UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

# Evaluación de la calidad de datos en Grafos de conocimiento

Informe de Proyecto de Grado presentado por

Leandro Alvarez, Ramiro Nieto, Joaquín Sarro

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Lorena Etcheverry
Camila Sanz

Montevideo, 5 de junio de 2025

# Agradecimientos

Queremos tomarnos un momento para agradecer a todas las personas que hicieron posible este trabajo de grado.

En especial, queremos darles las gracias a Lorena y Camila, nuestras tutoras, por su paciencia infinita, su guía y por no dejarnos rendir cuando todo se complicaba. Su apoyo y conocimientos fueron clave para que este proyecto fuese posible.

# Resumen

En la era de la inteligencia artificial, la calidad de los datos determina el éxito de aplicaciones que impactan nuestra vida diaria. Los grafos de conocimiento son pilares de estos sistemas, ya que organizan la información de forma estructurada y basada en relaciones con significado. Sin embargo, su construcción automática o semiautomática introduce errores que comprometen su fiabilidad. El concepto de "basura entra, basura sale" sigue vigente: si un grafo de conocimiento contiene datos inconsistentes o incompletos, cualquier aplicación que lo utilice heredará esos defectos. Por esto, este trabajo se enfoca en medir la calidad en los datos de grafos de conocimiento. Específicamente, en herramientas que permitan medir la calidad de forma confiable y reproducible.

Primero, se investiga el estado del arte en la evaluación de calidad de datos en grafos de conocimiento. Se busca responder a las preguntas "¿qué herramientas se encuentran disponibles?" y, sobre estas, responder "¿qué aspectos de calidad miden?" y "¿cómo miden estos aspectos de calidad?". Para esto, se analizan dimensiones, factores, métricas y métodos presentados en trabajos anteriores y se exploran trabajos que presentan herramientas específicas para este propósito.

Dada la extensa literatura sobre métricas de calidad para grafos de conocimiento, en este trabajo se selecciona un subconjunto de métricas a considerar. Además, se establece un marco teórico que define las dimensiones y factores relevantes identificados en la literatura. Posteriormente, las métricas seleccionadas se clasifican dentro de estas dimensiones y factores, realizando los ajustes pertinentes.

Finalmente, se procede a desarrollar una aplicación web que permite evaluar la calidad de datos en grafos de conocimiento, basándose en las métricas, factores y dimensiones previamente seleccionadas. Esta herramienta ofrece funcionalidades que otras herramientas no poseen, aportando valor a la comunidad, siendo la única aplicación web que permite a los usuarios definir sus propias dimensiones, factores, métricas y métodos, y siendo capaz de solicitar una medición a demanda sobre cualquier grafo con un SPARQL *endpoint* disponible.

**Palabras clave:** Calidad de datos, Grafos de conocimiento, Frameworks para evaluar bases de datos RDF, SPARQL, RDF, OWL, KG

# Índice general

# Capítulo 1

# Introducción

Este capítulo expone la motivación, los objetivos y las conclusiones esperadas del proyecto. Sobre el final, se describe la estructura del documento.

## 1.1. Contexto y motivación

El concepto de "Basura entra, basura sale" que sostiene que un *input* de baja calidad genera un *output* igualmente deficiente, sigue vigente hasta el día de hoy. En la época de la inteligencia artificial, donde los modelos tienen como entrada un conjunto de datos, garantizar la calidad de estos es más crucial que nunca. En la literatura, esto se identifica como un cambio de paradigma: el paso de un enfoque centrado en el modelo a uno centrado en los datos (Mohammed, Ehrlinger, Harmouch, Naumann, y Srivastava, 2024) (Zha y cols., 2025).

La creciente importancia de la calidad de los datos se extiende a diversas áreas, incluyendo los grafos de conocimiento (conocidos en inglés como *Knowledge Graphs* y abreviados como KGs), que han experimentado un aumento en su construcción y uso en los últimos años. Los KGs representan entidades y relaciones del mundo real de manera estructurada y tienen un gran potencial para almacenar conocimiento humano. Muchos grafos de conocimiento a gran escala, como DBpedia[1], YAGO[2] y Wikidata[3], se construyen a partir de diversas fuentes de datos. Sin embargo, al ser extraídos y fusionados de forma automática o semi-automática de estas fuentes, quedan lejos de ser perfectos, variando mucho en su calidad de datos (Xue y Zou, 2022). Estos grafos se han utilizado ampliamente en varias aplicaciones del mundo real, desde sistemas de recomendación, como el modelo KGAT (Knowledge Graph Attention Network for Recommendation) (Wang, He, Cao, Liu, y Chua, 2019), hasta sistemas que responden preguntas como TeBaQA (Vollmers y cols., 2021), y otras tareas relacionadas al dominio de los KGs.

[1] https://www.dbpedia.org/
[2] https://yago-knowledge.org/
[3] https://www.wikidata.org/

Para que aplicaciones como las mencionadas funcionen de la manera prevista, es necesario minimizar los errores en los datos de los grafos de conocimiento que utilizan. Por esto, este trabajo se centra en el desarrollo de herramientas para evaluar la calidad de los KGs, el primer paso en el camino de poder eliminar los errores detectados. El objetivo de esto es poder evaluar cualquier KG y no solo los que cuentan con un grafo de referencia validado (llamado en inglés y conocido en la literatura como "gold standard"). Además, se busca medir la calidad de los datos en un KG sin recurrir a razonadores OWL como HermiT (Shearer, Motik, y Horrocks, 2008), que, a pesar de ser herramientas potentes, sus altos costos computacionales pueden hacer inviable su uso en conjuntos de datos de gran tamaño e incluso en algunos más pequeños. Además, los razonadores solo nos permitirían evaluar la calidad en algunos aspectos del KG.

## 1.2. Objetivos

Este trabajo tendrá tres objetivos principales. En primer lugar, investigar el estado del arte en la evaluación de calidad de datos en KG. Esto incluye analizar dimensiones, factores, métricas y métodos presentados en trabajos anteriores y explorar si existen trabajos que presentan herramientas específicas para este propósito.

En segundo lugar, debido a la extensa literatura sobre métricas de calidad, se busca seleccionar un subconjunto de las métricas referenciadas como punto de partida para ser consideradas en el trabajo. Estas métricas serán elegidas considerando su diversidad en cuanto a dimensiones y factores de calidad, y su viabilidad para ser calculadas mediante consultas en el lenguaje estándar para grafos RDF SPARQL.

Finalmente, el tercer objetivo será desarrollar una herramienta que permita evaluar de manera eficiente la calidad de datos en grafos KG. A su vez, se buscará que la herramienta sea capaz de aportar valor a la comunidad, pudiendo ofrecer funcionalidades que otras herramientas no poseen.

## 1.3. Conclusiones

En primer lugar, se realizó un relevamiento de las herramientas existentes para medir la calidad de bases de datos RDF. Durante esta investigación, se identificaron las principales características y limitaciones de las herramientas disponibles en la literatura.

La principal observación fue la falta de una herramienta que ofreciera una interfaz de usuario capaz de definir con precisión qué aspectos específicos de calidad se desean evaluar y cómo realizar dicha evaluación. También se notó que las herramientas disponibles con interfaz amigable no permitían realizar cálculos a pedido. A su vez, no existe tampoco ninguna herramienta con una aplicación web que permita agregar nuevas dimensiones, factores ni métricas a la plataforma, lo que quita flexibilidad a la hora de crear los modelos de calidad

para evaluar el grafo.

En cuanto al estado del arte, se tomaron en cuenta múltiples trabajos de investigación relacionados al tema, en los cuales se presentan diferentes dimensiones, factores, métricas y métodos. Por esto, se realizó un análisis exhaustivo dentro de cada trabajo sobre cuáles métricas podían ser consideradas y cuáles descartadas, basándose en el alcance de la herramienta y en cómo las mismas son calculadas en el trabajo referenciado. Las métricas analizadas también fueron categorizadas dentro de las dimensiones y factores previamente seleccionados.

Finalmente, a partir de estos puntos, se desarrolló una aplicación que integra estas funcionalidades, abordando las necesidades detectadas durante el relevamiento.

## 1.4. Estructura del documento

Este documento se compone de siete capítulos, en los que se abordan distintos aspectos relevantes al proyecto desarrollado. A continuación, se presenta una descripción concisa del contenido de cada uno.

En el capítulo 2 se presentan conceptos esenciales para poder entender el trabajo realizado. Se divide en dos secciones: en la primera se introduce el concepto de grafos de conocimiento, describiendo todos sus conceptos relacionados, como los modelos de datos RDF, las ontologías y OWL, y el lenguaje utilizado para las consultas SPARQL. En la segunda Sección se presenta el concepto de modelo de calidad en bases de datos de grafo, donde se habla sobre dimensiones de calidad y del concepto de granularidad.

En el capítulo 3 se desarrolla sobre otros trabajos que plantean herramientas y métricas similares a las que se van a buscar implementar en este. A su vez, sobre el final, se realiza un análisis comparativo entre todas las herramientas de estos trabajos.

En el capítulo 4, llamado Diseño de un modelo de calidad de datos para grafos RDF, se presenta el trabajo de investigación propio, donde se determina qué dimensiones, factores, métricas, métodos y granularidades se van a utilizar en la herramienta a desarrollar y por qué son estas las seleccionadas.

Luego, en el capítulo 5 se describe la herramienta propuesta, donde se detallan los requerimientos, la arquitectura y el flujo de la aplicación. Se explican decisiones de diseño tomadas y partes clave de la implementación, así como información relevante relacionada con el desarrollo de software.

Después, en el capítulo 6 referente a las pruebas y experimentación, se muestran modelos de calidad instanciados con la herramienta y se hace un análisis de los resultados.

Por último, en el capítulo 7 se presentan las conclusiones del trabajo, haciendo un análisis comparativo de la herramienta desarrollada con las herramientas mencionadas en el capítulo 3, así como el planteamiento de posibles mejoras a la herramienta y trabajo a futuro.

# Capítulo 2

# Conceptos preliminares

En este capítulo se discuten los conceptos y definiciones necesarias para entender el contexto de trabajo, así como los aportes realizados en este informe.

Primero se introduce el concepto de grafo de conocimiento, su modelo de datos, los vocabularios que permiten definir el conocimiento que almacenan y el lenguaje de consulta que permite extraer dicho conocimiento. Luego, se presentan conceptos de calidad de datos, en particular la noción de modelo de calidad y algunas dimensiones de calidad de datos que son relevantes para este trabajo.

## 2.1. Grafos de Conocimiento

> "Un grafo de conocimiento es un grafo de datos cuyo propósito consiste en acumular y comunicar conocimiento del mundo real, cuyos nodos representan entidades de interés y cuyas aristas representan relaciones entre las entidades." (Hogan y cols., 2021)

La definición provista por los autores mencionados afirma que un grafo de conocimiento es un grafo de datos. Esto significa que los datos son abstraídos a un grafo, que puede ser un grafo dirigido con aristas etiquetadas, un grafo de propiedades, entre otros. Este trabajo se basa en el primero, que es el modelo implementado por el Marco de Descripción de Recursos (por su sigla en inglés RDF - *Resource Description Framework*) que se describe en la Sección 2.1.1.

Los autores de (Hogan y cols., 2021) definen la existencia de varios tipos de esquemas que pueden ser usados para proveer una estructura de alto nivel o definir semánticas que el grafo debería seguir. Entre estos esquemas se destaca el esquema semántico, que permite dar significado a ciertos términos de alto nivel que facilitan el razonamiento dentro del grafo usando dichos términos. Un ejemplo de vocabulario de este estilo es RDFS, tal como se menciona en la Sección 2.1.1.

Por otra parte, las ontologías son construcciones lógicas que permiten definir a mayor profundidad restricciones y reglas semánticas que deben cumplirse en

un grafo de conocimiento. En particular, en este trabajo se utiliza el estándar de la W3C OWL (W3C, 2004), como se detalla en la Sección 2.1.2.

### 2.1.1. Modelo de datos RDF

Un grafo, en el sentido más general, es una colección de nodos que pueden estar conectados mediante aristas. RDF nace como un estándar desarrollado por la *World Wide Web Consortium* (W3C) para describir recursos web de manera estructurada. Los grafos RDF son conjuntos de triplas <sujeto-predicado-objeto>, donde *sujeto* es el nodo fuente, *objeto* es el nodo destino y *predicado* es la arista que los conecta. Cada conjunto de datos RDF está compuesto por un grafo *default* y cero o más grafos nombrados.

Los elementos de una tripla RDF pueden ser IRIs (*Internationalized Resource Identifier*), nodos blancos o valores literales. Las IRIs y los literales representan una entidad del universo (el dominio en cuestión), a las que se refiere como recursos. Las IRIs hacen referencia a recursos, mientras que los literales representan valores constantes que pertenecen a un determinado *datatype* (string, número, fecha, etc.). Los predicados son IRIs que representan una propiedad, es decir, un recurso que se interpreta como una relación binaria. Por último, un nodo blanco representa la existencia de un recurso sin nombrarlo explícitamente.

Una declaración RDF (*RDF statement*) expresa que cierta relación, indicada por el predicado, es verdadera entre los recursos sujeto y objeto. Además, cumple ciertas reglas en cuanto a su estructura:

- el sujeto es una IRI o nodo blanco
- el predicado es una IRI
- objeto es una IRI, nodo blanco, literal o término

**RDF Dataset:** Un *dataset* RDF permite organizar una colección de grafos RDF, y se compone de un grafo *default* y uno o más grafos nombrados. El grafo *default* no tiene un nombre, y puede ser vacío. Los grafos nombrados son un par compuesto por el nombre del grafo (una IRI o nodo blanco) y un grafo RDF, donde cada nombre es único para el *dataset*.

**RDF Schema:** Si bien RDF permite modelar la información como un grafo, no alcanza para asignar un significado a los conceptos que se almacenan en él. Con este objetivo, se define el vocabulario **RDFS** (*RDF Schema* (W3C, 2014c)), una extensión semántica de RDF que permite describir clases de recursos y sus relaciones. De manera simplificada, RDFS permite especificar:

- Clases, que categorizan los recursos. Un ejemplo de clase es `Persona`, que incluiría a todas las instancias de este tipo. *RDFS* permite la jerarquización de clases mediante el uso de la propiedad `rdfs:subClassOf`. Por ejemplo, si decimos que la clase `Estudiante` es subclase de `Persona`, entonces los individuos pertenecientes a la extensión de la clase `Estudiante`

(es decir, los recursos que son instancias de la clase) deben ser un subconjunto de los individuos pertenecientes a la extensión de la clase `Persona`.

- Propiedades, que permiten definir relaciones entre recursos. Por ejemplo, podría definirse la propiedad `edad` para instancias del tipo `Persona` que describa la relación entre una persona y su edad. Al igual que en el caso de las clases, las propiedades también permiten la jerarquización. Por ejemplo, podría decirse que la relación `esEmpleadoDe` es una subpropiedad de la relación `trabajaEn`, que denota la empresa para la que trabaja una persona.

- Restricciones, que se especifican utilizando propiedades ya definidas por el vocabulario, pero que merecen la pena mencionar de manera aparte debido a la semántica asociada a ellas. Algunas de ellas son:

  - `rdfs:range`, que permite definir el rango de una propiedad. Esto es, permite afirmar que "los valores de una propiedad son instancias de una o más clases."(W3C, 2014c)
  - `rdfs:domain` permite definir el dominio de una relación. Es decir, permite afirmar que "cualquier recurso que tiene una propiedad dada es instancia de una o más clases."(W3C, 2014c)
  - `rdf:type` es "utilizada para declarar que un recurso es una instancia de una clase."(W3C, 2014c)

Las descripciones anteriores no abarcan la totalidad del vocabulario de modelado de datos RDFS. Si el lector está interesado en una explicación exhaustiva de los conceptos implicados, puede referirse a su documentación (W3C, 2014c). En el Listing 2.1, presentamos un ejemplo de definición de datos RDF utilizando el vocabulario RDFS, escrito en el lenguaje Turtle (W3C, 2014b) para RDF:

```
1   @prefix ex: <http://example.org/> .
2
3   # Se definen las clases
4   # La letra "a" usada como predicado es un atajo para la
    propiedad rdf:type
5   ex:Persona a rdfs:Class .
6   ex:Estudiante a rdfs:Class ;
7       rdfs:subClassOf ex:Persona .
8   ex:Empresa a rdfs:Class .
9
10  # Definir las propiedades
11  ex:edad a rdf:Property ;
12      rdfs:domain ex:Persona ;
13      rdfs:range xsd:integer .
14
15  ex:trabajaEn a rdf:Property ;
16      rdfs:domain ex:Persona ;
17      rdfs:range ex:Empresa .
18
19  # esEmpleadoDe es subpropiedad de trabajaEn
20  ex:esEmpleadoDe a rdf:Property ;
```

```
21          rdfs:subPropertyOf ex:trabajaEn ;
22          rdfs:domain ex:Persona ;
23          rdfs:range ex:Empresa .
24
25      # Se agregan instancias de personas
26      # @en es una tag que identifica el lenguaje (ingles en este
        caso)
27      ex:Juan rdf:type ex:Persona ;
28          ex:nombre "Juan Perez"@es ;
29          ex:edad 30^^xsd:int .
30      ex:Pedro rdf:type ex:Persona .
31          ex:nombre "Pedro Jimenez"@en ;
32          ex:edad 26^^xsd:int .
33      ex:Martin rdf:type ex:Persona .
34          ex:nombre "Martin Perez"@en ;
35          ex:edad 17^^xsd:int .
```

Listing 2.1: Ejemplo de uso de RDFS en el lenguaje Turtle

### 2.1.2. Ontologías y OWL

> "En el contexto de computación, una ontología es una representación concreta y formal de lo que ciertos términos significan en el ámbito en que se utilizan."(Hogan y cols., 2021)

Las ontologías tienen la particularidad de poder ser representadas como grafos de datos. En la práctica, OWL se usa como vocabulario para extender los vocabularios RDF y RDFS mencionados anteriormente, permitiendo definir jerarquías de clases más expresivas y restricciones lógicas adicionales a las que se permiten en RDFS.

Hay dos versiones de este lenguaje: OWL (W3C, 2004) y OWL 2 (W3C, 2012a). A su vez, cada lenguaje define sub-lenguajes. En el caso de OWL se definen los sub-lenguajes OWL Lite, OWL DL y OWL Full. En OWL 2 los sub-lenguajes se llaman perfiles; esos son OWL 2 EL, OWL 2 QL, y OWL 2 RL. OWL 2 es compatible hacia atrás con OWL, agregando nuevas funcionalidades (como la posibilidad de declarar propiedades disjuntas) construidas a partir de lo que permite su antecesor.

En este trabajo, como se verá más adelante, se hace uso de la expresividad ofrecida por OWL 2, en particular al definir los métodos para obtener mediciones de las métricas obtenidas. A continuación se muestran algunos ejemplos de la semántica que OWL permite expresar:

- `owl:ObjectProperty`, subclase de `rdf:Property`, permite establecer que la propiedad relaciona instancias de clases.
- `owl:DatatypeProperty`, también subclase de `rdf:Property`, permite establecer que la propiedad relaciona instancias de una clase con un valor de dato (*datatype*).
- `owl:disjointWith` permite declarar que dos clases son disjuntas.

- `owl:propertyDisjointWith` permite declarar que dos propiedades (ya sean *data* u *object properties*) son disjuntas.
- `owl:sameAs` permite establecer que dos individuos (instancias de una clase) son en realidad el mismo.

**Razonadores OWL:** Los razonadores OWL son motores lógicos que utilizan las definiciones formales de una ontología (clases, propiedades, restricciones) para inferir nuevo conocimiento implícito y garantizar la consistencia semántica del grafo. Estas herramientas son fundamentales en escenarios que requieren validación lógica exhaustiva, como la detección de contradicciones en taxonomías (W3C, 2012a).

Como se detalló en la Sección 1.1, en este trabajo se opta por no utilizar razonadores OWL para evaluar la calidad del grafo. Si bien estos sistemas son capaces de identificar inconsistencias o equivalencias entre entidades (por ejemplo, mediante axiomas como `owl:sameAs`), su alto costo computacional, especialmente en grafos con miles de triplas, dificultaría la aplicación práctica de métricas propuestas más adelante. Además, diversos estudios han señalado que muchos razonadores OWL presentan limitaciones significativas en términos de rendimiento y escalabilidad. En particular, la falta de optimización para arquitecturas modernas, como la computación en paralelo o el uso de GPU, ha impedido que estos sistemas se adapten eficientemente a escenarios de gran escala (Abicht, 2023). Asimismo, varias herramientas ampliamente utilizadas, como HermiT, han dejado de recibir mantenimiento activo, lo que dificulta su integración en entornos industriales donde la estabilidad y el soporte son cruciales (Abicht, 2023). Esta decisión no excluye el uso de la expresividad de OWL para modelar restricciones (como `owl:disjointWith` o `owl:propertyDisjointWith`), sino que enfoca el análisis en métodos verificables directamente sobre la estructura del grafo, implementados en SPARQL.

