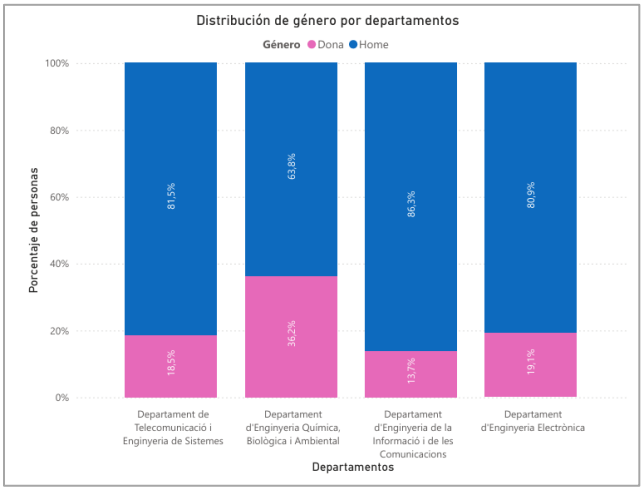

Fig. 9. Gráfico de distribución de género en Ingeniería

En la Figura 10, se muestra la evolución de la proporción de mujeres a lo largo de los últimos 40 años (1985-2025).
Se observa una tendencia claramente ascendente desde mediados de los años 90 hasta principios de los 2000, alcanzando valores cercanos al 50%. A partir de ese punto, el porcentaje se estabiliza, manteniendo una cierta constancia y por tanto indicando una progresiva equiparación en las incorporaciones por género.

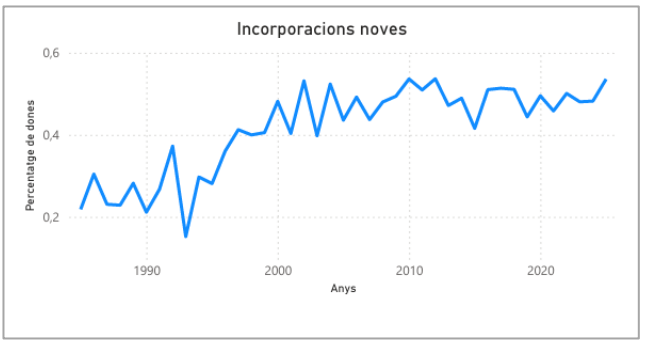

Fig. 10. Gráfico temporal de incorporaciones nuevas

Se incluye también esta visualización, Figura 11, para ilustrar de manera más visual las trayectorias de incorporación de hombres y mujeres, destacando el proceso de convergencia entre ambos a lo largo de los años.

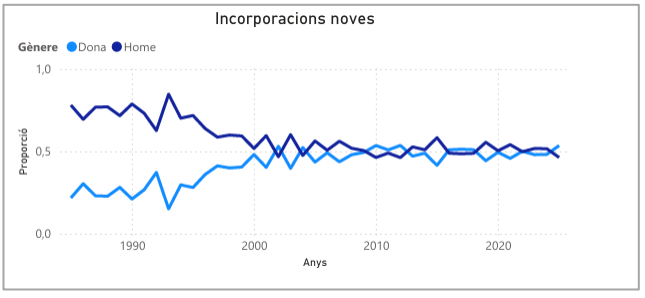

Fig. 11. Gráfico temporal de incorporaciones nuevas de ambos sexos

8. Conclusiones

A lo largo de este proyecto se ha desarrollado un sistema robusto y versátil capaz de adaptar los procesos tradicionales de extracción y transformación de datos (basados en XSLT sobre XML) a un nuevo entorno basado en JSON, acorde con la evolución de la API EGRETA. El trabajo ha permitido diseñar, probar y comparar distintas alternativas tecnológicas incluyendo Python (con Pandas y Polars), Apache NiFi y Saxon—, cada una evaluada en base a criterios como el tiempo de ejecución, la flexibilidad y la reutilización de componentes.

