Guía para Implementar un Nuevo Proveedor de Datos Biológicos en RareDiseaseFinder

índice

[1. Introducción a la Arquitectura del Sistema 1](#_Toc198553517)

[1.1 Patrón de Diseño Retriever/Parser/Filter/Processor 1](#_Toc198553518)

[2. Implementación Paso a Paso 2](#_Toc198553519)

[2.1 Determinación del tipo de Retriever y creación de clases 2](#_Toc198553520)

[2.2 Estrategia del Retriever 3](#_Toc198553521)

[2.3 Estrategia del Parser 6](#_Toc198553522)

[2.4 Estrategia del Processor 9](#_Toc198553523)

[3. Integración en el sistema de Workflow 13](#_Toc198553524)

[3.1 Workflow Step Personalizado 14](#_Toc198553525)

# 1. Introducción a la Arquitectura del Sistema

## 1.1 Patrón de Diseño Retriever/Parser/Filter/Processor

El sistema utiliza un patrón de cuatro capas para gestionar fuentes de datos externas:

1. Capa Retriever (Client/Scraper):

* Gestiona la comunicación externa (Client si es vía API o Scraper si es vía web scraping).
* Implementa lógica para recuperar datos brutos y verificar conectividad.

1. Capa Parser:
   * Transforma datos brutos en estructuras estandarizadas (DataFrames).
   * Incluye métodos específicos para cada tipo de dato (genes, fármacos, enfermedades).
2. Capa Filter
   * Crea de forma automatizada un json con los filtros definidos para la fuente
   * Mapea filtros de usuario a métodos específicos del Parser.
3. Capa Processor:
   * Coordina el flujo entre Client y Parser.
   * Coordina la comunicación de los filtros entre las capas superiores e inferiores.

En resumen:

El Retriever recupera datos de la fuente externa.

El Parser normaliza los datos según esquemas predefinidos.

El Filter crea filtros personalizados para los datos específicos de la Fuente.

El Processor unifica resultados y los prepara para el orquestador.

# 2. Implementación Paso a Paso

En este apartado explicaremos paso a paso y en detalle cada una de las decisiones e implementaciones que se deben llevar a cabo para crear una nueva fuente en el Sistema correctamente.

## 2.1 Determinación del tipo de Retriever y creación de clases

Debemos determinar si el Retriever será un Client o un Scraper en base a cómo se extrae la información de la fuente. Como en nuestro caso accedemos a través de una API GraphQL, crearemos un OpenTargetsClient.

Ahora, ya sabemos que las clases a crear tendrán la siguiente estructura:

* OpenTargetsClient: hereda de la clase BaseClient
* OpenTargetsParser: hereda de la clase BaseParser
* OpenTargetsProcessor: hereda de la clase BaseProcessor

Texto

El contenido generado por IA puede ser incorrecto.

NOTA: no se implementa una clase específica para los filtros puesto que toda la lógica está ya estructurada en BaseFilter, y nuestro sistema de herencia hace posible el no necesitar ningún ajuste adicional.

## 2.2 Estrategia del Retriever

Esta clase contendrá la lógica de conexión y la obtención de la información de la fuente en

crudo y sin procesar. Nuestra clase Retriever (Cliente) específica de la fuente debe

implementar los métodos abstractos de la clase padre BaseRetriever para funcionar.

**Métodos que deben implementarse obligatoriamente por herencia:**

* **\_ping\_logic()**: método privado que establece la lógica de conexión para las clases derivadas. Debemos encontrar la forma de verificar la conexión a la fuente. Este método debe devolver un valor numérico int que nos devuelva el código de estado de la web o un código de error determinado por nuestro sistema (999) si la conexión no consigue establecerse. Es decir:
  + Código de estado si se establece conexión correctamente
  + 999 si la conexión falla
  + 1001 si el driver falla (para los scrapers)

Para realizar esta implementación, utilizaremos el método estático de la clase

BaseRetriever **\_try\_connection(url)** que determina si una conexión es exitosa

o no. De esta forma, \_ping\_logic se centra exclusivamente en manejar los códigos

de conexión de la fuente en específico.

Texto

El contenido generado por IA puede ser incorrecto.

En nuestro caso de estudio, OpenTargets, tras investigar previamente la fuente, nos damos cuenta de que se accede sus datos a través de una solicitud HTTP post y mediante una consulta GraphQL. Con esto previamente en cuenta, creamos ambos métodos y la clase Retriever ya estaría implementada: Texto

El contenido generado por IA puede ser incorrecto.

## 

## 2.3 Estrategia del Parser

Esta clase contendrá la lógica para transformar los datos crudos obtenidos por el Retriever en estructuras de datos organizadas y utilizables (DataFrames). Nuestra clase Parser específica de la fuente debe heredar de la clase BaseParser para mantener la consistencia con el resto del sistema. Esta clase padre contiene los mecanismos para pasar una lista, un diccionario o un json a dataframe, permitiendo así a las hijas centrarse tan solo en cómo sacar los datos específicos de su fuente:

Texto

El contenido generado por IA puede ser incorrecto.