### 2.1.3. Lenguaje de consultas SPARQL

SPARQL (SPARQL *Protocol and RDF Query Language*) (W3C, 2013) es un lenguaje de consulta diseñado para expresar consultas sobre datos RDF, ya sea almacenados de forma nativa o expuestos como RDF mediante *middleware*. Soporta características avanzadas como agregaciones, subconsultas, negaciones, creación de valores mediante expresiones y otras. Los resultados de una consulta SPARQL pueden representarse como conjuntos de datos tabulares o como grafos RDF.

La sintaxis de SPARQL puede resultar familiar debido a que utiliza una estructura y escritura similar a la de SQL en el mundo relacional. Sin embargo, las consultas SPARQL se basan en la correspondencia de los datos del grafo con patrones de grafo.

Los patrones de grafo (del inglés, *graph patterns*) son un conjunto de reglas que permiten encontrar correspondencias entre el patrón definido y los datos del grafo. SPARQL define varios patrones que pueden combinarse para efectuar

consultas cada vez más complejas. Estos son: patrones de grafo básicos, de grupos, opcionales, alternativos y patrones de grafos nombrados. No se entrará en detalle de cada uno de ellos. Una explicación exhaustiva de los mismos puede encontrarse en (W3C, 2013). En el contexto de este trabajo, basta con tener cierto conocimiento básico para definir patrones y entender los resultados que generan.

**Patrón de grafo básico:** Consiste en una secuencia de patrones de triplas y filtros opcionales.

> "Los patrones de tripla son como triplas RDF excepto que sujeto, predicado y objeto pueden ser variables. Un patrón de grafo básico se corresponde a un subgrafo de los datos RDF cuando los términos RDF de dicho subgrafo pueden ser sustituidos por variables, y el resultado es un grafo RDF equivalente al subgrafo."(W3C, 2013)

La manera de definir un patrón como el mencionado en una consulta SPARQL es mediante la cláusula **WHERE**, mientras que los filtros se aplican mediante restricciones (*constraints*), que se expresan con la palabra clave **FILTER**.

**Patrón de grafo de grupo:** En una consulta SPARQL, un patrón de grafo de grupo se encuentra delimitado por paréntesis . Puede estar formado por uno o más patrones de grafo básicos (incluyendo filtros) y/o el patrón vacío ({ }, que se corresponde a cualquier grafo).

**Secuencia de soluciones:** Los resultados arrojados por una consulta SPARQL son un conjunto de datos, llamados **secuencia de soluciones**, que denotan la manera en que los datos del grafo RDF se corresponden al patrón. Esta secuencia puede ser vacía o contener una o múltiples soluciones. Cada solución en la secuencia de soluciones es una función que asocia las variables de la consulta con los datos que se corresponden a cada variable en el subgrafo de solución. El uso de la palabra clave **FILTER** permite restringir las soluciones del grupo en el que aparece a aquellas que cumplan la condición contenida en el filtro. Además, SPARQL define modificadores que permiten alterar de alguna manera la secuencia de solución devuelta por una consulta. En este trabajo se hace uso del modificador **DISTINCT**, que asegura que las soluciones contenidas en la secuencia sean únicas.

Además de los patrones de grafos, SPARQL define operaciones (denominadas formas de consulta) que se pueden efectuar sobre dichos grafos, y que se asocian a palabras reservadas del lenguaje. Ellas son: **SELECT**, **ASK**, **CONSTRUCT** y **DESCRIBE**. En el contexto de este trabajo, nos interesa entender las primeras dos.

**ASK** es la más simple de las formas de consulta. Devuelve un valor booleano que será verdadero si el patrón de grafo de consulta se corresponde con un subgrafo de los datos.

**SELECT** devuelve las variables ligadas a la consulta (o un subconjunto de ellas). Esto es, para cada variable devuelve un par $(var, dato)$ donde $var$ es la variable y $dato$ el dato que corresponde a dicha variable en la solución. Además de permitir seleccionar las variables del *pattern matching* asociado a una consulta, SELECT permite incluir variables nuevas mediante el uso de expresiones y el uso de la palabra clave *AS*.

Recapitulando lo visto hasta ahora, el Listing 2.2 muestra una consulta SPARQL que consta únicamente de un patrón de grafo básico.

```
PREFIX ex: <http://example.org/>
SELECT ?persona ?nombre ?edad WHERE {
    ?persona a ex:Persona .
    ?persona ex:name ?nombre .
    ?persona ex:edad ?edad .
    FILTER(?edad > 18)
    FILTER(lang(?nombre) = "en")
}
```

Listing 2.2: Ejemplo de una consulta utilizando un patrón de grafo básico. En este caso person name y age son variables

Esta consulta obtiene instancias de la clase `Persona`, que tienen propiedades `nombre` y `edad`, como se definió en el Listing 2.1, y restringe los resultados a las instancias que cumplan que la edad es mayor a 18 y la `tag` de lenguaje del nombre es inglés (denotado como "en"). Si esta consulta se ejecuta contra el *dataset* definido en el Listing 2.1, se obtendría como respuesta solo la instancia de nombre "Pedro Jimenez". Las otras dos personas se filtran: "Martin Perez"debido a que no es mayor a 18 años y "Juan Perez"porque la tag de lenguaje usada para su nombre es español (`@es`) y no inglés.

**Agregaciones y funciones:** Otro concepto importante que cabe mencionar en el contexto de este trabajo es el de las agregaciones. De manera similar a como se da en SQL, SPARQL define cláusulas **GROUP BY** para agrupar resultados dentro de una solución y **HAVING** para trabajar con grupos dentro de las soluciones. A su vez, existen funciones que trabajan sobre datos agregados como la función **COUNT**, que permite contar la cantidad de resultados que se corresponden a un patrón de consulta. No entraremos en detalle de cómo funcionan cada uno, dado que el funcionamiento es intuitivamente análogo al que se da en SQL. Si el lector quiere entender más a fondo estos conceptos, puede referirse a (W3C, 2013).

**Patrones de grafo opcionales:** En SPARQL, la palabra clave **OPTIONAL** permite obtener datos que pueden o no existir en el *dataset* RDF. Si existe

una correspondencia para el patrón opcional, las variables contenidas en este se vinculan a los datos encontrados. En caso contrario, la vinculación no se da, pero la solución no se rechaza debido a esto, como sucede con los patrones básicos y de grupos.

### 2.1.4. Linked Data y Open Linked Data

El movimiento *Linked Data* promueve la publicación e interconexión de datos en la web utilizando estándares abiertos, con el objetivo de mejorar la accesibilidad y la reutilización de la información. Tim Berners-Lee, en el contexto del *World Wide Web Consortium* (W3C) (Tim Berners-Lee, 2006), propuso cuatro principios fundamentales:

1. Usar Uniform Resource Identifiers (URIs) para identificar recursos de manera única.
2. Usar HTTP para permitir la localización e interacción con esos recursos.
3. Proporcionar datos en formatos estándar como RDF y SPARQL.
4. Incluir enlaces a otros datos para mejorar la conectividad y el descubrimiento de información.

Cuando estos principios se aplican a datos abiertos y accesibles públicamente, se habla de *Open Linked Data*, lo que fomenta la interoperabilidad y el libre acceso a la información.

En este contexto, un SPARQL *endpoint* es una interfaz web que permite la ejecución de consultas SPARQL sobre un conjunto de datos RDF. Estos *endpoints* funcionan como puntos de acceso a bases de conocimiento estructuradas y pueden ofrecer distintos niveles de acceso: por ejemplo, algunos son públicos y permiten consultas sin autenticación, mientras que otros pueden requerir usuario o imponer ciertos límites para controlar el rendimiento y el acceso. A través del protocolo HTTP, se pueden enviar consultas SPARQL y recibir respuestas en diversos formatos (XML, JSON, CSV, entre otros), lo que facilita la integración y reutilización de los datos en diferentes aplicaciones.

## 2.2. Calidad de datos en bases de datos de grafos

La calidad de datos es multi-dimensional. Es decir, se caracteriza en base a dimensiones que permiten clasificar y modelar aspectos específicos de la calidad (y también los potenciales problemas asociados a dichos aspectos). Un modelo de calidad de datos define qué características de calidad se manejan, sobre qué datos aplican y cómo se miden dichas características. Su objetivo es especificar los aspectos de calidad relevantes para el escenario en que se utilice el sistema de información, y el método de medición elegido para cada uno. De aquí se desprende que un modelo de calidad de datos es dependiente del escenario de uso, o dicho de otra manera, del contexto. Por ello es que muchas veces suele

encontrarse en la bibliografía la definición de calidad de datos como “adecuación para el uso” (*fitness for intended use*) (Carlo Batini, 2016).

Como se presenta en el curso de Calidad de Datos e Información (Lorena Etcheverry, Adriana Marotta, Camila Sanz, Sebastián García, 2025), y se muestra en la Figura 2.1, el modelo de calidad de datos consiste en:

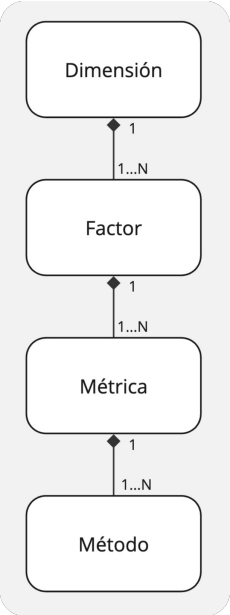



Figura 2.1: Jerarquía de conceptos de calidad

- Dimensiones: una dimensión representa a alto nivel la faceta de calidad que se evalúa.
- Factores: cada dimensión puede refinarse en uno o más factores que representen aspectos individuales de la dimensión.
- Métricas: una métrica define la forma de medir un factor de calidad. Consisten de un nombre, descripción, unidad de medida y granularidad, que es el nivel de detalle del dato sobre el que se mide la calidad, como se describe en la Sección 2.2.2.
- Métodos de medición: cada método define el proceso que implementa una métrica. Se encarga de obtener medidas pertinentes a la métrica a la que pertenece. Vale notar que una misma métrica puede ser evaluada mediante más de un método.

Pueden encontrarse en la literatura muchas definiciones y clasificaciones distintas de posibles dimensiones y factores de calidad. Sin embargo, no todas son

pertinentes a grafos de conocimiento, sino que abarcan múltiples modelos de datos (de clave-valor, grafos, relacional, documental, etc.). Por esta razón, se establecen en la siguiente Sección las dimensiones y factores considerados, así como sus definiciones en el contexto de un grafo de conocimiento.

### 2.2.1. Dimensiones de calidad

Dependiendo del autor, pueden encontrarse diversas dimensiones de calidad, así como distintas definiciones de cada una de ellas. Para poder medir calidad, primero hay que decidir qué es lo que se va a evaluar. En esta Sección se analizan algunas de las definiciones que se utilizan más adelante para definir el modelo de calidad de la aplicación implementada.

**Exactitud:** La exactitud "refiere al grado en que las entidades y relaciones representan correctamente los fenómenos de la vida real." (Hogan y cols., 2021)

Varios autores coinciden en la existencia de una componente estructural y otra temporal en la caracterización de la exactitud (Carlo Batini, 2016). Dentro de la componente estructural, destacan los factores **Exactitud Sintáctica** y **Exactitud Semántica**, mientras que la componente temporal consiste del factor de **Oportunidad**. En este informe se considera, siguiendo lo establecido en el curso de Calidad de Datos e Información del Instituto de Computación de la Facultad de Ingeniería (Lorena Etcheverry, Adriana Marotta, Camila Sanz, Sebastián García, 2025), la componente temporal como parte de una dimensión individual, que engloba varias perspectivas temporales de la calidad de datos. Esta dimensión, denominada **Frescura**, se describe más adelante en esta Sección.

La **Exactitud Sintáctica** busca establecer si los conceptos modelados en el grafo corresponden a conceptos válidos del dominio, independientemente de si reflejan o no la realidad. A modo de ejemplo, si un valor v, que debería representar el nombre de una persona, es v = "Juan", el valor es sintácticamente correcto incluso si la persona a la que en realidad quería referir es v' = "Pedro".

La **Exactitud Semántica** mide qué tan bien representada está la realidad por los datos almacenados. Es una medida de la cercanía entre el valor v y el valor verdadero v' que busca representar.

**Completitud:** La completitud "refiere al grado en el cual toda la información requerida está presente en el grafo."(Carlo Batini, 2016)

Tanto en (Carlo Batini, 2016) como en (Hogan y cols., 2021), los autores coinciden en la definición de esta dimensión. Sin embargo, en (Hogan y cols., 2021), la consideran parte de una dimensión mayor, llamada cobertura. En este informe se opta por seguir la primera referencia, considerando la dimensión como Completitud.

Además, en los trabajos mencionados existe un acuerdo en la distinción de cuatro factores de la completitud: completitud de esquema, completitud de

propiedad, completitud de población y completitud de vínculos. Esta última debe ser considerada solo en el contexto de *Linked Data*, es por ello que no se incluye como uno de los factores considerados en este informe.

La **Completitud de Esquema** "refiere al grado en el cual las clases y propiedades de un esquema están representadas en el grafo". (Carlo Batini, 2016)

La **Completitud de Propiedad** "refiere a la proporción de valores faltantes para una propiedad específica.". (Carlo Batini, 2016)

La **Completitud de Población** "refiere al porcentaje de todas las entidades del mundo real para un tipo particular que están representadas en el grafo".(Carlo Batini, 2016)

**Frescura:** La frescura refleja "el grado al cual el grafo de conocimiento se mantiene actualizado con el estado del mundo real."(Hogan y cols., 2021)

La **Actualidad** busca medir qué tan recientes son los datos del sistema. El objetivo de este factor es medir el desfasaje, si existe, entre los datos almacenados en el grafo con los datos de la realidad. (Lorena Etcheverry, Adriana Marotta, Camila Sanz, Sebastián García, 2025)

El factor de **Oportunidad** busca establecer si los datos del sistema están disponibles al momento en que se los necesita. Un dato puede ser actual, pero no ser ingresado a tiempo para el uso específico que se le quiere dar. Un ejemplo de esto es una actualización de *stock* que se ingresa en el grafo luego de que un consumidor ya efectuó un reporte de las compras recientes en base a la cantidad de *stock* disponible. (Lorena Etcheverry, Adriana Marotta, Camila Sanz, Sebastián García, 2025)

La **Volatilidad** caracteriza la frecuencia con que los datos cambian a lo largo del tiempo. Interesa medir el lapso de tiempo en que los datos son válidos. Por ejemplo, una fecha de nacimiento tiene volatilidad cero, mientras que la cantidad de stock del producto de una empresa tiene una alta volatilidad. (Lorena Etcheverry, Adriana Marotta, Camila Sanz, Sebastián García, 2025)

**Consistencia:** La consistencia mide la correctitud lógica del grafo de conocimiento. Según (Carlo Batini, 2016), que el grafo sea consistente "significa que la base de conocimiento se encuentra libre de contradicciones... con respecto a una representación particular del conocimiento y mecanismos de inferencia".

Los autores de (Hogan y cols., 2021) consideran a la consistencia como un factor dentro de una dimensión mayor de **Coherencia**. Dentro de esta coherencia, distinguen también el factor de **Validez**, que mide la satisfacción de restricciones como las capturadas por *shape expressions* (Baker y Prud'hommeaux, 2022). Sin embargo, dado que en el proyecto implementado no se tienen en consideración dichas expresiones, no se considera este factor, y el concepto de coherencia se ve entonces reducido al de consistencia.

**Concisión:** La concisión evalúa "la no inclusión de elementos irrelevantes al dominio o redundantes en el grafo."(Hogan y cols., 2021)

La **Concisión intensional** hace referencia a la minimización de redundancia a nivel de esquema (propiedades, clases, etc.).

La **Concisión extensional** refiere nuevamente a la minimización de redundancia, pero esta vez a nivel de los elementos de datos, es decir, las instancias.

Además, tanto (Carlo Batini, 2016) como (Hogan y cols., 2021) destacan la **Concisión Representacional** como un tercer factor que "refiere al grado al que el contenido es representado de forma compacta en el grafo de conocimiento.", y este factor puede considerarse también a nivel de esquema o datos. De aquí se desprenden entonces otros dos factores, la concisión representacional intensional y la concisión representacional extensional (Hogan y cols., 2021).

La **Concisión Representacional Intensional** hace referencia al grado en que el contenido del grafo, a nivel de esquema, se representa de manera compacta.

La **Concisión Representacional Extensional** es análoga a su par intensional, pero considerada a nivel de datos.

**Facilidad de comprensión:** la facilidad de comprensión busca "medir el grado en que los datos contenidos en el grafo pueden ser comprendidos sin ambigüedad y utilizados por humanos."(Hogan y cols., 2021)

### 2.2.2. Granularidad de la medición

No se encontraron trabajos que definieran granularidades en el contexto de un grafo de conocimiento. Sin embargo, en (Emilio Cristalli y Marotta, 2018) se define la granularidad de manera general, así como las granularidades posibles en el contexto de bases de datos de documentos. En este trabajo se adopta dicha definición general de granularidad, para poder definir en la Sección 4 las granularidades posibles en nuestro contexto.

Definimos granularidad entonces como "el nivel de detalle del *dataset* al cual la métrica se aplica".

# Capítulo 3

# Trabajos relacionados

Los estudios analizados abordan la medición de calidad en datos RDF con diferentes enfoques: monitoreo de *endpoints* SPARQL, metodologías basadas en *tests*, métricas estructurales y *frameworks* extensibles. A continuación, se describen las contribuciones que consideramos más relevantes en orden cronológico.

## 3.1. Test-driven evaluation of linked data quality - 2014

Este artículo (Kontokostas y cols., 2014) presenta una metodología para evaluar la calidad de datos en *Linked Data*, inspirada en la práctica de *Test Driven Development* (TDD). El método se basa en la definición de patrones de testeo de calidad de datos (*Data Quality Test Pattern*, DQTP).

Un DQTP consiste en una plantilla SPARQL que contiene variables tipadas, que pueden ser sustituidas al momento de evaluar la consulta sobre el grafo deseado. Cada instanciación de un DQTP da lugar a un *Test Pattern Binding*, que es una tripla $(\sigma, S, C)$ donde $\sigma$ es un mapeo de variables a valores válidos para dichas variables, $S$ es una plantilla SPARQL y $C$ es una clasificación para identificar el error detectado. Al aplicar un mapeo $\sigma$ a una plantilla $S$ obtenemos un *Data Quality Test Case*, es decir, una consulta que puede ser ejecutada, y para la cual cada resultado indica una violación del test (y por consiguiente de la métrica que representa).

Un ejemplo de un DQTP que devuelve los recursos para los cuales no se cumple cierta relación de orden entre los valores asociados a un par de propiedades dadas. La plantilla SPARQL $S$ se muestra en el Listing 3.1

```
SELECT ?s WHERE {
    ?s %%P1%% ?v1 .
    ?s %%P2%% ?v2 .
    FILTER ( ?v1 %%OP%% ?v2 )
}
```

Listing 3.1: Plantilla SPARQL

En este caso, tenemos las variables $P1$, $P2$, y $OP$. Un posible ejemplo de *Test Case* podría ser el que se muestra en el Listing 3.2.

```
SELECT ?s WHERE {
    ?s dbo:startDate ?v1 .
    ?s dbo:endDate ?v2 .
    FILTER (?v1 > ?v2)
}
```

Listing 3.2: *Test Case*

donde $\sigma$ queda definido como se indica en el Listing 3.3.

```
P1 -> dbo:startDate
P2 -> dbo:endDate
OP -> ">"
```

Listing 3.3: Ejemplo de $\sigma$ para un *Test Case*

En el *Test Case* presentado, se verifica que la fecha de comienzo de un recurso no sea posterior a la fecha en que terminó en DBPedia, lo cual indicaría una inconsistencia en los datos. Los resultados encontrados son eventos con fechas erróneas.

Además de lo anterior, el método define autogeneradores de test ($TAG$), que utilizan el esquema del grafo para autogenerar tests de calidad. En el artículo puede encontrarse una descripción detallada del funcionamiento de los autogeneradores, tanto para la detección de tests como para la instanciación de los mismos. Esta cualidad, aunque interesante, no entra dentro del alcance del trabajo actual.

Además del planteo de la metodología, los mismos autores presentan una aplicación para ejecutarla: RDFUnit. Esta herramienta automatiza la ejecución de los DQTPs. Toma las plantillas SPARQL parametrizadas, las instancia con valores específicos del dominio (por ejemplo, propiedades de DBpedia como `dbo:birthDate`), y ejecuta las consultas resultantes sobre el grafo RDF objetivo.