Gracias a estas pruebas, se ha podido comprobar que las soluciones implementadas en Python ofrecen los mejores resultados en términos de rendimiento y capacidad de adaptación, especialmente al aplicar formatos optimizados como NDJSON. Polars, en particular, ha demostrado una gran eficiencia tanto en velocidad como en uso de memoria, siendo especialmente adecuado para conjuntos de datos medianos y grandes.

Sin embargo, es importante remarcar que no se ha podido llevar a cabo la migración completa del sistema actual, ya que la nueva API JSON aún se encuentra en fase de desarrollo y presenta ciertas limitaciones. Entre estas destacan la ausencia de algunos endpoints equivalentes, comportamientos inconsistentes y estructuras de datos

aún inestables, lo cual impide una transición total sin riesgo de pérdida o inconsistencia de datos.

Pese a estas restricciones, el trabajo ha servido para dejar sentadas las bases técnicas y metodológicas que permitirán, en el futuro, una transición fluida y controlada una vez la nueva API esté completamente operativa. Además, se ha demostrado el potencial de aplicar estos datos a casos analíticos como el análisis de la desigualdad de género en el ámbito docente y de investigación, permitiendo extraer conclusiones relevantes, y detectando posibles brechas estructurales.

BIBLIOGRAFIA

[1] PowerData, "¿Qué es un Data Warehouse?", [Online]. Disponible en: https://www.powerdata.es/data-warehouse
[2] UAB, "Documentación de la API EGRETA", [Online]. Disponible en: https://egreta.uab.cat/ws/api/documentation/index.html
[3] UAB, "Guía de API Keys", [Online]. Disponible en: https://egreta.uab.cat/ws/api/documentation/user-guide/api-keys.html
[4] Qalified, "Postman para API Testing", [Online]. Disponible en: https://qalified.com/es/blog/postman-para-api-testing/
[5] Saxonica, "SAXON - XSLT and XQuery Processor", [Online]. Disponible en: https://www.saxonica.com/welcome/welcome.xml
[6] Oracle, "Oracle Data Integrator (ODI)", [Online]. Disponible en: https://www.oracle.com/es/middleware/technologies/data-integrator.html
[7] Microsoft, "¿Qué es Power BI?", *Power BI Documentation*, [Online]. Disponible en: https://learn.microsoft.com/es-es/power-bi/fundamentals/power-bi-overview
[8] Asana, "Diagrama de Gantt: Guía completa", [Online]. Disponible en: https://asana.com/es/resources/gantt-chart-basics
[9] W3Schools, "Pandas Tutorial", *W3Schools*, [Online]. Disponible en: https://www.w3schools.com/python/pandas/default.asp
[10] Polars Documentation, "Performance-focused DataFrame library in Rust/Python", [Online]. Disponible en: https://pola-rs.github.io/polars-book/
[11]Aprender Big Data, "Introducción a Apache NiFi", [Online]. Disponible en: https://aprenderbigdata.com/introduccion-apache-nifi/
[12] JSON Lines, "NDJSON: Newline-Delimited JSON", [Online]. Disponible en: https://jsonlines.org/
[14] Wikipedia, "Distancia de Levenshtein", *Wikipedia, la enciclopedia libre*, [Online]. Disponible en: https://es.wikipedia.org/wiki/Distancia_de_Levenshtein
[14] European Commission, "Guidelines on Data Protection Impact Assessment (DPIA)", [Online]. Disponible en: https://ec.europa.eu/newsroom/article29/items/611236

APÉNDICE

A1. DIAGRAMA DE GANTT

Fases	Marzo				Abril				Mayo				Junio			
	10-mar	17-mar	24-mar	31-mar	07-abr	14-abr	21-abr	28-abr	05-may	12-may	19-may	26-may	02-jun	09-jun	16-jun	23-jun
Análisis de la documentación y procesos																
Desarrollo del sistema																
Evaluación de ODI																
Evaluación de herramientas y optimización																
Pruebas y validaciones																
Visualizaciones																
Implementación final																