Para los métodos específicos que debe implementar la clase derivada, aclarar que:

* Todos los métodos deben devolver un pandas Dataframe
* Todos los métodos deben de tener como parámetro de entrada el diccionario de datos sin procesar que le devuelve el cliente.
* Todos los métodos reciben como entrada el mismo diccionario por el punto anterior.
* Algunos métodos pueden incluir una variable filtros con la que cambiar la lógica de procesamiento de información.

En el caso de OpenTargets: Texto

El contenido generado por IA puede ser incorrecto.

Cabe destacar también que intentamos siempre usar la misma estructura: guardamos los elementos que necesitamos en una lista y pasamos dicha lista al método del padre parse\_to\_dataframe.

## 2.4 Estrategia del Processor

Esta clase actúa como coordinador entre el Retriever (que obtiene datos crudos) y el Parser (que transforma estos datos). El Processor implementa la lógica de alto nivel para obtener, filtrar y procesar la información de una fuente específica, siguiendo el patrón arquitectónico Client/Parser/Processor establecido en nuestro sitema.

**Métodos que deben implementarse obligatoriamente por herencia:**

* **\_\_init\_\_()**: Utilizamos el constructor de la clase para definir el retriever y el parser específico de la fuente. Aquí asociaremos las clases creadas anteriormente con el procesador.
* **get\_method\_map()**: Este método abstracto debe ser implementado para proporcionar un mapeo entre los nombres de filtros o banderas usados en el workflow y los métodos correspondientes en el Parser. Debe devolver un diccionario que relaciona las claves (strings utilizados como identificadores en los filtros) con los nombres de los métodos del Parser asociado. Esta lógica nos permite asociar un conjunto de datos a obtener con el método que lo obtiene a bajo nivel.

En OpenTargets:

Texto

El contenido generado por IA puede ser incorrecto.

El diseño del sistema se basa en una sólida abstracción en la clase BaseProcessor, que implementa la mayoría de la funcionalidad genérica, permitiendo que las clases derivadas como OpenTargetsProcessor sean extremadamente concisas. Esta abstracción nos proporciona una estructura muy clara y directa mientras evitamos la compilación de código duplicado.

**Métodos heredados de BaseWorkflowStep:**

* **\_\_init\_\_():** Configura un nuevo paso del workflow con su nombre, descripción y el procesador que usará. Opcionalmente acepta filtros iniciales.
* **get\_status\_code()**: Consulta si el servicio está disponible pidiendo el código de estado al procesador.
* **process():**Ejecuta el paso del workflow, procesando los datos según los filtros configurados.
* **revert()**: Método vacío por defecto, pensado para deshacer las acciones del paso si algo sale mal.

Texto

El contenido generado por IA puede ser incorrecto. Texto

El contenido generado por IA puede ser incorrecto.

# 3. Integración en el sistema de Workflow

Una vez que has implementado correctamente el Retriever, Parser y Processor para tu nueva fuente de datos, el siguiente paso es integrar estos componentes en el sistema de workflow y orquestación del sistema. Esta integración permite que la fuente creada participe en los flujos de trabajo automatizados del sistema.

**¿Qué es un Workflow?**

Un Workflow es una secuencia estructurada de pasos de procesamiento que permiten obtener, transformar y combinar datos biológicos de múltiples fuentes de manera organizada y flexible. Representa un flujo de trabajo completo para el análisis de datos biológicos, desde la consulta inicial hasta la presentación de resultados procesados.

Se podría definir como la hoja de rutas que el sistema sigue para obtener un informe definido. El Workflow define a qué fuentes accederá el sistema, qué información se sacará de cada fuente, y en qué orden, obteniendo así exactamente los elementos de búsqueda que necesita el usuario.

A continuación, definiremos las implementaciones necesarias para utilizar una nueva fuente en un workflow

## 3.1 Workflow Step Personalizado

Los WorkflowSteps son cada uno de los lugares por los que pasa nuestro sistema al obtener

la información. Un WorkflowStep representa el acceso a una de las fuentes de nuestro

sistema.

Al igual que con los procesadores, hemos abstraído todas las funcionalidades comunes en

una clase BaseWorkflowStep, haciendo así muy sencilla la implementación de un nuevo

WorkflowStep, pues sólo se debe crear una clase hija que llame al constructor del padre,

especificando los parámetros de entrada.

En el ejemplo de OpenTargets:

Texto

El contenido generado por IA puede ser incorrecto.

Con estos pasos, la fuente ya estaría implementada y podría usarse dentro de un Workflow definido, o creando un Workflow nuevo.