## 3.2. Luzzu – A Framework for Linked Data Quality Assessment - 2016

La aplicación desarrollada en este trabajo busca proporcionar una interfaz extensible para la evaluación de la calidad de los Datos Enlazados. Permite a los usuarios definir métricas personalizadas utilizando clases de Java tradicionales o usando *Luzzu Quality Metric Language*, un lenguaje específico de Luzzu (Debattista, Auer, y Lange, 2016) que permite definir métricas de manera declarativa, utilizando notaciones abreviadas y abstracciones que mejoran la expresividad y la comprensión. Además, la aplicación incluye una biblioteca de métricas predefinidas que los usuarios pueden utilizar para evaluar conjuntos de *linked data* de manera inmediata, lo que reduce la carga de trabajo inicial y acelera el proceso de evaluación.

La aplicación ofrece un algoritmo de clasificación personalizable que permite a los usuarios priorizar métricas de calidad según sus necesidades específicas,

facilitando así la identificación de conjuntos de datos que son más adecuados para su uso previsto.

Sin embargo, esta aplicación fue desarrollada en 2016 y su última actualización fue en 2021, estando ahora sin mantenimiento activo. Por otra parte, su despliegue presenta ciertas complicaciones, ya que no se trata de una aplicación web ni se puede instalar de forma fácil, lo que dificulta su uso de forma inmediata. El equipo intentó hacer uso de Luzzu, pero el *build* de la dependencia requerida hdt-java falla, imposibilitando la instalación de la herramienta.

## 3.3. SPARQLES: Monitoring Public SPARQL Endpoints - 2017

El artículo aborda los desafíos de confiabilidad y calidad en los *endpoints* SPARQL públicos, recursos críticos para acceder a *linked data* en la web semántica. Los autores destacan que, aunque estos servicios permiten consultar billones de datos en temas como gobierno, genómica o cultura, su utilidad se ve comprometida por problemas de disponibilidad, rendimiento y cumplimiento de estándares técnicos. Por ejemplo, muchos *endpoints* experimentan tiempos de inactividad prolongados, respuestas lentas o soporte limitado para funciones SPARQL 1.1, lo que dificulta su adopción. Para abordar esto, presenta SPARQLES (Vandenbussche y cols., 2017), un sistema de monitoreo continuo que evalúa más de 400 *endpoints* públicos en cuatro dimensiones clave: descubribilidad (metadatos disponibles), interoperabilidad (cumplimiento de estándares SPARQL), rendimiento (velocidad de respuesta) y disponibilidad (tiempo activo).

La metodología propuesta por SPARQLES combina consultas automatizadas y análisis. El sistema realiza consultas cada hora para verificar la disponibilidad y ejecuta pruebas estandarizadas, como consultas de unión o recuperación de grandes volúmenes de datos, para medir el rendimiento. Los resultados se integran en una plataforma web que muestra estadísticas en tiempo real. Los resultados de las consultas se almacenan en *MongoDB*, mientras que el *frontend*, desarrollado con *NodeJS*, permite visualizar tendencias históricas y comparar proveedores.

Como fue mencionado anteriormente, la aplicación se centra más en dimensiones relacionadas a la calidad de la respuesta de un *endpoint* SPARQL, sin priorizar tanto la calidad del contenido de la respuesta, que será el punto central de este trabajo. Además, al momento de escribir este análisis, a pesar de proveer un enlace a la herramienta, este no funciona, ya que la aplicación se encuentra *offline*.

## 3.4. YummyData: providing high-quality open life science data - 2018

Este trabajo habla sobre el desafío de la fragmentación y la falta de transparencia en los conjuntos de datos biomédicos disponibles a través de *endpoints* SPARQL. Los autores destacan que, aunque estos recursos son esenciales para la investigación en ciencias de la vida, los investigadores enfrentan dificultades para identificar qué proveedores ofrecen datos actualizados y estables. Esto se debe a que múltiples proveedores pueden distribuir los mismos datos con niveles variables de calidad, soporte técnico y cumplimiento de estándares, lo que dificulta la elección de fuentes confiables. Para resolver esto, presentan YummyData (Yamamoto, Yamaguchi, y Splendiani, 2018), una plataforma que evalúa y monitorea continuamente *endpoints* SPARQL relevantes para la biomedicina.

La herramienta realiza consultas periódicas para medir el rendimiento y la conformidad de los *endpoints* con estándares como SPARQL 1.1 (W3C, 2013), *Vocabulary of Interlinked Datasets*[1] (VoID) o *Cross-Origin Resource Sharing* (CORS)[2]. YummyData se caracteriza por generar un "Umaka Score", una puntuación integral que sintetiza seis dimensiones clave: Disponibilidad, Frescura, Operatividad, Utilidad, Validez y Rendimiento. También se clasifican todos los *endpoints* con notas (llamadas "Umaka rank") de la A a la E dependiendo del Umaka Score. Esta puntuación permite a los investigadores comparar *endpoints* de manera objetiva para poder decidir cuál usar, como se ve en la Figura 3.1.


Figura 3.1: Dashboard de rankings de todos los *endpoints* considerados por la aplicación sacada de https://yummydata.org/ el 1/2/2025

No entraremos en detalle sobre el cálculo del Umaka Score[3], pero es importante destacar que cada dimensión se compone de varias métricas, las cuales se combinan para generar un puntaje de calidad específico para dicha dimensión. Finalmente, el Umaka Score se obtiene promediando los valores de todas las dimensiones. Otro punto que vale la pena destacar es que esta aplicación permite

[1] https://www.w3.org/TR/void/
[2] https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
[3] Está muy bien descrito en https://yummydata.org/umaka-score.html

ver cómo varía la calidad de los datos de un grafo a través del tiempo. Esto se puede ver en la gráfica 3.2, donde se toma el KG de puntaje más alto y se grafica el Umaka Score durante un período de tiempo.

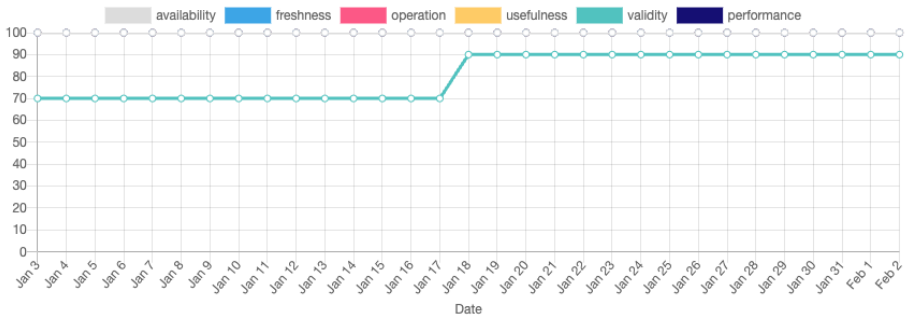



Figura 3.2: Visualización de métricas a través del tiempo del mejor *endpoint* según la Umaka Score sacada de https://yummydata.org/ el 1/2/2025

Los puntos positivos de esta aplicación son que tiene tiempos de respuesta rápidos y que es fácil de utilizar. Por el otro lado, está acotada a unos pocos *endpoints* SPARQL relevantes a la investigación biomédica, y todas sus métricas son genéricas, es decir, no se pueden evaluar cosas a medida.

## 3.5. Structural quality metrics to evaluate knowledge graph quality - 2022

En este trabajo (Seo, Cheon, Kim, y Hyun, 2022) se propone un conjunto de seis métricas estructurales para evaluar la calidad de la estructura de los grafos de conocimiento. A diferencia de las otras evaluaciones basadas en indicadores de escala, este estudio se enfoca en la estructura del grafo, lo que permite captar características que van más allá de la magnitud del grafo.

Entre las métricas presentadas se encuentra el *Instantiated Class Ratio*, que mide la proporción de clases definidas en la ontología que poseen instancias reales, y el *Instantiated Property Ratio*, que cuantifica el grado en que las propiedades definidas se utilizan en triplas RDF.

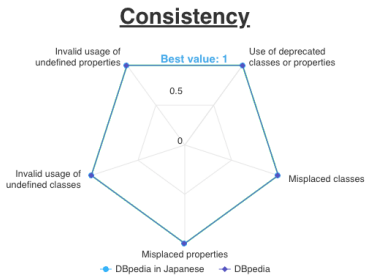
El estudio, que se aplica a diversos KG: Wikidata, DBpedia, YAGO, Google Knowledge Graph, Freebase y Raftel, termina demostrando que una mayor cantidad de clases o propiedades no garantiza una alta calidad si no se utilizan activamente. Este trabajo presenta métricas interesantes, pero no muestra una herramienta o aplicación asociada.

## 3.6. KGHeartBeat: an Open Source Tool for Periodically Evaluating the Quality of Knowledge Graphs - 2024

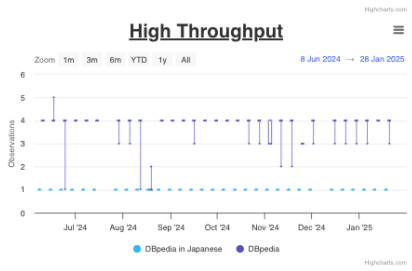
Esta herramienta de código abierto presentada en (Pellegrino, Rula, y Tuozzo, 2024) es la que se acerca más a nuestro objetivo. Desarrollada en el mismo año que este trabajo, plantea dos productos: una aplicación web y una biblioteca.

La aplicación web consta de una interfaz que permite a los usuarios explorar visualmente y comparar diferentes grafos de conocimiento. Además, ofrece una amplia selección de dimensiones y métricas predefinidas, con valores ya calculados para un conjunto de grafos disponibles en *Linked Open Data (LOD) Cloud*[4] y en *DataHub*[5]. Sin embargo, esto implica una limitación: solo es posible evaluar la calidad de los grafos que se encuentran en estas fuentes, sin la opción de analizar otros grafos externos.

Uno de los puntos fuertes de la aplicación es que permite comparar dos o más grafos en función de distintos criterios de calidad, lo que facilita identificar diferencias y tendencias entre ellos. Además, cada métrica cuenta con una visualización específica. En la Figura 3.3 se muestra que para Consistencia la visualización es un pentágono, y para alto rendimiento es una gráfica con ejes X e Y. Esto permite una interpretación clara y adaptada a la naturaleza específica de cada métrica.




(a) Comparación de DBpedia y DBpedia en Japonés en consistencia




(b) Comparación de DBpedia y DBpedia en Japonés en una métrica de rendimiento

Figura 3.3: Dos métricas diferentes con sus respectivas gráficas. Imágenes tomadas de http://www.isislab.it:12280/kgheartbeat/ el 29/1/2025

Otro aspecto destacado es la posibilidad de visualizar la evolución de la calidad de los datos a lo largo del tiempo, como se ve en la Figura 3.4. Dado que los grafos a evaluar están predefinidos y su calidad se calcula semanalmente, los resultados anteriores quedan almacenados, lo que facilita su comparación con

[4] https://lod-cloud.net/
[5] https://datahubproject.io/

los valores actuales y el análisis de tendencias.

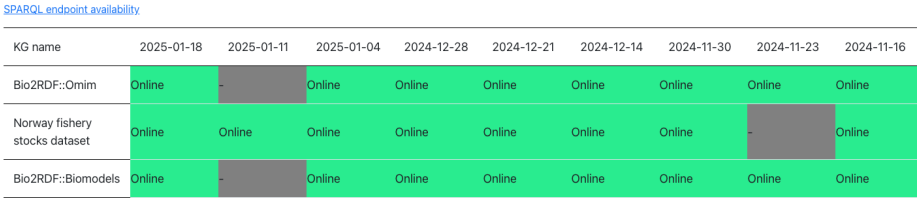



Figura 3.4: Comparación de métrica disponibilidad de endpoint SPARQL sobre tres grafos a través del tiempo. Imágenes tomadas de http://www.isislab.it:12280/kgheartbeat/ el 29/1/2025

Por otro lado, la biblioteca KGHeartBeat API, publicada en el *Python Package Index* (PyPI), permite calcular las métricas en cualquier momento. Para esto, hay que instalar la biblioteca en un ambiente local. Por ejemplo, para calcular la disponibilidad de un grafo, se podría hacer como se muestra[6] en el Listing 3.6.

```
from kgheartbeat import KnowledgeGraph

# Instanziate a KnowledgeGraph class, passing the id of the kg to
    be analyzed
kg = KnowledgeGraph('dbpedia')

# Check the SPARQL endpoint availability
sparqlAv = kg.checkEndpointAv()
# Check if the links for download the dataset is up
checkDump = kg.checkDownload()

print(f"SPARQL endpoint availability: {sparqlAv}\n \
       RDF dump link availability: {checkDump}\n\
```

Esta, a diferencia de la aplicación web, devuelve únicamente los resultados numéricos. Sin embargo, ofrece resultados más actualizados, ya que, mientras la interfaz web puede mostrar datos con hasta una semana de antigüedad, la biblioteca permite obtener mediciones en tiempo real cuando se necesite mayor frescura en los datos. Otro punto a destacar es que comparte el defecto de la aplicación web, solo puede evaluar grafos que estén disponibles en *LOD Cloud* o en *DataHub*.

Por último, consideramos importante señalar que, al intentar utilizar la página para realizar este análisis comparativo, encontramos que su uso no resulta fácil. A pesar de ser la única herramienta con una interfaz relativamente amigable, aún presenta varios aspectos por mejorar, especialmente en términos de usabilidad y tiempos de carga.

[6] Ejemplo sacado de https://pypi.org/project/kgheartbeat/

## 3.7. Conclusiones

Habiendo analizado todos estos trabajos, realizamos un análisis comparativo con el objetivo de determinar qué cualidades tienen las herramientas disponibles, y qué otras nos parecerían interesantes que tuvieran.

En primer lugar, el trabajo *Structural quality metrics to evaluate knowledge graph quality* (Seo y cols., 2022) a pesar de que presenta métricas muy interesantes y cómo medirlas, no presenta una herramienta. Como en esta Sección buscamos comparar herramientas, se descarta del análisis comparativo.

Por otro lado, el trabajo *KGHeartBeat* (Pellegrino y cols., 2024) presenta dos aplicaciones diferentes, una aplicación web y una biblioteca, por lo que creemos que merecen ser analizadas individualmente, a pesar de que ambas están en el mismo repositorio.

Por último, a pesar de que se agregaron al análisis comparativo, SPARQLES y Luzzu no son aplicaciones que hoy se puedan usar, ya que no están disponibles o no son mantenidas; por eso, en la tabla 3.5 se encuentran con un fondo más oscuro.

Se tuvieron en consideración seis aplicaciones y cinco cualidades interesantes que una aplicación ideal tendría que tener. Las seis aplicaciones analizadas son: KGHeartBeat API, KGHeartBeat web app, RDFUnit, YummyData, SPARQLES, Luzzu. Las cinco cualidades a tener en cuenta son:

1. **Personalización**: Refiere a que un usuario pueda editar, borrar y redefinir dimensiones, factores, métricas y métodos. Esto parece clave para tener una aplicación a medida, donde cada usuario puede tener una cierta prioridad de qué le es importante medir, para eso tiene que ser capaz de medir lo que desee, y de la forma en que desee.

2. **Flexibilidad**: Que se pueda evaluar la calidad de cualquier KG con un *endpoint* SPARQL público. Se encuentra valor en que no se necesite que el *endpoint* a evaluar esté en algún catálogo de *endpoints* (como *DataHub* o *LOD Cloud*).

3. **Accesibilidad**: Consideraremos como accesibles a aquellas herramientas que tienen una interfaz web. Además, se busca que la herramienta de evaluación de la calidad pueda ser utilizada por cualquier usuario interesado en evaluar la calidad de datos de un grafo de conocimiento, no solamente los usuarios técnicos capaces de utilizar una interfaz de línea de comandos.

4. **Frescura**: Diremos que tiene frescura si permite la re evaluación de la calidad a demanda. Se considera importante el hecho de que el usuario pueda tener resultados recientes. Especialmente si el usuario es dueño del grafo de conocimiento que se quiere evaluar, y está tratando de corregir los errores de calidad encontrados por la herramienta en tiempo real.

5. **Monitoreo**: Esta capacidad refiere a la posibilidad de comparar calidad a través del tiempo dentro de la herramienta. Es valioso poder ver tendencias

de como evoluciona la calidad de los datos al pasar el tiempo, más si el usuario es responsable por la calidad del grafo a evaluar.

La comparación de las cualidades y aplicaciones mencionadas se presenta en la tabla 3.5. Aquí se nota un vacío en las soluciones relevadas en la literatura, ya que no se tienen herramientas que tengan las cinco cualidades al mismo tiempo, ni siquiera cuatro.

De las herramientas en funcionamiento, RDFUnit (Kontokostas y cols., 2014) permite personalizar las dimensiones, factores, métricas y métodos; sin embargo, está hecha para usuarios que están familiarizados con la línea de comandos, y no tiene una interfaz web. Por otro lado, YummyData (Yamamoto y cols., 2018), que tiene una interfaz web muy cuidada y permite ver la calidad a través del tiempo, está reducida a solo medir la calidad en un subconjunto muy pequeño de KG relacionados con la biomedicina. Por último, la API de KGHeartBeat (Pellegrino y cols., 2024) es similar a RDFUnit pero no permite personalizar las consultas SPARQL, y aunque la web app de KGHeartBeat sea una buena herramienta, no permite medir KG con métricas a medida, ni brinda la posibilidad de reevaluar el KG a demanda.

| | Personalización | Flexibilidad | Accesibilidad | Frescura | Monitoreo |
|---|---|---|---|---|---|
| SPARQLES | ✘ | ✘ | ✓ | ✘ | ✘ |
| Luzzu | ✓ | ✓ | ✘ | ✓ | ✘ |
| RDFUnit | ✓ | ✓ | ✘ | ✓ | ✘ |
| YummyData | ✘ | ✘ | ✓ | ✘ | ✓ |
| KGHeartBeat API | ✓ | ✘ | ✘ | ✓ | ✘ |
| KGHB Web App | ✘ | ✘ | ✓ | ✘ | ✓ |

Figura 3.5: Tabla comparativa de las herramientas analizadas

Habiendo hecho este análisis, y priorizando estas cualidades, se plantea el desafío de poder desarrollar una herramienta que cumpla con la máxima cantidad de las cualidades mencionadas, la cual se presenta en el capítulo siguiente.

# Capítulo 4

# Diseño de un modelo de calidad de datos para grafos RDF

Antes de poder desarrollar una herramienta que llene el espacio faltante en la literatura, es necesario definir de forma clara qué es lo que esta herramienta busca medir. En este capítulo se describe el trabajo y los aportes realizados por el grupo en el contexto del presente proyecto de grado en términos del establecimiento de un marco teórico en el que sustentar el producto de software a implementar.

Primero, en la Sección 4.1, se describen las dimensiones, factores y métricas elegidos como punto de partida para trabajar en la implementación de la herramienta. La Sección 4.2 contextualiza el concepto de granularidad para un grafo RDF y define las granularidades posibles en una base de datos de este estilo. Por último, en la Sección 4.3 se detallan las métricas seleccionadas a partir de la literatura y las razones de su elección o descarte.

## 4.1. Dimensiones y factores considerados por la herramienta

En la Sección 2.2.1 se detallaron las dimensiones y factores relevados de la bibliografía. En esta Sección se provee un resumen de las elecciones realizadas por el equipo. La tabla 4.1 muestra un resumen de las dimensiones elegidas y sus definiciones.

| Dimensión | Definición |
|---|---|
| Exactitud | La Exactitud refiere al grado en que las entidades y relaciones representan correctamente los fenómenos de la vida real |

| | |
|---|---|
| Completitud | La Completitud refiere al grado en el cual toda la información requerida está presente en el grafo |
| Frescura | La Frescura refleja el grado al cual el grafo de conocimiento se mantiene actualizado con el estado del mundo real |
| Consistencia | La Consistencia mide la correctitud lógica del grafo de conocimiento |
| Concisión | La Concisión evalúa la no inclusión de elementos irrelevantes al dominio o redundantes en el grafo |
| Facilidad de comprensión | La Facilidad de comprensión busca medir el grado en que los datos contenidos en el grafo pueden ser comprendidos sin ambigüedad y utilizados por humanos |

Tabla 4.1: Elección de dimensiones

En cuanto a los factores, ocurre que en la bibliografía consultada no siempre se contempla explícitamente el concepto de factor de calidad como un mecanismo para refinar las dimensiones de calidad de datos. Esta ausencia llevó a la necesidad de asignar las métricas relevadas a factores existentes en la literatura, un aporte de este trabajo que permitió establecer correspondencias para estructurar y categorizar adecuadamente cada métrica dentro del marco conceptual adoptado.

Considerando lo anterior y dadas las dimensiones mencionadas, en la tabla 4.2 se muestra un resumen de los factores elegidos para cada dimensión (recordar que algunos de ellos fueron descartados, como se detalla en la Sección 4.1).

| **Dimensión** | **Factor** | **Definición** |
|---|---|---|
| Exactitud | Exactitud Sintáctica | La exactitud sintáctica busca establecer si los conceptos modelados en el grafo corresponden a conceptos válidos del dominio, independientemente de si reflejan o no la realidad |
| Exactitud | Exactitud Semántica | La exactitud semántica mide qué tan bien representada está la realidad por los datos almacenados |
| Completitud | Completitud de Esquema | La completitud de esquema refiere al grado en el cual las clases y propiedades de un esquema están representadas en el grafo |
| Completitud | Completitud de Propiedad | La completitud de propiedad refiere a la proporción de valores faltantes para una propiedad específica |

| | | |
|---|---|---|
| Completitud | Completitud de Población | La completitud de población refiere al porcentaje de todas las entidades del mundo real para un tipo particular que están representadas en el grafo |
| Frescura | Actualidad | La actualidad busca medir qué tan recientes son los datos del sistema |
| Frescura | Oportunidad | La oportunidad busca establecer si los datos del sistema están disponibles al momento en que se los necesita |
| Frescura | Volatilidad | La volatilidad caracteriza la frecuencia con que los datos cambian a lo largo del tiempo |
| Consistencia | Consistencia | La consistencia mide la correctitud lógica del grafo de conocimiento |
| Concisión | Concisión Intensional | La concisión intensional hace referencia a la minimización de redundancia a nivel de esquema (propiedades, clases, etc.) |
| Concisión | Concisión Extensional | La concisión extensional refiere a la minimización de redundancia a nivel de los elementos de datos, es decir, las instancias |
| Concisión | Concisión Representacional Intensional | La concisión representacional intensional hace referencia al grado en que el contenido del grafo, a nivel de esquema, se representa de manera compacta |
| Concisión | Concisión Representacional Extensional | La concisión representacional extensional hace referencia al grado en que el contenido del grafo, a nivel de datos, se representa de manera compacta |
| Facilidad de Comprensión | Facilidad de Comprensión | La facilidad de comprensión mide el grado en que los datos contenidos en el grafo pueden ser comprendidos sin ambigüedad y utilizados por humanos |

Tabla 4.2: Elección de factores

## 4.2. Granularidades en grafos de conocimiento RDF

Como se mencionó en la Sección 2.2.2, no se encontraron trabajos previos que definan las granularidades pertinentes al evaluar la calidad en una base de datos RDF. En esta Sección se busca atacar ese vacío en la literatura y proveer dichas definiciones.

### 4.2.1. Análisis

En el contexto de calidad de datos, la granularidad debe ser definida para las métricas que se desean considerar como parte del modelo de datos. La granularidad existe entonces ligada a la métrica, y busca representar el dato o conjunto de datos sobre los cuáles la métrica permite efectuar una afirmación o valoración.

Si nos abstraemos al concepto puramente teórico de grafo, tenemos que un grafo es un conjunto de nodos unidos por aristas. A priori, esto nos permite pensar en tres granularidades: el **grafo** en su totalidad, granularidad de **nodo**, es decir, la métrica se asocia a un nodo (o un conjunto de nodos) en particular, y granularidad de **arista**, donde la métrica se asocia a una arista (o conjunto de aristas). Además, un subconjunto de nodos y sus correspondientes aristas conforman un subgrafo del grafo, por lo que también puede considerarse **subgrafo** como una granularidad (más allá de los desafíos técnicos que esto implique, como poder describir con exactitud el subgrafo al que se está refiriendo).

Ahora bien, como ya se mencionó en la Sección 2.1.1, un grafo RDF posee un formato estandarizado y soporta vocabularios (como RDFS y OWL) que dan semántica a los datos contenidos en él. El formato de triplas y la existencia de un esquema generan clasificaciones de los recursos del grafo. Esto hace pertinente considerar nuevas granularidades que se desprendan de dichas clasificaciones.

**Granularidades RDF:** Los conceptos definidos en RDF que pueden entenderse como granularidades son: **tripla**, **recurso** (a veces referido como entidad en la literatura), **subgrafo**, **grafo** y ***dataset***.

Las **triplas** son el nivel más fundamental en RDF. Si bien puede considerarse como una posible granularidad, en el contexto de este trabajo no tiene sentido hacerlo. Esto se debe a que en ningún momento estamos interesados en evaluar la calidad de una tripla individualmente, aislada del resto de los datos y, por ende, descontextualizada del grafo. Un *recurso* en RDF es una entidad del universo que puede representarse como un dato. Esto abarca un espectro muy grande de entidades; los conceptos, propiedades, valores literales, *datatypes*, instancias; todos ellos pueden representarse como recursos en RDF. El problema con definir una granularidad de este tipo es, entonces, la poca especificidad que aporta a la descripción de la métrica. Como se verá más adelante, el uso de un esquema

permite mejorar esta situación, por lo que se descarta recurso como granularidad en favor de las granularidades que se introducen a continuación.

Un **subgrafo** RDF es un subconjunto de triplas del grafo (W3C, 2014a). Es perfectamente válido considerar la granularidad de subgrafo, dado que podría ser de interés conocer la calidad de un subconjunto de los datos contenidos en un grafo RDF. Sin embargo, en este trabajo se decidió descartar esta granularidad por razones puramente técnicas y de alcance del proyecto. Para poder evaluar la calidad de un subgrafo, es necesario proveer un mecanismo que permita describir dicho subgrafo, es decir, dar cuenta de la información que contiene. Lo que se desea entonces es una funcionalidad de vistas sobre los datos, de manera similar a la que se ofrece en el lenguaje SQL. Sin embargo, el estándar SPARQL no define una manera de lograr lo anterior. Si bien el comando `CONSTRUCT` permitiría definir un subgrafo, no existen mecanismos para efectuar consultas posteriores a dicha definición (Lorena Etcheverry, 2012). **Por esta razón, decidimos rechazar esta granularidad para la implementación final**, pero es pertinente reconocerla dado que en ciertos contextos puede ser de utilidad. Por ejemplo, si tuviéramos un grafo de conocimiento de personas y quisiéramos evaluar la calidad de todas las triplas asociadas a personas residentes de un país en particular (notar que las triplas mencionadas estarían efectivamente definiendo un subgrafo del grafo más general).

Como ya se mencionó en la Sección 2.1.1, un *dataset* RDF se compone de un grafo *default* y uno o más grafos nombrados. Los grafos nombrados tienen (valga la redundancia) un nombre que los identifica, por lo que al considerar la **granularidad de grafo** se sobreentiende que la manera de identificar el conjunto de datos al que la granularidad refiere es mediante dicho nombre. Esto no es cierto para el grafo *default*, que carece de un identificador de esta índole. Además, el estándar RDF tampoco restringe la relación que un grafo *default* debe mantener con los grafos nombrados; de hecho, ni siquiera impone la existencia de una relación. La semántica de los *datasets* RDF es de libre elección, y existen varias alternativas a considerar y aplicar (W3C, 2014a), que impactan en los resultados obtenidos mediante SPARQL al ejecutar consultas contra el grafo *default* (W3C, 2013). Debido a que no hay un consenso sobre lo que el grafo *default* representa en una consulta sobre todo un *dataset* RDF, **se decide no considerar la granularidad dataset como parte de este trabajo**.

**Granularidades RDFS:** El esquema definido por RDFS permite extender la noción de granularidades a clases, propiedades y *datatypes*.

Como se mencionó en la Sección 2.1.1, RDFS permite describir clases de recursos y sus relaciones. Es entonces la noción de clase la que permite categorizar los recursos y agruparlos de cierta manera. Esta extensión semántica de RDF define múltiples clases: `rdfs:Resource`, `rdfs:Class`, `rdfs:Literal`, `rdfs:Datatype`, `rdf:langString`, `rdf:HTML`, `rdf:XMLLiteral` y `rdf:Property` (W3C, 2014c). Si bien todas estas clases, que son en fin clasificaciones de re-

cursos en el grafo RDF, tienen méritos para ser consideradas granularidades, consideramos que algunas de ellas destacan por el significado semántico que poseen: las clases y propiedades. La **granularidad clase** permite asociar la evaluación de calidad de una métrica a una clase (`rdfs:Class`) o conjunto de clases especificas del dominio evaluado. Esto quiere decir que, al hacer una valoración sobre una métrica con esta granularidad, estamos afirmando algo sobre el conjunto de instancias pertenecientes a esa clase (o conjunto de clases). Algo análogo sucede al considerar la **granularidad propiedad**, pero lógicamente, asociado a instancias de la o las propiedades relacionadas.

**Granularidades OWL:** OWL se construye sobre RDFS, extendiendo la semántica a partir de la definición de nuevas clases, subclases de `rdfs:Class`, que describan con más exactitud los recursos. No se proveerá aquí de una lista exhaustiva con las definiciones del vocabulario, si el lector está interesado puede encontrarla en (W3C, 2012b). Si bien todas las clases definidas en OWL generan una clasificación de recursos, algunas de ellas son demasiado específicas para ser consideradas como granularidades. Un ejemplo es la clase `owl:DeprecatedProperty`, que permite clasificar las propiedades deprecadas en una ontología como se muestra en el Listing 4.1.

```
owl:DeprecatedProperty a rdfs:Class ;
     rdfs:label "DeprecatedProperty" ;
     rdfs:comment "The class of deprecated properties." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf rdf:Property .
```

Listing 4.1: Definición de owl:DeprecatedProperty

A pesar de que es cierto que esta definición genera una clasificación de recursos, creemos que no es lo suficientemente general para ser considerada una granularidad. Resulta cuanto menos extraño pensar en una métrica de calidad específica a propiedades deprecadas de la ontología. Incluso si se quisiera medir algo así, podría hacerse considerando una granularidad de propiedad y aplicando la métrica a todas las instancias de tipo *DeprecatedProperty*. Considerar una granularidad de propiedades deprecadas no sólo no aporta demasiado en términos de describir lo que se está midiendo, sino que además dificulta el entendimiento de las granularidades en sí, ya que desdibuja la frontera de esta granularidad con la granularidad de propiedad.

No obstante lo anterior, en algunos casos las clases agregadas por el vocabulario OWL permiten mejorar la especificidad de lo que se mide, manteniendo la generalidad necesaria para poder considerar esas clases como granularidades. A nuestro entender, éstas son `owl:Class`, `owl:ObjectProperty` y `owl:DatatypeProperty`. La primera hace referencia a clases OWL, y es una subclase de `rdfs:Class`, pero que permite mayor expresividad (por ejemplo, permite la definición de axiomas de disjunción y equivalencia entre clases). Las últimas dos particionan el espacio de las propiedades (`rdf:Property`) en dos; la primera de ellas agrupa propiedades que relacionan individuos (instancias de clases) con otros individuos, la segunda relaciona individuos con valores de datos

(por ejemplo, instancias de un `datatype` RDF).

### 4.2.2. Granularidades consideradas en la herramienta

A partir del análisis efectuado en la Sección anterior, se consideran las granularidades de grafo, subgrafo, clase y propiedad.

**Grafo** La granularidad más general posible en el contexto de este trabajo. Dado el problema descrito respecto de los *datasets* RDF en la Sección 4.2.1, se establece que esta granularidad necesariamente hace referencia a un grafo nombrado, por lo que las métricas asociadas a ella exigen que se las provea de tal grafo.

**Subgrafo** Como se mencionó en la Sección anterior, esta granularidad no se considera en la implementación de este trabajo debido a la dificultad de su implementación y al alcance definido.

**Clase** Esta granularidad permite al usuario evaluar métricas asociadas a una clase, o conjunto de clases en particular del dominio a evaluar. Notar que la noción de clase como granularidad que estamos introduciendo es independiente del vocabulario que se utiliza. Es decir, no estamos distinguiendo una granularidad de clases RDFS y otra de clases OWL, sino que consideramos ambas contenidas en la noción más general de "granularidad de clase". La razón de esta decisión consiste en que consideramos que la granularidad tiene que ser un concepto general y no atarse a cuestiones de implementación, como el uso de vocabulario OWL. A modo de ejemplo, el usuario podría evaluar el cumplimiento de un axioma `owl:disjointWith` entre una clase $A$ y otra clase $B$, y la métrica asociada sería considerada de granularidad clase, independientemente del uso de OWL. Es una granularidad que se incluye en la solución, dado que consideramos que aporta valor y permite al usuario definir restricciones que hagan sentido al dominio particular al cual su grafo hace referencia.

**Propiedad** Esta granularidad permite al usuario evaluar métricas asociadas a una propiedad o conjunto de propiedades. Se incluye en la solución por las mismas razones que el punto anterior. Si bien se discutió en la Sección 4.2.1 la distinción de `owl:ObjectProperty` y `owl:DatatypeProperty`, decidimos evitar el uso de estas granularidades en la herramienta desarrollada. Esto se debe, de manera similar a lo que sucede en el caso de `owl:Class`, a que obligan al uso de lenguaje OWL, y por ende, permean cuestiones de implementación en una definición que debería ser puramente teórica y aplicable a todos los grafos de conocimiento que utilicen lenguaje RDF.

A modo de resumen, la Tabla 4.3 presenta las granularidades consideradas en la implementación, junto con sus definiciones:

| Granularidad | Definición |
|---|---|
| Grafo | La métrica asociada permite efectuar valoraciones sobre un grafo nombrado |
| Clase | La métrica asociada permite efectuar valoraciones sobre instancias de una clase o de un conjunto de clases |
| Propiedad | La métrica asociada permite efectuar valoraciones sobre instancias de una propiedad o de un conjunto de propiedades |

Tabla 4.3: Elección de granularidades

## 4.3. Elección de métricas

En esta Sección se describen las métricas consideradas y sus métodos, la decisión final sobre su implementación como parte de nuestra herramienta y, en caso de que lo amerite, las razones por las cuales se descartan. Además, se propone una clasificación de las métricas contempladas en dimensiones y factores, así como la asignación a una granularidad particular.

Se utilizaron métricas cuyos métodos son implementados como consultas SPARQL o división de resultados entre dos consultas SPARQL. Esta decisión se fundamenta en que SPARQL, estandarizado por el W3C para consultar datos en RDF, permite evaluar la calidad directamente sobre la estructura del grafo, sin intermediarios. Como lenguaje ampliamente adoptado, garantiza métodos reproducibles y adaptables a distintos entornos.

Antes de comenzar, vale la pena explicitar que la clasificación de todas las métricas presentadas, y corrección de algunas, es parte esencial del aporte de este trabajo.

### 4.3.1. Métricas consideradas del trabajo Test-driven Evaluation of Linked Data Quality

A continuación se presentan las métricas definidas en (Kontokostas y cols., 2014) que fueron consideradas para el trabajo. Primero se describen los patrones definidos en dicho trabajo, que son consultas SPARQL pensadas para ejecutarse como una suite de tests, como se explicó en la sección 3.1. Luego, se da un ejemplo de instanciación para cada patrón y, por último, se define la implementación de la métrica y cómo se construye a partir del método original.

Un detalle a destacar de todos los patrones definidos en (Kontokostas y cols., 2014) es que, al estar pensado para ejecutar como un conjunto de tests sobre un grafo, los patrones devuelven (en caso de encontrar instancias), violaciones de lo que se desea chequear. A modo de ejemplo, si se quiere validar una restricción de orden con el patrón COMP (detallado más adelante) donde $v_1 > v_2$, al instanciar el *template* debe usarse el inverso del operador, es decir <=. Entonces,

semánticamente hablando, el patrón chequea efectivamente una restricción de tipo mayor entre las variables. Pero, al estar diseñado como un test, es necesario que la consulta devuelva instancias que no cumplen con lo pedido, para poder reportarlas junto a la falla de dicho test. Es por esto que para cada patrón luego se definirá la métrica correspondiente, la cual tendrá un tipo de resultado y posiblemente requiera alterar ligeramente la estructura del patrón.

Otra aclaración pertinente es que se encontraron errores en los patrones descritos en el documento en cuestión. En algunos casos se encuentran errores sintácticos en la escritura de la consulta en sí. En otros, el error es semántico, donde, por ejemplo, se hace un uso equívoco de una propiedad. Dado que los errores semánticos son, en su mayoría, menores, en esta Sección se escribe el patrón corregido. En los casos en los que el error es semántico, se anota en el patrón correspondiente.

### Métrica basada en el patrón COMP

Este patrón se basa en comparar los valores asociados a un par de propiedades dadas. A partir de un operador que define una restricción de orden y de un par de propiedades, el patrón busca recursos donde los literales asociados mediante las propiedades indicadas violen la restricción. El Listing 4.2 presenta la consulta SPARQL que implementa la búsqueda mencionada, donde P1 y P2 son variables a ser instanciadas por propiedades cuyos rangos son valores del mismo datatype, y OP por un operador de comparación booleano.

```
SELECT ?s WHERE {
    ?s %%P1%% ?v1 .
    ?s %%P2%% ?v2 .
    FILTER ( ?v1 %%OP%% ?v2 )
}
```

Listing 4.2: Patrón COMP

A modo de ejemplo, si instanciamos a las variables P1 y P2 como las propiedades `dbo:deathDate` y `dbo:birthDate` respectivamente y OP con el operador $<$, como se muestra en el Listing 4.3, podemos detectar recursos donde el valor asociado a `dbo:deathDate` es menor que el valor de `dbo:birthDate`. Este es un buen ejemplo para hacer referencia a la afirmación de que estos patrones encuentran violaciones de lo que se quiere chequear. Notar que el patrón de grafo buscado es en realidad inverso a lo que semánticamente se desea chequear. La consulta busca instancias donde la fecha de fallecimiento se da antes que su fecha de nacimiento. Esto es lógicamente inverso a lo que se espera que semánticamente cumpla el grafo: que todas las personas hayan nacido antes de fallecer.

```
SELECT ?s WHERE {
    ?s dbo:deathDate ?v1 .
    ?s dbo:birthDate ?v2 .
    FILTER ( ?v1 < ?v2 )
```

```
}
```

Listing 4.3: Ejemplo COMP

**Definición de la métrica:** Como cada instanciación de este patrón requiere el establecimiento de un par de propiedades, definiremos la métrica con granularidad *Propiedad* y asociada al conjunto de propiedades {P1, P2}. Además, consideramos la métrica de tipo *booleano*, donde los valores que una evaluación de ella puede tomar son 0 o 1. Esto implica que tengamos que reescribir el método de medición para poder obtener un valor de este estilo, mediante el comando `ASK` de SPARQL. El Listing 4.4 muestra la definición de este nuevo método.

```
ASK {
    ?s %%P1%% ?v1 .
    ?s %%P2%% ?v2 .
    FILTER ( ?v1 %%OP%% ?v2 )
}
```

Listing 4.4: Patrón COMP asociado a la métrica

A partir de esta consulta, obtendremos el valor `true` si existen recursos que contradigan la restricción de orden que queremos verificar, y `false` en caso contrario. Como deseamos obtener un valor numérico para las métricas, el resultado se computa como el complemento del valor numérico correspondiente al *booleano* retornado. Es decir que una evaluación de esta métrica devuelve $1 - \texttt{resultado}$, donde `resultado` es 1 si la consulta devuelve `true`, y 0 en caso contrario.

Por último, la Tabla 4.4 presenta la dimensión y factor a la que corresponde esta métrica, la granularidad a la que corresponde, el tipo de resultado y las variables que son necesarias instanciar para aplicarla.

| | |
|---|---|
| Dimensión | Exactitud |
| Factor | Exactitud Semántica |
| Resultado | Booleano (0 ó 1) |
| Granularidad | Propiedad (conjunto {P1, P2}) |
| Variables | $P_1$: `owl:DatatypeProperty`.<br>$P_2$: `owl:DatatypeProperty`.<br>$OP$: Operador de comparación. |

Tabla 4.4: Métrica en base al patrón COMP

### Métrica basada en el patrón MATCH

Este patrón se basa en la comparación de los valores asociados a una propiedad dada con una expresión regular. Esa comparación puede darse de manera natural o negada. Lo que el patrón pretende encontrar son valores que no cumplan con la expresión regular (en caso de estar usando una comparación mediante negación) o valores que cumplen con dicha expresión, pero donde la

expresión en sí denota un incumplimiento del formato esperado del valor. El Listing 4.5 presenta la consulta SPARQL asociada al patrón. P1 es una variable a ser instanciada por la propiedad que se quiere evaluar, NOP es una variable opcional que puede instanciarse como el operador de negación !, y REGEXP se instancia como la expresión regular a chequear.

```
SELECT DISTINCT ?s
WHERE {
    ?s %%P1%% ?value .
    FILTER ( %%NOP%% regex(str(?value), %%REGEXP%%) )
}
```

Listing 4.5: Patrón MATCH

Por ejemplo, el Listing 4.6 muestra una instanciación del patrón que busca códigos postales que no cumplan con un formato de 5 dígitos, todos numéricos. En este caso, P1 se instancia como `dbo:postalCode`, el NOP como el operador de negación ! y REGEXP como la expresión regular `^[0-9]{5}$`. No parece muy natural pensar en casos donde no se utilice la variable NOP, y en los que, en cambio, la expresión regular capture violaciones del formato esperado, pero por completitud brindamos un ejemplo en el Listing 4.7. En este caso no se instancia la variable NOP, y se utiliza como expresión regular `[a-zA-Z]+`, que captura cualquier carácter alfabético en el valor (y por consiguiente, cualquier resultado encontrado iría en contra del formato esperado).

```
SELECT DISTINCT ?s
WHERE {
    ?s dbo:postalCode ?value .
    FILTER ( !regex(str(?value), ^[0-9]{5}$) )
}
```

Listing 4.6: Patrón MATCH - Ejemplo 1

```
SELECT DISTINCT ?s
WHERE {
    ?s dbo:postalCode ?value .
    FILTER ( regex(str(?value), ^[a-zA-Z]+) )
}
```

Listing 4.7: Patrón MATCH - Ejemplo 2

**Definición de la métrica:** Como cada instanciación de este patrón se evalúa a partir de la definición de la propiedad referenciada por P1, definiremos la métrica con granularidad *Propiedad* y asociada a P1. Además, consideramos la métrica de tipo *booleano*, donde los valores que una evaluación de ella puede tomar son 0 o 1. Esto implica nuevamente que tengamos que reescribir el método de medición para poder obtener un valor de este estilo. El Listing 4.8 muestra la definición de este nuevo método:

```
ASK {
    ?s %%P1%% ?value .
    FILTER ( %%NOP%%regex(str(?value), %%REGEXP%%) )
```

```
}
```

Listing 4.8: Patrón MATCH asociado a la métrica

Como buscamos obtener un valor numérico para las métricas, nuevamente el resultado se computa como el complemento del valor numérico correspondiente al *booleano* retornado. Esto es, $1 - \texttt{resultado}$, donde `resultado` es 1 si la consulta devuelve `true`, y 0 en caso contrario.

La Tabla 4.5 presenta la dimensión y factor a la que corresponde esta métrica, el tipo de resultado, la granularidad a la que corresponde, y las variables que son necesarias instanciar para aplicarla.

| | |
|---|---|
| Dimensión | Consistencia |
| Factor | Consistencia |
| Resultado | Booleano (0 ó 1) |
| Granularidad | Propiedad |
| Variables | $P_1$: `owl:DatatypeProperty`.<br>$NOP$: Operador de negación ! (opcional).<br>$REGEXP$: expresión regular a chequear. |

Tabla 4.5: Patrón MATCH

### Métrica basada en el patrón LITRAN

El patrón LITRAN se basa en el chequeo de una restricción de rango sobre una propiedad dada asociada a una clase, también dada. Al igual que en el caso MATCH, la comparación de valores puede darse de forma natural o negada. Lo que este patrón busca son valores que queden por fuera del rango definido (en caso de estar usando la negación) o dentro de él, siendo el rango el complemento de lo que se espera que la propiedad cumpla (es decir, definiendo el rango de valores no deseado). El Listing 4.9 presenta la consulta SPARQL que implementa este patrón. T1 es la clase a la que la propiedad P1 a examinar se asocia, y Vmin y Vmax definen las cotas inferior y superior del rango que se espera que los valores de la propiedad cumplan (o no cumplan si no se usa la negación). Estos valores deben cumplir que su rango sean valores del mismo *datatype*. NOP nuevamente es opcional, y en caso de estar instanciado representa el operador de negación !.

```
SELECT DISTINCT ?s WHERE {
?s rdf:type %%T1%% .
?s %%P1%% ?value .
FILTER (%%NOP%% (?value < %%Vmin%% || ?value > %%Vmax%%)) }
```

Listing 4.9: Patrón LITRAN

La instanciación de variables mostrada en el Listing 4.10), encuentra personas (instancias de `dbo:Person`) cuya altura (`dbo:height`) esté por debajo de 0,4 o sea mayor a 2,5. La semántica que busca comprobarse a nivel del grafo

es que no existan personas con alturas por debajo o por encima de los valores mencionados.

```
SELECT DISTINCT ?s WHERE {
?s rdf:type dbo:Person .
?s dbo:height ?value .
FILTER (?value < 0.4 || ?value > 2.5) }
```

Listing 4.10: Patrón LITRAN

**Definición de la métrica:** En este caso, si bien la restricción de rango se verifica contra la propiedad denotada por la variable P1, dicha propiedad está ligada a la existencia de un sujeto de tipo T1, donde T1 se instancia como una clase. Por esta razón, definiremos la métrica con granularidad de *Clase* y asociada a T1. En el caso de este patrón, consideramos la métrica de tipo *ratio*, siendo su valor un número entre 0 y 1. Dado que el ratio es una comparación entre valores, necesitamos definir qué queremos comparar en relación a qué otra cosa, que además deberán ser valores numéricos. La consulta mostrada en el Listing 4.9 devuelve recursos, por lo que requiere ser modificada para devolver valores numéricos, como se muestra en el Listing 4.11. El comparador que usaremos para calcular el ratio son los resultados de la consulta definida en 4.12. A esta última la denominamos la consulta de universo (en el código: `universe_query`) y lo que hace es devolver la cantidad de instancias pertenecientes a la clase T1.

```
SELECT COUNT (DISTINCT ?s) AS ?count
WHERE {
    ?s rdf:type %%T1%% .
    ?s %%P1%% ?value .
    FILTER (%%NOP%%(?value < %%Vmin%% || ?value > %%Vmax%%))
}
```

Listing 4.11: Patrón LITRAN asociado a la métrica

```
SELECT COUNT (DISTINCT ?s) AS ?universe
WHERE { ?s rdf:type %%T1%% . }
```

Listing 4.12: Consulta para comparar resultado de la métrica asociada al patrón LITRAN

El valor de las evaluaciones para esta métrica se computa entonces como los resultados obtenidos de la consulta del Listing 4.11 (a la que llamaremos `resultado_patron`) sobre los resultados obtenidos de 4.12 (`resultado_universo`). Notar que como 4.11 cuenta la cantidad de recursos que incumplen la restricción de rango que se quiere comprobar, lo que nos interesa obtener es el complemento del ratio calculado. Nos queda entonces que el resultado para una métrica asociada al patrón LITRAN es:

$$\texttt{resultado} = 1 - \frac{\texttt{resultado\_patron}}{\texttt{resultado\_universo}}$$

La Tabla 4.6 presenta la dimensión y factor a la que corresponde esta métrica, el tipo de resultado, la granularidad a la que corresponde, y las variables que son necesarias instanciar para aplicarla.

| | |
|---|---|
| Dimensión | Consistencia |
| Factor | Consistencia |
| Resultado | Ratio [0,1] |
| Granularidad | Clase |
| Variables | $T1$: `rdfs:Class` ó `owl:Class`.<br>$P1$: `owl:DatatypeProperty`.<br>$NOP$: Operador de negación ! (opcional).<br>$V_{min}$: Cota inferior del rango a chequear.<br>$V_{max}$: Cota superior del rango a chequear. |

Tabla 4.6: Patrón LITRAN

### Métrica basada en el patrón TYPEDEP

Este patrón se construye con el propósito de detectar incumplimientos de implicancia entre clases. Asumiendo que se sabe que debería cumplirse que todas las instancias de una cierta clase `T1` son también instancias de la clase `T2`, la consulta de TYPEDEP busca instancias de tipo (`rdf:type`) `T1` que no sean instancias de tipo `T2`. El Listing 4.13 muestra dicha consulta, donde las variables que representan a las clases mencionadas son T1 y T2.

Identifica las instancias que no cumplen las restricciones de implicancia entre clases. Específicamente, devuelve las instancias que pertenecen a una clase T1 pero no están declaradas como pertenecientes a la clase T2.

```
1 SELECT DISTINCT ?s WHERE {
2 ?s rdf:type %%T1%% .
3 FILTER NOT EXISTS { ?s rdf:type %%T2%% } }
```

Listing 4.13: Patrón TYPEDEP

**Definición de la métrica:** En este caso, la restricción de implicancia se verifica entre las clases denotadas por las variables T1 y T2. Dado que estamos evaluando el cumplimiento de una restricción sobre una clase en relación con otra, definiremos la métrica con granularidad de *Clase* y asociada a T1. Al igual que en el caso del patrón LITRAN, consideramos la métrica de tipo *ratio*, siendo su valor un número entre 0 y 1.
Para calcular el ratio, se requiere comparar la cantidad de instancias que incumplen con el total de instancias pertenecientes a la clase T1. La consulta del Listing 4.13 devuelve recursos que incumplen la restricción, por lo que debe ser modificada para devolver un valor numérico, como se muestra en el Listing 4.14.

El comparador que usaremos para calcular el ratio son los resultados de la consulta de universo definida en el Listing 4.15, que devuelve la cantidad de instancias pertenecientes a la clase T1.

```
SELECT COUNT (DISTINCT ?s) AS ?count WHERE {
    ?s rdf:type %%T1%% .
    FILTER NOT EXISTS {
        ?s rdf:type %%T2%%
    }
}
```

Listing 4.14: Patrón TYPEDEP asociado a la métrica

```
SELECT COUNT (DISTINCT ?s) AS ?universe
WHERE { ?s rdf:type %%T1%% . }
```

Listing 4.15: Consulta para comparar resultado de la métrica asociada al patrón TYPEDEP

El valor de las evaluaciones para esta métrica se computa entonces como los resultados obtenidos de la consulta del Listing 4.14 (a la que llamaremos `resultado_patron`) sobre los resultados obtenidos de 4.15 (`resultado_universo`). Notar que como 4.14 cuenta la cantidad de recursos que incumplen la restricción de rango que se quiere comprobar, lo que nos interesa obtener es el complemento del ratio calculado. Nos queda entonces que el resultado para una métrica asociada al patrón TYPEDEP es:

$$\texttt{resultado} = 1 - \frac{\texttt{resultado\_patron}}{\texttt{resultado\_universo}}$$

La Tabla 4.7 presenta la dimensión y factor a la que corresponde esta métrica, el tipo de resultado, la granularidad a la que corresponde, y las variables que son necesarias instanciar para aplicarla.

| | |
|---|---|
| Dimensión | Consistencia |
| Factor | Consistencia |
| Resultado | Ratio [0,1] |
| Granularidad | Clase (conjunto $\{T1, T2\}$) |
| Variables | $T1$: `rdfs:Class` ó `owl:Class`.<br>$T2$: `rdfs:Class` ó `owl:Class`. |

Tabla 4.7: Patrón TYPEDEP

### Métrica basada en el patrón TYPRODEP

Este patrón se basa en verificar que una instancia de una clase determinada posea al menos una propiedad esperada. Asumiendo que es de esperarse que, si una instancia $s$ es de tipo $T_1$, $s$ esté asociada al menos a un recurso mediante la

propiedad $P_1$, la ausencia de una tripla `<s, P1, o>` indica una posible inconsistencia o falta de información relevante. Este patrón permite identificar recursos que no cumplen con esta restricción.
En el Listing 4.16 se presenta una consulta SPARQL que implementa este patrón, donde $T_1$ representa la clase esperada y $P_1$ la propiedad obligatoria.

```
SELECT DISTINCT ?s WHERE {
    ?s rdf:type %%T1%% .
    FILTER NOT EXISTS { ?s %%P1%% ?o}
}
```

Listing 4.16: Patrón TYPRODEP

A modo de ejemplo, en DBpedia, los recursos de la clase `dbo:Place` suelen tener una propiedad `geo:lat` que indica su latitud geográfica. Si instanciamos en el patrón, como se aprecia en 4.17, las variables $T_1$ con `dbo:Place` y $P_1$ con `geo:lat`, podemos detectar aquellas instancias de lugares en DBpedia que carecen de esta información.

```
SELECT DISTINCT ?s WHERE {
    ?s rdf:type dbo:Place .
    FILTER NOT EXISTS { ?s geo:lat ?o}
}
```

Listing 4.17: Ejemplo TYPRODEP

**Definición de la métrica:** En este caso, la restricción se verifica sobre la propiedad denotada por la variable P1, la cual está ligada a la existencia de un sujeto de tipo T1. Por esta razón, definiremos la métrica con granularidad de *Clase* y asociada a T1. Al igual que en los casos anteriores, consideramos la métrica de tipo *ratio*, siendo su valor un número entre 0 y 1.
Para calcular el ratio, se requiere comparar la cantidad de instancias que no poseen la propiedad esperada con el total de instancias pertenecientes a la clase T1.
La consulta del Listing 4.16 devuelve recursos que incumplen la restricción, por lo que debe ser modificada para devolver un valor numérico, como se muestra en el Listing 4.18. El comparador que usaremos para calcular el ratio son los resultados de la consulta de universo definida en el Listing 4.19, que devuelve la cantidad de instancias pertenecientes a la clase T1.

```
SELECT COUNT (DISTINCT ?s) AS ?count WHERE {
    ?s rdf:type %%T1%% .
    FILTER NOT EXISTS { ?s %%P1%% ?o}
}
```

Listing 4.18: Patrón TYPRODEP asociado a la métrica

```
SELECT COUNT (DISTINCT ?s) AS ?universe
WHERE { ?s rdf:type %%T1%% . }
```

Listing 4.19: Consulta para comparar resultado de la métrica asociada al patrón TYPRODEP

El valor de las evaluaciones para esta métrica se computa entonces como los resultados obtenidos de la consulta del Listing 4.18 (a la que llamaremos `resultado_patron`) sobre los resultados obtenidos de 4.19 (`resultado_universo`). Notar que como 4.18 cuenta la cantidad de recursos que incumplen la restricción de rango que se quiere comprobar, lo que nos interesa obtener es el complemento del ratio calculado. Nos queda entonces que el resultado para una métrica asociada al patrón TYPRODEP es:

$$\texttt{resultado} = 1 - \frac{\texttt{resultado\_patron}}{\texttt{resultado\_universo}}$$

Por último, la Tabla 4.8 presenta la dimensión y factor a la que corresponde esta métrica, el tipo de resultado, la granularidad a la que corresponde, y las variables que son necesarias instanciar para aplicarla.

| | |
|---|---|
| Dimensión | Completitud |
| Factor | Completitud de Propiedad |
| Resultado | Ratio [0,1] |
| Granularidad | Clase |
| Variables | $T1$: `rdfs:Class` ó `owl:Class`.<br>$P1$: `rdfs:Property` ó `owl:DatatypeProperty` ó `owl:ObjectProperty` |

Tabla 4.8: Patrón TYPRODEP

### Métrica basada en el patrón PVT

Este patrón se utiliza para verificar restricciones de clasificación de recursos a través de los valores literales de ciertas propiedades. Permite comprobar si los recursos que tienen un valor literal particular en una propiedad (especificada como $P_1$) están correctamente clasificados o agrupados según los valores de otras propiedades ($P_2$). Si un recurso tiene un valor en $P_1$ que corresponde a $V_1$, pero no tiene la propiedad $P_2$, se considera una violación de la restricción de clasificación. En el Listing 4.20 se presenta la consulta SPARQL que implementa este patrón.

```
SELECT DISTINCT ?s WHERE
{
    ?s %%P1%% %%V1%% .
    FILTER NOT EXISTS { ?s %%P2%% ?p }
}
```

Listing 4.20: Patrón PVT

Un ejemplo de aplicación del patrón PVT se puede observar cuando se verifica que los recursos que pertenecen a la categoría `dbc:1907_births` (recursos

relacionados con personas nacidas en 1907) deben necesariamente tener la propiedad `dbo:birthDate` (fecha de nacimiento). Esto se debe a que, según las reglas del conjunto de datos, cualquier recurso que esté clasificado como una persona nacida en 1907 debería tener una fecha de nacimiento asociada.
Para instanciar el patrón, la propiedad $P_1$ sería `dbo:birthDate` y el valor $V_1$ sería la fecha de nacimiento asociada. La propiedad $P_2$ sería `dbo:birthDate`, la cual debería existir para todos los recursos que tienen la propiedad `dbo:category` con el valor `dbc:1907_births`. En el Listing 4.21 se muestra la consulta instanciada.

```
SELECT DISTINCT ?s WHERE
{
    ?s dbo:category dbc:1907_births .
    FILTER NOT EXISTS { ?s dbo:birthDate ?p }
}
```

Listing 4.21: Ejemplo patrón PVT

**No se incluye una métrica asociada a este patrón en la solución.**

**Motivo de descarte:** La métrica asociada al patrón PVT fue descartada dado que su uso depende de una clasificación explícita y consistente de los recursos en los datos, lo que no siempre está disponible o claramente definido en todos los conjuntos de datos RDF. Además, la restricción de tener una propiedad adicional para recursos clasificados de una manera específica puede no aplicarse de manera universal, lo que limita su utilidad general.

Si bien esta métrica se descarta, durante su proceso de evaluación se definieron gran parte de las características que la componen. Por ello, en la Tabla 4.9 se presentan dichas características, con el objetivo de facilitar su futura implementación en caso de que sea de interés incluirla en algún escenario en particular.

| | |
|---|---|
| Dimensión | Consistencia |
| Factor | Consistencia |
| Resultado | No aplica (métrica descartada) |
| Granularidad | Propiedad |
| Variables | $P_1$: `owl:DatatypeProperty` cuyos valores literales clasifican los recursos de alguna manera<br>$V_1$: Valor literal que caracteriza cierta agrupación para la propiedad<br>$P_2$: Propiedad que debería existir para los recursos que cumplan que el valor de la propiedad $P_1$ es $V_1$ |

Tabla 4.9: Patrón PVT

### Métrica basada en el patrón TRIPLE

Este patrón se utiliza para encontrar triplas que podrían ser errores o indicativos de problemas (*bad smells*) en los datos, basándose en la verificación de ciertos valores asociados a una propiedad. Este patrón es útil en grafos de conocimiento cuando se sabe que ciertas propiedades pueden indicar problemas o inconsistencias. En el Listing 4.22 se presenta la consulta SPARQL que implementa este patrón, donde $P_1$ es la propiedad evaluada y $V_1$ es el valor que puede ser un literal o un recurso, dependiendo del contexto.

```
SELECT DISTINCT ?s WHERE { ?s %%P1%% %%V1%% }
```

Listing 4.22: Patrón TRIPLE

En este caso, si el patrón se utiliza con `dbp:wikiPageUsesTemplate` como propiedad y el valor `dbt:Inconsistent_citations`, la consulta encuentra recursos que contienen la plantilla de citas inconsistentes extraída de Wikipedia. En el Listing 4.23 se muestra la consulta instanciada.

```
SELECT DISTINCT ?s WHERE { ?s dbp:wikiPageUsesTemplate dbt:
    Inconsistent_citations }
```

Listing 4.23: Ejemplo patrón TRIPLE

**No se incluye una métrica asociada a este patrón en la solución.**

**Motivo de descarte:** La métrica TRIPLE fue descartada debido a que su aplicabilidad depende demasiado del dominio de los datos y no proporciona una validación automática significativa para todos los casos. Aunque puede detectar ciertos problemas o *bad smells*, su generalización a diferentes conjuntos de datos es limitada.

Si bien esta métrica se descarta, durante su proceso de evaluación se definieron gran parte de las características que la componen. Por ello, en la Tabla 4.10 se presentan dichas características, con el objetivo de facilitar su futura implementación en caso de que sea de interés incluirla en algún escenario en particular.

| | |
|---|---|
| Dimensión | Exactitud |
| Factor | Exactitud Semántica |
| Resultado | No aplica (métrica descartada) |
| Granularidad | Propiedad |
| Variables | $P_1$: `rdfs:Property` ó `owl:DatatypeProperty` ó `owl:ObjectProperty`.<br>$V_1$: Valor que puede ser literal o no |

Tabla 4.10: Patrón TRIPLE

### Métrica basada en el patrón ONELANG

Este patrón busca identificar valores literales que tienen más de una etiqueta de lenguaje asociada para el mismo lenguaje. De acuerdo con el RFC 5646 que describe el uso de las etiquetas de lenguaje, es importante que cada valor literal solo tenga una única etiqueta de lenguaje asociada. El uso de múltiples etiquetas para el mismo lenguaje puede generar inconsistencias en los datos. Este patrón hace uso de la función `lang` de SPARQL, que permite extraer la etiqueta de lenguaje asociada a un valor literal. El RFC completo que describe el uso de las etiquetas de lenguaje se puede encontrar en el siguiente enlace: RFC 5646. En cuanto a la función `lang` de SPARQL, la documentación oficial está disponible aquí.
En el Listing 4.24 se presenta la consulta SPARQL que implementa este patrón, donde $P_1$ es la propiedad evaluada y $V_1$ es el valor literal del lenguaje que se verifica.

```
SELECT DISTINCT ?s WHERE {
    ?s %%P1%% ?c
    BIND ( lang(?c) AS ?l )
    FILTER ( isLiteral(?c) && lang (?c) = %%V1%%)
}
GROUP BY ?s
HAVING (COUNT (?l) > 1)
```

Listing 4.24: Patrón ONELANG

A modo de ejemplo, en un conjunto de datos RDF, el nombre de una persona puede estar representado en inglés con una propiedad `foaf:name`. Si se instancia la propiedad `foaf:name` con el valor de idioma `en` (inglés), esta debería tener solo una etiqueta de idioma asociada. Si el valor literal tiene más de una etiqueta de lenguaje asociada para el mismo idioma, la consulta del patrón detectaría este problema. En el Listing 4.25 se muestra la consulta instanciada:

```
SELECT DISTINCT ?s WHERE {
    ?s foaf:name ?c
    BIND ( lang(?c) AS ?l )
    FILTER ( isLiteral(?c) && lang (?c) = "en")
}
GROUP BY ?s
HAVING (COUNT (?l) > 1)
```

Listing 4.25: Ejemplo patrón ONELANG

**Definición de la métrica:** En este caso, la restricción de exactitud se verifica sobre los valores literales asociados a la propiedad denotada por la variable P1, los cuales deben poseer una única etiqueta de lenguaje para un mismo idioma. Debido a que esta métrica se evalúa sobre propiedades en lugar de clases, definiremos su granularidad como *Propiedad*. Además, consideramos la métrica de tipo *booleano*, donde los valores que una evaluación de ella puede tomar son 0 o 1. Esto implica que tengamos que reescribir el método de medición para poder obtener un valor de este estilo, mediante el comando `ASK` de SPARQL. El Listing 4.26 muestra la definición de este nuevo método.

```
ASK {
    SELECT DISTINCT ?s WHERE {
        ?s %%P1%% ?c BIND ( lang(?c) AS ?l ) FILTER ( isLiteral(?c)
     && lang (?c) = %%V1%%)
    }
    GROUP BY ?s HAVING (COUNT (?l) > 1)
}
```

Listing 4.26: Patrón ONELANG asociado a la métrica

Como buscamos obtener un valor numérico para las métricas, nuevamente el resultado se computa como el complemento del valor numérico correspondiente al *booleano* retornado. Esto es, $1 - \texttt{resultado}$, donde `resultado` es 1 si la consulta devuelve `true`, y 0 en caso contrario.

Por último, la Tabla 4.11 presenta la dimensión y factor a la que corresponde esta métrica, el tipo de resultado, la granularidad a la que aplica y las variables necesarias para su aplicación.

| | |
|---|---|
| Dimensión | Exactitud |
| Factor | Exactitud sintáctica |
| Resultado | Booleano (0 ó 1) |
| Granularidad | Propiedad |
| Variables | $P1$: `owl:DatatypeProperty`.<br>$V1$: Valor literal que identifica el lenguaje, por ejemplo "en" para inglés. |

Tabla 4.11: Patrón ONELANG

### Métrica basada en el patrón RDFSDOMAIN

Este patrón se basa en verificar que la atribución de una propiedad a un sujeto sea válida, es decir, que la clase del sujeto pertenezca al dominio de la propiedad atribuida. En un conjunto de datos RDF bien estructurado, las propiedades deben estar asociadas a un dominio específico, y los sujetos a los que se les asignan estas propiedades deben ser instancias de las clases que forman parte de dicho dominio. Este patrón permite identificar recursos donde los sujetos no cumplen con esta restricción. En el Listing 4.27 se presenta una consulta SPARQL que implementa este patrón, donde $P_1$ es la propiedad evaluada y $T_1$ es la clase que debe pertenecer el sujeto.

**Nota**: El *template* escrito en (Kontokostas y cols., 2014) utiliza una variable $OP$ que se entiende que es un error de tipeo, ya que no se explica su uso ni tiene sentido en la consulta en sí. A continuación se presenta la consulta modificada.

```
SELECT DISTINCT ?s WHERE {
    ?s %%P1%% ?v .
    FILTER NOT EXISTS {
  ?s rdf:type ?t .
  ?t rdfs:subClassOf %%T1%% .
    }
```

```
    FILTER NOT EXISTS {?s rdf:type %%T1%% }
}
```

Listing 4.27: Patrón RDFSDOMAIN

A modo de ejemplo, en DBpedia, la propiedad `dbo:dissolved` debería atribuirse únicamente a instancias de la clase `dbo:SoccerClub`, ya que representa la fecha en la que un club de fútbol se disolvió. Si instanciamos en el patrón la propiedad `dbo:dissolved` y la clase `dbo:SoccerClub`, podemos detectar recursos que contienen sujetos incorrectos. En el Listing 4.28 se muestra la consulta instanciada:

```
SELECT DISTINCT ?s WHERE {
    ?s dbo:dissolved ?v .
    FILTER NOT EXISTS {
  ?s rdf:type ?t .
  ?t rdfs:subClassOf dbo:SoccerClub .
    }
    FILTER NOT EXISTS {?s rdf:type dbo:SoccerClub }
}
```

Listing 4.28: Ejemplo RDFSDOMAIN

En este caso, si la consulta devuelve resultados, significaría que existen sujetos a los que se les ha atribuido la propiedad `dbo:dissolved` pero que no son instancias de `dbo:SoccerClub`, lo que podría indicar una inconsistencia en los datos.

**No se incluye una métrica asociada a este patrón en la solución.**

**Motivo de descarte:** La métrica RDFSDOMAIN fue descartada debido a la falta de definición explícita del dominio de las propiedades en muchos conjuntos de datos RDF. En muchos casos, las propiedades no tienen un dominio declarado formalmente, lo que limita la efectividad de la métrica para detectar posibles inconsistencias.

Si bien esta métrica se descarta, durante su proceso de evaluación se definieron gran parte de las características que la componen. Por ello, en la Tabla 4.12 se presentan dichas características, con el objetivo de facilitar su futura implementación en caso de que sea de interés incluirla en algún escenario en particular.

| | |
|---|---|
| Dimensión | Consistencia |
| Factor | Consistencia |
| Resultado | No aplica (métrica descartada) |
| Granularidad | Propiedad |
| Variables | $P_1$: `rdfs:Property` ó `owl:DatatypeProperty` ó `owl:ObjectProperty`<br>$T_1$: Clase a la que pertenece |

Tabla 4.12: Patrón RDFSDOMAIN

### Métrica basada en el patrón RDFSRANGE

Este patrón se basa en verificar que el objeto de una tripla pertenezca al rango definido para la propiedad utilizada. En un conjunto de datos RDF bien estructurado, cada propiedad debe estar asociada a un rango específico, y los valores asignados a dicha propiedad deben ser instancias de la clase correspondiente. Este patrón permite identificar recursos cuyos valores no cumplen con esta restricción. En el Listing 4.29 se presenta una consulta SPARQL que implementa este patrón, donde $P_1$ es la propiedad evaluada y $T_1$ es la clase esperada para el objeto.
**Nota**: El *template* escrito en (Kontokostas y cols., 2014) utiliza una variable $OP$ que se entiende que es un error de tipeo, ya que no se explica su uso ni tiene sentido en la consulta en sí.

```
SELECT DISTINCT ?s WHERE {
    ?s %%P1%% ?c .
    FILTER NOT EXISTS {
  ?c rdf:type ?t .
    ?t rdfs:subClassOf %%T1%% .
    }
    FILTER NOT EXISTS {?c rdf:type %%T1%% }
}
```

Listing 4.29: Patrón RDFSRANGE

A modo de ejemplo, en DBpedia, la propiedad `dbo:dean` debería referirse únicamente a instancias de la clase `dbo:Person`, ya que representa el decano de una institución educativa. Si instanciamos en el patrón la propiedad `dbo:dean` y la clase `dbo:Person`, podemos detectar recursos que contienen valores incorrectos. En el Listing 4.30 se muestra la consulta instanciada:

```
SELECT DISTINCT ?s WHERE {
    ?s dbo:dean ?c .
    FILTER NOT EXISTS {
  ?c rdf:type ?t .
    ?t rdfs:subClassOf dbo:Person .
    }
    FILTER NOT EXISTS {?c rdf:type dbo:Person }
}
```

Listing 4.30: Ejemplo RDFSRANGE

En este caso, si la consulta devuelve resultados, significaría que existen recursos donde el decano asignado no es una instancia de `dbo:Person`, lo que podría indicar una inconsistencia en los datos.

**No se incluye una métrica asociada a este patrón en la solución.**

**Motivo de descarte:** La métrica basada en el patrón RDFSRANGE fue descartada debido a la falta de especificación de restricciones de rango en muchos conjuntos de datos RDF. En la práctica, no todas las propiedades en modelos RDF tienen un rango definido explícitamente, lo que limita la aplicabilidad de

esta métrica.

Si bien la métrica se descarta, durante su proceso de evaluación se definieron gran parte de las características que la componen. Por ello, en la Tabla 4.13 se presentan dichas características, con el objetivo de facilitar su futura implementación en caso de que sea de interés incluirla en algún escenario en particular.

| | |
|---|---|
| Dimensión | Consistencia |
| Factor | Consistencia |
| Resultado | No aplica (métrica descartada) |
| Granularidad | Propiedad |
| Variables | $P_1$: `rdfs:Property` ó `owl:DatatypeProperty` ó `owl:ObjectProperty`.<br>$T_1$: Clase a la que pertenece |

Tabla 4.13: Patrón RDFSRANGE

### Métrica basada en el patrón RDFSRANGED

Este patrón se basa en verificar que los valores asociados a una DataProperty pertenezcan al tipo de dato esperado según la definición del modelo de datos. En un conjunto de datos RDF bien estructurado, cada DataProperty debe tener valores dentro del rango especificado por un Datatype concreto. Este patrón permite identificar recursos cuyos valores no cumplen con esta restricción. En el Listing 4.31 se presenta una consulta SPARQL que implementa este patrón, donde $P_1$ es la propiedad evaluada y $D_1$ es el tipo de dato esperado.

```
SELECT DISTINCT ?s WHERE {
    ?s %%P1%% ?c.
    FILTER (DATATYPE (?c) != %%D1%%)
}
```

Listing 4.31: Patrón RDFSRANGED

A modo de ejemplo, en DBpedia, la propiedad `dbo:certificationDate` debería contener únicamente valores de tipo `xsd:date`, ya que representa fechas de certificación. Si instanciamos en el patrón la propiedad `dbo:certificationDate` y el tipo de dato `xsd:date`, podemos detectar recursos que contienen valores con un formato incorrecto. En el Listing 4.32 se muestra la consulta instanciada:

```
SELECT DISTINCT ?s WHERE {
    ?s dbo:certificationDate ?c.
    FILTER (DATATYPE (?c) != xsd:date)
}
```

Listing 4.32: Ejemplo RDFSRANGED

En este caso, si la consulta devuelve resultados, significaría que existen recursos con valores de `dbo:certificationDate` que no son del tipo `xsd:date`,

lo que podría indicar inconsistencias en los datos.

**No se incluye una métrica asociada a este patrón en la solución.**

**Motivo de descarte:** La métrica basada en el patrón RDFSRANGED fue descartada debido a la baja especificación de restricciones de tipo en muchos conjuntos de datos RDF. En la práctica, los modelos de datos RDF no siempre declaran explícitamente los rangos de las propiedades de datos, lo que impide una validación automática efectiva.

Si bien la métrica se descarta, durante su proceso de evaluación se definieron gran parte de las características que la componen. Por ello, en la Tabla 4.14 se presentan dichas características, con el objetivo de facilitar su futura implementación en caso de que sea de interés incluirla en algún escenario en particular.

| | |
|---|---|
| Dimensión | Consistencia |
| Factor | Consistencia |
| Resultado | No aplica (métrica descartada) |
| Granularidad | Propiedad |
| Variables | $P_1$: `owl:DatatypeProperty`.<br>$D_1$: Datatype al que debe pertenecer el literal asociado a $P_1$. |

Tabla 4.14: Patrón RDFSRANGED

## Métrica basada en el patrón INVFUNC

Este patrón verifica restricciones de unicidad para el valor de una propiedad en un conjunto de datos RDF. En un modelo de datos bien estructurado, una propiedad que debería ser única no debe tener el mismo valor asociado a diferentes instancias. Este patrón permite identificar recursos que violan esta restricción. En el Listing 4.33 se presenta una consulta SPARQL que implementa este patrón, donde $P_1$ es la propiedad evaluada y $V_1$ es un valor opcional (denotado en la query con comentarios, que serían lo que se necesita incluir en la consulta original para agregar la comprobación de valor).

```
SELECT DISTINCT ?a ?b WHERE {
?a %%P1%% ?v1 . # ?a %%P1%% %%V1%% .
?b %%P1%% ?v2 . # ?b %%P1%% %%V1%% .
FILTER ((str(?v1) = str(?v2)) && (?a != ?b))
}
```

Listing 4.33: Patrón INVFUNC

A modo de ejemplo, en DBpedia, cada recurso no debería compartir la misma página de inicio (`foaf:homepage`) con otro recurso. En el Listing 4.34 se muestra la consulta instanciada:

```
SELECT DISTINCT ?a ?b WHERE {
?a foaf:homepage ?v1 .
?b foaf:homepage ?v2 .
FILTER ((str(?v1) = str(?v2)) && (?a != ?b))
}
```

Listing 4.34: Ejemplo INVFUNC

En este caso, la consulta devuelve dos instancias diferentes, http://dbpedia.org/resource/2020_Shimadzu_All_Japan_Indoor_Tennis_Championships y http://dbpedia.org/resource/2019_Shimadzu_All_Japan_Indoor_Tennis_Championships, que tienen la misma *foaf:homepage* http://alljapan-indoor-tennis.com/.

**Definición de la métrica:** Dado que la restricción de unicidad está asociada a una propiedad en particular, definiremos la métrica con granularidad de *Propiedad*. Además, consideramos la métrica de tipo *booleano*, donde los valores que una evaluación de ella puede tomar son 0 o 1. Esto implica que tengamos que reescribir el método de medición para poder obtener un valor de este estilo, mediante el comando `ASK` de SPARQL. El Listing 4.35 muestra la definición de este nuevo método.

```
ASK {
    SELECT DISTINCT ?a ?b WHERE {
        ?a %%P1%% ?v1 . ?b %%P1%% ?v2 .
        FILTER ((str(?v1) = str(?v2)) && (?a != ?b))
    }
}
```

Listing 4.35: Patrón INVFUNC asociado a la métrica

Como buscamos obtener un valor numérico para las métricas, nuevamente el resultado se computa como el complemento del valor numérico correspondiente al *booleano* retornado. Esto es, $1 - \texttt{resultado}$, donde `resultado` es 1 si la consulta devuelve `true`, y 0 en caso contrario.

Por último, la Tabla 4.15 presenta la dimensión y factor a la que corresponde esta métrica, el tipo de resultado, la granularidad a la que corresponde y las variables que son necesarias instanciar para aplicarla.

| | |
|---|---|
| Dimensión | Consistencia |
| Factor | Consistencia |
| Resultado | Booleano (0 ó 1) |
| Granularidad | Propiedad |
| Variables | $P_1$: `owl:DatatypeProperty`.<br>$V_1$: Valor literal asociado a $P_1$ (opcional). |

Tabla 4.15: Patrón INVFUNC

### Métrica basada en el patrón OWLCARD

Este patrón se basa en verificar restricciones de cardinalidad sobre una propiedad en un conjunto de datos RDF. En un modelo de datos bien estructurado, una propiedad no debería aparecer más veces de lo permitido por una restricción de cardinalidad predefinida. Este patrón permite identificar recursos que violan estas restricciones. En el Listing 4.36 se presenta una consulta SPARQL que implementa este patrón, donde $P_1$ es la propiedad evaluada, $OP$ es el operador de comparación, y $V_1$ es el valor numérico de la restricción.

```
SELECT DISTINCT ?s WHERE {
?s %%P1%% ?c
}
GROUP BY ?s
HAVING (COUNT(?c) %%OP%% %%V1%%)
```

Listing 4.36: Patrón OWLCARD

A modo de ejemplo, en DBpedia, cada recurso puede tener un máximo de 20 etiquetas $rdfs : label$, ya que la plataforma soporta 20 idiomas diferentes. Si instanciamos en el patrón la propiedad `rdfs:label`, el operador `>` y el valor 20, podemos detectar recursos que tienen más etiquetas de las permitidas. En el Listing 4.37 se muestra la consulta instanciada:

```
SELECT DISTINCT ?s WHERE
{ ?s rdfs:label ?c }
GROUP BY ?s
HAVING (COUNT(?c) > 20)
```

Listing 4.37: Ejemplo OWLCARD

En este caso, la consulta podría devolver resultados si existen recursos que superan el límite esperado de etiquetas, lo que indicaría una posible inconsistencia en los datos.

**No se incluye una métrica asociada a este patrón en la solución.**

**Motivo de descarte:** La métrica basada en el patrón OWLCARD fue descartada debido a la falta de restricciones explícitas de cardinalidad en la mayoría de los conjuntos de datos RDF, lo que limita su aplicabilidad práctica. En muchos casos, las restricciones de cardinalidad no están formalmente definidas en los modelos de datos utilizados, lo que impide validar automáticamente la corrección de los resultados obtenidos con este patrón. Además, la interpretación de violaciones de cardinalidad puede depender de criterios específicos del dominio, lo que dificulta su uso generalizado en diferentes contextos.

Si bien la métrica se descarta, durante su proceso de evaluación se definieron gran parte de las características que la componen. Por ello, en la Tabla 4.16 se presentan dichas características, con el objetivo de facilitar su futura implementación en caso de que sea de interés incluirla en algún escenario en particular.

| | |
|---|---|
| Dimensión | Consistencia |
| Factor | Consistencia |
| Resultado | No aplica (métrica descartada) |
| Granularidad | Propiedad |
| Variables | $P_1$: `owl:DatatypeProperty`.<br>$OP$: Operador de comparación para la cardinalidad ($<$, $<=$, $>$, $>=$, $=$, $!=$).<br>$V_1$: Valor numérico de la restricción cardinal. |

Tabla 4.16: Patrón OWLCARD

### Métrica basada en el patrón OWLDISJC

Este patrón se basa en verificar que una instancia no pertenezca simultáneamente a dos clases que hayan sido declaradas como disjuntas mediante el uso del axioma `owl:disjointWith`. En un modelo de datos bien estructurado, si una instancia $s$ es de tipo $T_1$, no debería ser también de tipo $T_2$ si existe una restricción de disjunción entre ambas clases. Este patrón permite identificar recursos que violan esta restricción. En el Listing 4.38 se presenta una consulta SPARQL que implementa este patrón, donde $T_1$ y $T_2$ serán instanciadas por clases declaradas como disjuntas.

```
SELECT DISTINCT ?s WHERE {
?s rdf:type %%T1%% .
?s rdf:type %%T2%% .
}
```

Listing 4.38: Patrón OWLDISJC

A modo de ejemplo, en DBpedia, las clases `dbo:Person` y `dbo:Place` están declaradas como disjuntas. Si instanciamos en el patrón tal como se aprecia en 4.39 las variables $T_1$ y $T_2$ con estas clases, podemos detectar recursos que han sido asignados incorrectamente a ambas.
En particular, se puede observar que la consulta devuelve por ejemplo https://dbpedia.org/page/Cannington_Camp que cumple:

```
<https://dbpedia.org/page/Cannington_Camp rdf:type dbo:Person>
<https://dbpedia.org/page/Cannington_Camp rdf:type dbo:Place>
```

.

```
SELECT DISTINCT ?s WHERE {
?s rdf:type dbo:Person .
?s rdf:type dbo:Place .
}
```

Listing 4.39: Ejemplo OWLDISJC

**Definición de la métrica:** Dado que la restricción de disjunción está asociada a pares de clases, definiremos la métrica con granularidad de *Clase*. Además, consideramos la métrica de tipo *booleano*, donde los valores que una evaluación

de ella puede tomar son 0 o 1. Esto implica que tengamos que reescribir el método de medición para poder obtener un valor de este estilo, mediante el comando `ASK` de SPARQL. El Listing 4.40 muestra la definición de este nuevo método.

```
ASK {
    ?s rdf:type %%T1%% .
    ?s rdf:type %%T2%% .
}
```

Listing 4.40: Patrón OWLDISJC asociado a la métrica

Como buscamos obtener un valor numérico para las métricas, nuevamente el resultado se computa como el complemento del valor numérico correspondiente al *booleano* retornado. Esto es, $1 - \texttt{resultado}$, donde `resultado` es 1 si la consulta devuelve `true`, y 0 en caso contrario.

Por último, la Tabla 4.17 presenta la dimensión y factor a la que corresponde esta métrica, el tipo de resultado, la granularidad a la que corresponde, y las variables que son necesarias instanciar para aplicarla.

| | |
|---|---|
| Dimensión | Consistencia |
| Factor | Consistencia |
| Resultado | Booleano (0 ó 1) |
| Granularidad | Clase (conjunto $\{T_1, T_2\}$) |
| Variables | $T_1$: `rdfs:Class` ó `owl:Class`.<br>$T_2$: `rdfs:Class` ó `owl:Class`. |

Tabla 4.17: Patrón OWLDISJC

### Métrica basada en el patrón OWLDISJP

Este patrón se basa en verificar que una misma instancia no pueda estar relacionada con la misma entidad mediante dos propiedades declaradas como disjuntas mediante la propiedad `owl:disjointProperty`. En un modelo de datos bien estructurado, si una instancia $s$ está relacionada con un valor $v$ a través de la propiedad $P_1$, no debería estar relacionada con el mismo valor $v$ a través de la propiedad $P_2$ si existe una restricción de disjunción entre ambas propiedades. Este patrón permite identificar recursos que violan esta restricción. En el Listing 4.41 se presenta una consulta SPARQL que implementa este patrón, donde $P_1$ y $P_2$ serán instanciadas por propiedades declaradas como disjuntas.

```
SELECT DISTINCT ?s WHERE {
    ?s %%P1%% ?v .
    ?s %%P2%% ?v .
}
```

Listing 4.41: Patrón OWLDISJP

A modo de ejemplo, en DBpedia, las propiedades `dbo:bandMember` y `dbo:birthPlace` deberían ser disjuntas. Esta es una restricción que un experto

de dominio puede inferir, ya que DBpedia no define actualmente restricciones `owl:disjointProperty`. Si instanciamos en el patrón las variables $P_1$ y $P_2$ con estas propiedades, podemos detectar recursos que han sido asignados incorrectamente a ambas. En particular, si la consulta devuelve resultados, significaría que existen instancias que tienen la misma entidad relacionada tanto por `dbo:bandMember` como por `dbo:birthPlace`, lo cual representa una inconsistencia. En el Listing 4.42 se muestra la consulta instanciada:

```
SELECT DISTINCT ?s WHERE {
    ?s dbo:bandMember ?v .
    ?s dbo:birthPlace ?v .
}
```

Listing 4.42: Ejemplo OWLDISJP

En este caso, la consulta no devuelve resultados en DBpedia, lo que indica que no hay violaciones de la restricción para estas propiedades.

Cabe destacar que OWL 1 no soporta restricciones de disjunción entre properties, pero OWL 2 introduce la propiedad `owl:propertyDisjointWith`, lo que permite expresar este tipo de restricciones de manera formal. En la herramienta utilizada, se emplea OWL 2 para garantizar el nivel de expresividad necesario en la implementación de todas las métricas elegidas.

**Definición de la métrica:** Dado que la restricción de disjunción está asociada a pares de propiedades, definiremos la métrica con granularidad de *Propiedad*. Además, consideramos la métrica de tipo *booleano*, donde los valores que una evaluación de ella puede tomar son 0 o 1. Esto implica que tengamos que reescribir el método de medición para poder obtener un valor de este estilo, mediante el comando `ASK` de SPARQL. El Listing 4.43 muestra la definición de este nuevo método.

```
ASK {
    ?s %%P1%% ?v .
    ?s %%P2%% ?v .
}
```

Listing 4.43: Patrón OWLDISJP asociado a la métrica

Como buscamos obtener un valor numérico para las métricas, nuevamente el resultado se computa como el complemento del valor numérico correspondiente al *booleano* retornado. Esto es, $1 - \texttt{resultado}$, donde `resultado` es 1 si la consulta devuelve `true`, y 0 en caso contrario.

Por último, la Tabla 4.18 presenta la dimensión y factor a la que corresponde esta métrica, el tipo de resultado, la granularidad a la que corresponde y las variables que son necesarias instanciar para aplicarla.

| | |
|---|---|
| Dimensión | Consistencia |
| Factor | Consistencia |
| Resultado | Booleano (0 ó 1) |
| Granularidad | Propiedad (conjunto $\{P_1, P_2\}$) |
| Variables | $P_1$: `rdfs:Property` ó `owl:DatatypeProperty` ó `owl:ObjectProperty`.<br>$P_2$: `rdfs:Property` ó `owl:DatatypeProperty` ó `owl:ObjectProperty` (mismo tipo que $P_1$). |

Tabla 4.18: Patrón OWLDISJP

### Métrica basada en el patrón OWLASYMP

Este patrón se basa en verificar que una propiedad declarada como asimétrica no se aplique de manera simétrica sobre dos recursos. Una propiedad es asimétrica si, cuando se cumple para un par de recursos en una dirección, no puede cumplirse en la dirección opuesta. En otras palabras, si existe una relación $< aP_1b >$, no puede existir $< bP_1a >$. Este patrón busca identificar instancias donde una propiedad definida como asimétrica es violada. El Listing 4.44 presenta una consulta SPARQL que implementa este patrón, donde P1 será instanciada por una propiedad declarada como asimétrica.

```
SELECT ?r1 WHERE {
?r1 %%P1%% ?r2 .
?r2 %%P1%% ?r1 .
}
```

Listing 4.44: Patrón OWLASYMP

A modo de ejemplo, si instanciamos en el patrón la variable P1 como `dbo:child` tal que se aprecia en el Listing 4.45 , podemos detectar recursos donde un individuo aparece como hijo y padre de otro individuo, lo cual es una relación inválida en un modelo de datos bien estructurado.
La consulta asociada devuelve por ejemplo `https://dbpedia.org/page/Carmen_Calisto`, que es hija (`dbo:child`) de sí misma.

```
SELECT ?r1 WHERE {
?r1 dbo:child ?r2 .
?r2 dbo:child ?r1 .
}
```

Listing 4.45: Ejemplo OWLASYMP

Cabe destacar que OWL 1 no soporta restricciones de asimetría, pero OWL 2 introduce la propiedad `owl:AsymmetricProperty`, lo que permite expresar este tipo de restricciones de manera formal. En la herramienta utilizada, se emplea OWL 2 para garantizar el nivel de expresividad necesario en la implementación de todas las métricas elegidas.

**Definición de la métrica:** Dado que la restricción de asimetría está asociada a una propiedad, definiremos la métrica con granularidad de *Propiedad*. Además, consideramos la métrica de tipo *booleano*, donde los valores que una evaluación de ella puede tomar son 0 o 1. Esto implica que tengamos que reescribir el método de medición para poder obtener un valor de este estilo, mediante el comando `ASK` de SPARQL. El Listing 4.46 muestra la definición de este nuevo método.

```
ASK {
    ?r1 %%P1%% ?r2 .
    ?r2 %%P1%% ?r1 .
}
```

Listing 4.46: Patrón OWLASYMP asociado a la métrica

Como buscamos obtener un valor numérico para las métricas, nuevamente el resultado se computa como el complemento del valor numérico correspondiente al *booleano* retornado. Esto es, $1 - \texttt{resultado}$, donde `resultado` es 1 si la consulta devuelve `true`, y 0 en caso contrario.

Por último, la Tabla 4.19 presenta la dimensión y factor a la que corresponde esta métrica, el tipo de resultado, la granularidad a la que corresponde, y las variables que son necesarias instanciar para aplicarla.

| | |
|---|---|
| Dimensión | Consistencia |
| Factor | Consistencia |
| Resultado | Booleano (0 ó 1) |
| Granularidad | Propiedad |
| Variables | $P_1$: `owl:ObjectProperty`. |

Tabla 4.19: Patrón OWLASYMP

### Métrica basada en el patrón OWLIRREFL

Este patrón se basa en verificar que una propiedad declarada como irreflexiva no se aplique sobre un mismo recurso. Una propiedad es irreflexiva si no puede relacionar un recurso consigo mismo. Este patrón busca identificar instancias donde una propiedad definida como irreflexiva es violada, es decir, donde un mismo recurso aparece como sujeto y objeto de dicha propiedad. El Listing 4.47 presenta una consulta SPARQL que implementa este patrón, donde P1 será instanciada por una propiedad declarada como irreflexiva.

```
SELECT DISTINCT ?s WHERE { ?s %%P1%% ?s . }
```

Listing 4.47: Patrón OWLIRREFL

A modo de ejemplo, si instanciamos en el patrón la variable P1 como `dbo:parent` tal como se aprecia en el Listing 4.48 , podemos detectar recursos donde el valor asociado a `dbo:parent` es el mismo recurso, es decir, un individuo que aparece como su propio padre.

Esta consulta devuelve varias instancias, entre ellas https://dbpedia.org/page/Chel_Diokno, cuyo `dbo:parent` es el mismo recurso.

```
SELECT DISTINCT ?s WHERE { ?s dbo:parent ?s . }
```

Listing 4.48: Ejemplo OWLIRREFL

**Definición de la métrica:** Dado que la restricción de irreflexividad está asociada a una propiedad, definiremos la métrica con granularidad de *Propiedad*. Además, consideramos la métrica de tipo *booleano*, donde los valores que una evaluación de ella puede tomar son 0 o 1. Esto implica que tengamos que reescribir el método de medición para poder obtener un valor de este estilo, mediante el comando `ASK` de SPARQL. El Listing 4.49 muestra la definición de este nuevo método.

```
ASK {
    ?s %%P1%% ?s .
}
```

Listing 4.49: Patrón OWLIRREFL asociado a la métrica

Como buscamos obtener un valor numérico para las métricas, nuevamente el resultado se computa como el complemento del valor numérico correspondiente al *booleano* retornado. Esto es, $1 - \texttt{resultado}$, donde `resultado` es 1 si la consulta devuelve `true`, y 0 en caso contrario.

Por último, la Tabla 4.20 presenta la dimensión y factor a la que corresponde esta métrica, el tipo de resultado, la granularidad a la que corresponde, y las variables que son necesarias instanciar para aplicarla.

| | |
|---|---|
| Dimensión | Consistencia |
| Factor | Consistencia |
| Resultado | Booleano (0 ó 1) |
| Granularidad | Propiedad |
| Variables | $P_1$: `owl:ObjectProperty`. |

Tabla 4.20: Patrón OWLIRREFL

### 4.3.2. Métricas consideradas del trabajo Structural Quality Metrics to Evaluate Knowledge Graph Quality

A continuación se presentan las métricas consideradas del trabajo mencionado. En este caso, dos de las métricas presentadas por los autores forman parte de la solución propuesta por este proyecto.

#### Métrica basada en instantiated class ratio

Está métrica se basa en la proporción de qué tan usadas son las clases. Es un indicador de qué tan bien (o qué tanto) están siendo usadas las clases. En

Listing 4.50 se presenta la consulta SPARQL utilizada para calcular esta métrica.
La consulta cuenta el número de clases (`?classWithInstances`) que poseen al menos una instancia mediante la propiedad `rdf:type`. Este valor (`?count`) refleja cuántas clases están activamente utilizadas en el KG, permitiendo identificar clases definidas pero no instanciadas.

```
SELECT (COUNT(DISTINCT classWithInstances) AS count)
WHERE {
 ?classWithInstances rdf:type rdfs:Class.
 ?instance rdf:type ?classWithInstances.
}
```

Listing 4.50: Métrica basada en instantiated class ratio

**Definición de la métrica:** En este caso, la restricción de completitud se verifica sobre todo el grafo. Por esta razón, definiremos la métrica con granularidad de *Grafo*. Consideramos la métrica de tipo *ratio*, siendo su valor un número entre 0 y 1.
Para calcular el ratio, se requiere comparar la cantidad devuelta por la consulta con el total de instancias.
El comparador que usaremos para calcular el ratio son los resultados de la consulta de universo definida en el Listing 4.51, que devuelve la cantidad total de clases del universo.

```
SELECT (COUNT(DISTINCT ?class) AS ?universe)
WHERE {
 ?class a rdfs:Class .
}
```

Listing 4.51: Consulta para comparar resultado de la métrica asociada a intantiated class ratio

El valor de las evaluaciones para esta métrica se computa entonces como los resultados obtenidos de la consulta del Listing 4.50 (a la que llamaremos `resultado_consulta`) sobre los resultados obtenidos de 4.51 (`resultado_universo`). Nos queda entonces que el resultado para una métrica asociada a intantiated class ratio es:

$$\texttt{resultado} = \frac{\texttt{resultado\_consulta}}{\texttt{resultado\_cantidad\_clases}}$$

Por último, la Tabla 4.21 presenta la dimensión y factor a la que corresponde esta métrica, el tipo de resultado, la granularidad a la que corresponde, y las variables que son necesarias instanciar para aplicarla.

| Dimensión | Completitud |
|---|---|
| Factor | Completitud de esquema |
| Resultado | Ratio [0,1] |
| Granularidad | Grafo |
| Variables | No necesita |

Tabla 4.21: Instantiated class ratio

### Métrica basada en instantiated property ratio

Esta métrica se basa en la proporción de qué tan usadas son las propiedades. Es un indicador de qué tan bien (o qué tanto) están siendo usadas las propiedades. En Listing 4.52 se presenta la consulta SPARQL utilizada para calcular esta métrica.
La consulta obtiene el número total de propiedades definidas (`?totalProperties`) y el número de propiedades que han sido efectivamente utilizadas en triples (`?instantiatedProperties`). Luego, calcula la proporción entre ambas (`?ratio`), proporcionando una medida de qué tan utilizadas están las propiedades en relación con las definidas en el KG.

```
SELECT (?instantiatedProperties / ?totalProperties AS ?ratio) WHERE {
  { SELECT (COUNT(DISTINCT ?property) AS ?totalProperties) WHERE {
    ?property rdf:type rdf:Property. } }
  {
    SELECT (COUNT(DISTINCT ?instantiatedProperty) AS ?
    instantiatedProperties) WHERE {
      ?subject ?instantiatedProperty ?object.
      ?instantiatedProperty rdf:type rdf:Property.
    }
  }
}
```

Listing 4.52: Métrica basada en instantiated property ratio

**Definición de la métrica:** En este caso, la restricción de completitud se verifica sobre todo el grafo. Por esta razón, definiremos la métrica con granularidad de *Grafo*. Consideramos la métrica de tipo *ratio*, siendo su valor un número entre 0 y 1.
Para calcular el ratio, se requiere comparar la cantidad devuelta por la consulta con el total de instancias.
El comparador que usaremos para calcular el ratio son los resultados de la consulta de universo definida en el Listing 4.53, que devuelve la cantidad total de clases del universo.

```
SELECT (COUNT(DISTINCT ?prop) AS ?universe)
```

```
2 WHERE {?prop a rdf:Property .}
```

Listing 4.53: Consulta para comparar resultado de la métrica asociada a intantiated property ratio

El valor de las evaluaciones para esta métrica se computa entonces como los resultados obtenidos de la consulta del Listing 4.52 (a la que llamaremos `resultado_consulta`) sobre los resultados obtenidos de 4.53 (`resultado_universo`). Nos queda entonces que el resultado para una métrica asociada a intantiated property ratio es:

$$\texttt{resultado} = \frac{\texttt{resultado\_consulta}}{\texttt{resultado\_universo}}$$

Por último, la Tabla 4.22 presenta la dimensión y factor a la que corresponde esta métrica, el tipo de resultado, la granularidad a la que corresponde, y las variables que son necesarias instanciar para aplicarla.

| | |
|---|---|
| Dimensión | Completitud |
| Factor | Completitud de esquema |
| Resultado | Ratio [0,1] |
| Granularidad | Grafo |
| Variables | No necesita |

Tabla 4.22: Instantiated property ratio

### Métrica basada en class instantiation

Esta métrica se basa en medir qué tan instanciadas están las clases.
En el trabajo no se presenta una consulta para definir la métrica, sino que se define una ecuación. La misma se presenta en la ecuación 4.1, donde $ir(c_i)$ es el ratio de clases instanciadas y $d(c_i)$ representa la distancia de la clase $Class$ a $c_i$, es decir, indica cuántos niveles de relaciones `rdfs:subClassOf` existen entre la clase base y la clase específica $c_i$.

$$CI(\text{Class}) = \sum_{i=1}^{n_c} \frac{\text{ir}(c_i)}{2^{d(c_i)}} \tag{4.1}$$

**No se incluye una métrica asociada a este patrón en la solución.**

**Motivo de descarte:** su cálculo requiere consultas complejas en SPARQL que pueden afectar el rendimiento. Obtener el ratio de instanciación y la distancia en la jerarquía implica consultas recursivas y agregadas, lo que en grafos grandes puede generar tiempos de espera prolongados o incluso fallos por timeout. Dado que la aplicación solo utiliza consultas SPARQL y prioriza métricas que puedan calcularse de manera eficiente, se descarta esta métrica por su

alto costo computacional y dificultad de implementación.

Si bien la métrica se descarta, durante su proceso de evaluación se definieron gran parte de las características que la componen. Por ello, en la Tabla 4.23 se presentan dichas características, con el objetivo de facilitar su futura implementación en caso de que sea de interés incluirla en algún escenario en particular.

| | |
|---|---|
| Dimensión | Completitud |
| Factor | Completitud de esquema |
| Resultado | No aplica (métrica descartada) |
| Granularidad | Grafo |
| Variables | No necesita |

Tabla 4.23: Class instantiation

### Métrica basada en subclass property acquisition

Esta métrica se basa en medir la cantidad de propiedades que están definidas exclusivamente en la subclase y no en la clase base. Por ejemplo, no sería lógico que tanto la clase *Entidad* como la clase *Persona* compartieran las mismas propiedades. Para justificar la existencia de la subclase *Persona*, esta debe contar con atributos distintivos, como *esposo/a*, *hijos*, *titulo universitario*, entre otros.

En el trabajo no se presenta una consulta para definir la métrica, sino que se define una ecuación. La misma se presenta en la ecuación 4.2, donde $P_{\text{subclass}}$ representa el conjunto de propiedades definidas en la subclase, mientras que $P_{\text{superclass}}$ corresponde al conjunto de propiedades heredadas desde la superclase. La diferencia entre ambos conjuntos indica la cantidad de propiedades exclusivas de la subclase. Además, $N_i$ representa el número de instancias de la subclase y $N(C)$ es el número total de clases en la ontología.

$$SPA(\text{Ontology}) = \frac{\sum \left(N_i \left(P_{\text{subclass}} - P_{\text{superclass}}\right)\right)}{N(C)} \tag{4.2}$$

**No se incluye una métrica asociada a este patrón en la solución.**

**Motivo de descarte:** su implementación mediante SPARQL puede ser costosa, ya que requiere identificar y comparar propiedades entre clases y subclases, lo que implica consultas complejas. En bases de datos grandes, esto puede generar tiempos de respuesta elevados o incluso fallos por timeout. Dado que la aplicación prioriza métricas eficientes en SPARQL, se descarta esta métrica por

su alto costo computacional y dificultades en la extracción de datos.

Si bien la métrica se descarta, durante su proceso de evaluación se definieron gran parte de las características que la componen. Por ello, en la Tabla 4.24 se presentan dichas características, con el objetivo de facilitar su futura implementación en caso de que sea de interés incluirla en algún escenario en particular.

| | |
|---|---|
| Dimensión | Completitud |
| Factor | Completitud de esquema |
| Resultado | No aplica (métrica descartada) |
| Granularidad | Grafo |
| Variables | No necesita |

Tabla 4.24: Subclass property acquisition

### Métrica basada en subclass property instantiation

Esta métrica evalúa cuántas propiedades se instancian en triplas específicas de la subclase, diferenciándolas de aquellas presentes en la clase base. Esta métrica analiza el uso práctico de los atributos definidos en la subclase, verificando que sus propiedades se utilicen efectivamente en las instancias.

En el trabajo no se presenta una consulta para definir la métrica, sino que se define una ecuación. La misma se presenta en la ecuación 4.3, donde $T_{\text{Class}}$ representa el conjunto de triplas asociadas a la subclase, mientras que $T_{\text{class\_superclass}}$ corresponde a las triplas heredadas de la superclase. La diferencia entre ambos conjuntos indica las triplas exclusivas de la subclase. Además, $N(T_{\text{Class}})$ es el número total de triplas de la subclase.

$$SPI(\text{Class}) = \frac{N\left(T_{\text{Class}} - T_{\text{class\_superclass}}\right)}{N(T_{\text{Class}})} \tag{4.3}$$

**No se incluye una métrica asociada a este patrón en la solución.**

**Motivo de descarte:** su cálculo requiere consultas SPARQL para identificar y contar propiedades instanciadas en cada nivel de la jerarquía, lo que puede ser costoso en bases de datos grandes. La complejidad de estas consultas puede provocar tiempos de espera elevados o fallos por timeout. Dado que la aplicación prioriza métricas eficientes en SPARQL, se descarta esta métrica por su alto costo computacional y dificultades en su ejecución.

Si bien la métrica se descarta, durante su proceso de evaluación se definieron gran parte de las características que la componen. Por ello, en la Tabla 4.25 se presentan dichas características, con el objetivo de facilitar su futura implementación en caso de que sea de interés incluirla en algún escenario en particular.

| | |
|---|---|
| Dimensión | Completitud |
| Factor | Completitud de esquema |
| Resultado | No aplica (métrica descartada) |
| Granularidad | Grafo |
| Variables | No necesita |

Tabla 4.25: Subclass property instantiation

### Métrica basada en el método inverse multiple inheritance

Esta métrica se basa en un método que permite evaluar la simplicidad del grafo. En particular, dicho método mide la frecuencia con la que una clase tiene un número excesivo de superclases, lo que puede dificultar la gestión del KG. Se utiliza el valor inverso para expresar que, cuanto mayor sea la métrica, más simple resulta el KG. En Listing 4.54 se presenta la consulta SPARQL utilizada para calcular esta métrica.
La consulta obtiene el número de superclases distintas (`?nsup`) de cada clase (`?class`) a partir de la propiedad `rdfs:subClassOf`. Luego, calcula el promedio de superclases por clase y aplica su inverso para obtener el valor final de la métrica (`?IMI`).

```
SELECT (1 / (SUM(?nsup) / COUNT(?class)) AS ?IMI)
WHERE {
  {es
    SELECT ?class (COUNT(DISTINCT ?superclass) AS ?nsup)
    WHERE {
      ?class a rdfs:Class .
      OPTIONAL { ?class rdfs:subClassOf ?superclass . }
    }
    GROUP BY ?class
  }
}
```

Listing 4.54: Métrica basada en inverse multiple inheritance

**No se incluye una métrica asociada a este patrón en la solución.**

**Motivo de descarte:** a pesar de poder calcularse con una consulta SPARQL, la implementación de esta métrica requiere agregaciones sobre todas las clases y sus relaciones de herencia, lo que en grafos grandes puede generar tiempos de espera prolongados o fallos por timeout. Dado que la aplicación prioriza métricas eficientes en SPARQL, se descarta esta métrica por su alto costo computacional y potencial impacto en el rendimiento.

Si bien la métrica se descarta, durante su proceso de evaluación se definieron gran parte de las características que la componen. Por ello, en la Tabla 4.26 se presentan dichas características, con el objetivo de facilitar su futura implementación en caso de que sea de interés incluirla en algún escenario en particular.

| | |
|---|---|
| Dimensión | Completitud |
| Factor | Completitud de esquema |
| Resultado | No aplica (métrica descartada) |
| Granularidad | Grafo |
| Variables | No necesita |

Tabla 4.26: Inverse multiple inheritance

### 4.3.3. Métricas consideradas del trabajo Evaluating the Quality of the LOD Cloud: An Empirical Investigation

Como parte de este trabajo, se analizaron también las métricas definidas en (Debattista, Lange, Auer, Cortis, y Dominic, 2018). Sin embargo, todas ellas fueron descartadas al ser (como se interpreta en el título) métricas relacionadas a *Linked Data* y, por lo tanto, fuera del alcance del proyecto. Debido a esto, no se incluyen estas métricas dentro del conjunto de métricas disponibles por defecto en la herramienta implementada. Además, tampoco se incluye un análisis exhaustivo de dichas métricas como parte del trabajo principal. Si el lector está interesado en el análisis efectuado y la justificación de descarte de cada métrica, puede hacer referencia al anexo A.

### 4.3.4. Métricas seleccionadas

Dado que la herramienta desarrollada no conoce las restricciones existentes en el grafo (obtener todas ellas es muy costoso, tanto de desarrollar como de ejecutar al momento de evaluar un grafo), los patrones definidos en nuestro trabajo asumen precondiciones basadas en conocimiento del dominio. Esto implica que el input humano es imprescindible para garantizar la validez de las evaluaciones. Por ejemplo, una máquina sin inteligencia artificial no puede deducir automáticamente reglas fundamentales, como que la fecha de nacimiento de una persona debe preceder a la de su fallecimiento. En este sentido, la definición de las métricas y restricciones depende de un usuario experto en el dominio del grafo, quien conoce tanto las restricciones implícitas como explícitas contenidas en la base de conocimiento.

A modo de ejemplo, consideremos el caso en que se quiere chequear la disyunción entre las instancias de las clases $T_1$ y $T_2$. Se asume que existe una restricción de tipo $owl : disjointWith(T_1, T_2)$, pero la herramienta no lo impone realmente, y por ende, es responsabilidad del usuario de la aplicación chequear la existencia del axioma.

Por otro lado, existen casos donde a partir de un patrón o consulta surgen dos métricas con diferentes granularidades. Estos son los casos de OWLDISJC y OWLDISJG, donde la primera representa el patrón aplicado a dos clases (por

lo que su granularidad es *Clase*), mientras que la segunda representa el patrón aplicado a todas las clases del grafo (por lo que es granularidad *Grafo*). También tenemos el caso de INVFUNC e INVFUNCV, donde la primera verifica las restricciones de unicidad para un conjunto de propiedades, mientras que la segunda verifica lo mismo pero además permite asociarle un valor literal a la comprobación (ambas de granularidad *Propiedad*). Por último, tenemos Instantiated Class Ratio e Instantiated Class Ratio for OWL, donde, como su nombre lo indica, la segunda está pensada para grafos que utilizan OWL.

En la tabla 4.27 se presentan las métricas seleccionadas, con sus respectivas dimensiones, factores, granularidades y variables.

| **Métrica** | **Dimensión** | **Factor** | **Granularidad** | **Variables** |
|---|---|---|---|---|
| COMP | Exactitud | Exactitud Semántica | Propiedad | $P_1$, $P_2$: `owl:DatatypeProperty`.<br>$OP$: Operador de comparación. |
| MATCH | Consistencia | Consistencia | Propiedad | $P_1$: `owl:DatatypeProperty`.<br>$NOP$: Operador de negación ! (opcional).<br>$REGEXP$: Expresión regular. |
| LITRAN | Consistencia | Consistencia | Clase | $T1$: `rdfs:Class` ó `owl:Class`.<br>$P1$: `owl:DatatypeProperty`.<br>$NOP$: Operador de negación !.<br>$V_{min}$: Cota inferior.<br>$V_{max}$: Cota superior. |
| TYPEDEP | Consistencia | Consistencia | Clase | $T1$, $T2$: `rdfs:Class` ó `owl:Class`. |
| TYPRODEP | Completitud | Completitud de Propiedad | Clase | $T1$: `rdfs:Class` ó `owl:Class`.<br>$P1$: `rdfs:Property` ó `owl:DatatypeProperty` ó `owl:ObjectProperty`. |
| ONELANG | Exactitud | Exactitud sintáctica | Propiedad | $P1$: `rdfs:Property` ó `owl:DatatypeProperty` ó `owl:ObjectProperty`.<br>$V1$: Valor literal (ej: "en"). |
| INVFUNC | Consistencia | Consistencia | Propiedad | $P_1$: `owl:DatatypeProperty`.<br>$V_1$: Valor literal (opcional). |
| INVFUNCV | Consistencia | Consistencia | Propiedad | $P_1$: `owl:DatatypeProperty`.<br>$V_1$: Valor literal (opcional). |
| OWLDISJC | Consistencia | Consistencia | Clase | $T_1$, $T_2$: `rdfs:Class` ó `owl:Class`. |
| OWLDISJG | Consistencia | Consistencia | Grafo | $T_1$, $T_2$: `rdfs:Class` ó `owl:Class`. |
| OWLDISJP | Consistencia | Consistencia | Propiedad | $P_1$, $P_2$: `rdfs:Property` ó `owl:DatatypeProperty` ó `owl:ObjectProperty`. |
| OWLASYMP | Consistencia | Consistencia | Propiedad | $P_1$: `owl:ObjectProperty`. |
| OWLIRREFL | Consistencia | Consistencia | Propiedad | $P_1$: `owl:ObjectProperty`. |
| Instantiated Class Ratio | Completitud | Completitud de esquema | Grafo | No necesita. |
| Instantiated Class Ratio for OWL | Completitud | Completitud de esquema | Grafo | No necesita. |
| Instantiated Property Ratio | Completitud | Completitud de esquema | Grafo | No necesita. |

Tabla 4.27: Tabla completa de métricas seleccionadas con su clasificación.

Para terminar, en 4.28 se muestran cuáles métricas fueron implementadas de la misma forma que se plantean originalmente y cuales fueron modificadas en nuestra implementación.

| **Métrica** | **Fue modificada** |
|---|---|
| COMP | Sí |
| MATCH | Sí |
| LITRAN | Sí |
| TYPEDEP | Sí |
| TYPRODEP | Sí |
| ONELANG | Sí |
| INVFUNC | Sí |
| INVFUNCV | Sí |
| OWLDISJC | Sí |
| OWLDISJG | Sí |
| OWLDISJP | Sí |
| OWLASYMP | Sí |
| OWLIRREFL | Sí |
| Instantiated Class Ratio | No |
| Instantiated Class Ratio for OWL | No |
| Instantiated Property Ratio | No |

Tabla 4.28: Tabla completa de métricas y si fueron modificadas o no.

# Capítulo 5

# Herramienta desarrollada

Este proyecto se centra en desarrollar una herramienta que permita evaluar la calidad de KG con las cualidades mencionadas en la Sección 3.7, llenando el vacío que se encontró en la literatura. Para ello, se decidió implementar una aplicación web que es descrita en este capítulo.

## 5.1. Propuesta

Luego de hacer un análisis de cómo debería comportarse la aplicación, se identificaron dos flujos principales. El primero permite ingresar a la aplicación con un nombre y contraseña, crear un modelo de calidad para cualquier KG con una URL pública que permita hacerle consultas SPARQL (aprovechando las funcionalidades de SPARQL descritas en 2.1.4), seleccionar las métricas a medir de un conjunto predefinido y visualizar los resultados de la evaluación en tiempo real. El segundo tiene la capacidad de dar de alta a otros usuarios y editar las dimensiones, factores, métricas y métodos disponibles en el sistema.

Asimismo, se establecieron dos categorías de usuarios: usuarios y usuarios administradores. Los usuarios solo pueden acceder al primer flujo, mientras que los usuarios administradores tienen acceso a ambos. Todos los usuarios comparten el mismo conjunto de dimensiones, factores, métricas y métodos. En caso de que un usuario administrador decida crear, eliminar o editar alguno de estos, todos los usuarios se verán afectados. Para que dos equipos tengan diferentes dimensiones, factores, métricas o métodos se tiene que desplegar la aplicación en dos entornos diferentes, con bases de datos independientes.

Las métricas están implementadas mediante métodos que constan de consultas SPARQL. Para evaluar una métrica, se tiene que ejecutar la consulta SPARQL sobre el *endpoint* proporcionado por el usuario. Estos métodos SPARQL, en algunos casos, definen variables que tienen que ser instanciadas por el usuario. Dado que un usuario puede querer instanciar una misma métrica múltiples veces con variables diferentes, se introduce el concepto "medida". Diremos que una medida es una instanciación del método de una métrica, con

ciertas variables determinadas. En la aplicación, para cada métrica, se permiten hacer varias medidas con diferentes valores para cada variable.

Se espera que los usuarios tengan conocimiento básico sobre cómo funcionan las consultas SPARQL y, sobre todo, del grafo a medir. Es importante que el usuario entienda qué medidas se quieren extraer del grafo, y cómo interpretar sus resultados. Esta no es una herramienta de *profiling*, y en muchas de las métricas predefinidas es necesario instanciar variables, algo difícil de hacer sin conocimiento del KG a evaluar. El conocimiento de SPARQL básico se busca para poder aprovechar al máximo la aplicación, entender cómo es que se mide la métrica y poder entender mejor los resultados, sobre todo cuando estos no son los esperados.

## 5.2. Requerimientos

Para formalizar qué es lo que se espera de la aplicación, en esta Sección se desarrollan los requerimientos funcionales y no funcionales que se tuvieron en cuenta al desarrollarla.

### 5.2.1. Requerimientos funcionales

A continuación, se presentan los requerimientos funcionales que definirán las capacidades y comportamientos esperados del sistema:

1. Como usuario, quiero poder ingresar al sistema al proveer un nombre de usuario y contraseña.
    a) Se deberá mostrar dos *inputs*, uno de usuario y otro de contraseña. En caso de ser válidos, se deberá mostrar todos los modelos de calidad que ese usuario creó en sesiones pasadas y la posibilidad de crear uno nuevo.
    b) En caso de que usuario o contraseña no sean válidos se deberá mostrar un error
2. Como usuario, al ingresar al sistema quiero poder ver mis evaluaciones de calidad ya creadas y un botón para crear una nueva.
    a) Se deberá mostrar un listado de todas las evaluaciones de calidad ya creados, mostrando el nombre del modelo, el *endpoint* SPARQL usado, y en caso de haber seleccionado un grafo nombrado, el grafo nombrado seleccionado.
    b) Además, se deberá mostrar, para cada una de las evaluaciones, su estado. Si se terminaron de calcular todas las medidas de una evaluación deberá decir que terminó. En caso de que alguna medida siga en progreso debe decir que la evaluación está en progreso.
3. Como usuario, al apretar el botón de crear nueva evaluación de calidad, quiero ser redireccionado a una pantalla para definir la nueva evaluación.

   a) Se debe disponer de un *input* para ingresar la URL de un *endpoint* que permita hacer consultas SPARQL a un grafo existente en un sistema externo. El sistema deberá chequear que la URL provista permita consultas SPARQL y en caso de que si, redireccionar a otra pantalla que liste las métricas que el sistema permite medir.

   b) En caso de que la URL ingresada no responda a consultas SPARQL, se deberá mostrar un error al usuario.

   c) Además se debe disponer de un *input* obligatorio donde el usuario le de un nombre a la evaluación para poder identificarla fácilmente.

   d) Por último, se deberá mostrar un *dropdown* opcional, para que, en caso de que el *dataset* tenga distintos grafos nombrados, el usuario pueda seleccionar sobre cual se realizan las consultas.

4. Como usuario, quiero poder seleccionar métricas de diferentes dimensiones y factores con diferentes granularidades para definir qué métricas quiero medir sobre el KG.

   a) El sistema deberá listar todas las métricas disponibles para crear el modelo de calidad para cada dimensión y factor, dando una breve descripción de cada una de ellas.

   b) Cada métrica tendrá que indicar si se espera que el resultado sea un ratio (un real entre 0 y 1) o un booleano representado como 0 o 1.

5. Como usuario, quiero poder instanciar las variables de las métricas seleccionadas para poder terminar de definir el modelo de calidad.

   a) El sistema deberá, para cada variable de cada métrica, tener un *input* para que el usuario ingrese el valor de cada variable. Cada *input* tiene que mostrar todos los valores posibles para el tipo de la variable, por ejemplo, si la variable es de tipo *Clase*, se deben listar todas las clases en un *dropdown* con búsqueda para el usuario seleccione.

   b) De la misma forma, quiero poder hacer varias medidas de una misma métrica con diferentes valores de las variables.

6. Como usuario, quiero poder obtener los resultados de la evaluación de calidad definidos para su análisis.

   a) El sistema deberá mostrar el resultado para cada métrica seleccionada previamente.

   b) El sistema deberá mostrar la consulta que fue realizada para evaluar el resultado de cada medida de cada métrica.

   c) En caso de que alguna de las consultas falle, se deberá mostrar que hubo un error al hacer la consulta. Con esto y la capacidad de ver la consulta realizada, el usuario puede investigar de por qué la consulta falló.

7. Como usuario administrador, quiero poder agregar, editar y borrar dimensiones, factores, métricas y métodos para personalizar mis modelos de calidad.

8. Como usuario administrador, quiero poder agregar, editar y borrar otros usuarios para poder administrar usuarios en mi aplicación.

### 5.2.2. Requerimientos no funcionales

A continuación, se presentan los requerimientos no funcionales que deberá cumplir el sistema:

1. El sistema deberá persistir las dimensiones, factores, métricas y métodos. Y para cada modelo de calidad, se deberán persistir los resultados de cada medida.

2. Una vez instanciado el modelo de calidad, y haberle pedido a la herramienta que calcule los resultados, el usuario deberá poder ver en tiempo real cuando se terminan de calcular, demostrando un buen desempeño en general para realizar este calculo.

3. Al levantar el sistema por primera vez deberá tener en la base de datos persistidas las dimensiones, factores y métricas seleccionadas de la Sección 4.3.4.

## 5.3. Arquitectura de la aplicación

En las dos secciones anteriores, se busca responder la pregunta de qué es lo que se va a implementar. Esta Sección y la siguiente buscan responder la pregunta de cómo se implementó la herramienta. En particular, esta sección presenta su arquitectura, la cual se ilustra en la Figura 5.1.

La arquitectura de la herramienta combina un *frontend* basado en *React*[1] con un *backend* desarrollado en *Django*[2], junto con otras herramientas estándar de este último, para el procesamiento y cálculo de los modelos de calidad. A continuación, se describen en detalle los componentes que conforman el sistema.

**React App:** El *frontend* de la herramienta está implementado con React, una librería moderna para el desarrollo de interfaces de usuario dinámicas. Este componente se encarga de ofrecer una interfaz donde los usuarios pueden definir un modelo de calidad para el grafo y consultar los resultados de las evaluaciones realizadas. Todas las interacciones del usuario, como la definición de métricas y la visualización de resultados, son enviadas al *backend* a través de una *Application Programming Interface* (API).

[1] https://react.dev/
[2] https://www.djangoproject.com/

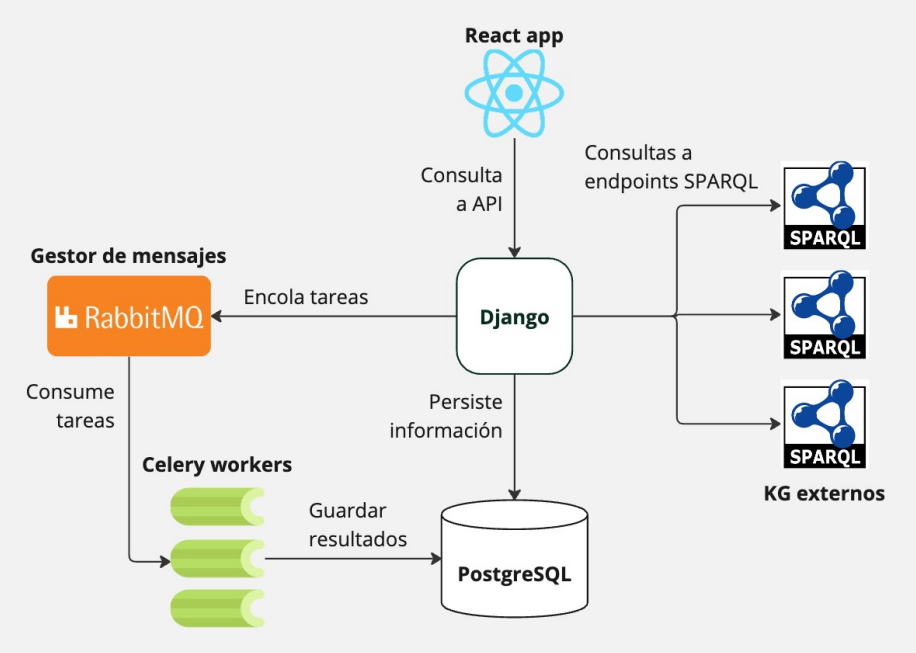



Figura 5.1: Arquitectura de la aplicación

**Django:** Este componente es responsable de gestionar la lógica del negocio, recibir las solicitudes del *frontend* y procesarlas de manera adecuada. *Django* expone una API que permite la comunicación con el *frontend* y facilita la interacción con otros componentes del sistema. Entre las principales responsabilidades del *backend* se encuentran: el encolar tareas que hagan consultas SPARQL para que sean procesadas posteriormente en segundo plano, y responder a las consultas del *frontend*.

**Gestor de mensajes:** *RabbitMQ*[3] actúa como gestor de mensajes entre *Django* y los *workers* de *Celery*[4]. El mismo se encarga de mantener y gestionar la cola de tareas a procesar y luego enviarlas a los diferentes *workers*. Cada vez que el *backend* necesita calcular medidas de métricas de calidad, estas tareas son encoladas en *RabbitMQ*. Los mensajes encolados son posteriormente procesados por los *workers*, asegurando una separación clara entre la lógica principal del *backend* y las operaciones de largo plazo, que se ejecutan de forma asíncrona y se describen en más detalle a continuación.

**Celery Workers:** son los encargados de procesar las tareas asíncronas previamente mencionadas. Estas tareas incluyen el cálculo de medidas de métricas de calidad para los grafos (realizando consultas SPARQL a los

[3] https://www.rabbitmq.com/
[4] https://docs.celeryq.dev/

KG externos) y el procesamiento de los resultados obtenidos. Los *workers* están diseñados para manejar múltiples tareas de manera concurrente, lo que permite escalar el sistema y mantener una alta disponibilidad. Una vez procesadas las tareas, los resultados son enviados a la base de datos *PostgreSQL*[5], donde quedan almacenados para su posterior consulta.

**PostgreSQL:** La base de datos *PostgreSQL* es el componente encargado de almacenar toda la información relevante para la herramienta. Esto incluye las definiciones de los modelos de calidad proporcionados por los usuarios, los resultados de las métricas calculadas y los detalles de las consultas realizadas. Este componente es fundamental para garantizar que la herramienta pueda manejar grandes volúmenes de información y responder de manera rápida a las solicitudes de los usuarios.

**KG externos:** proporcionan los datos necesarios para evaluar la calidad del grafo. Estas bases de datos son accesibles mediante el protocolo SPARQL, lo que permite al *backend* enviar consultas personalizadas para obtener la información requerida. Los resultados de estas consultas son procesados por el *backend* o los trabajadores de *Celery*, dependiendo del caso.

### 5.3.1. Diseño de la Base de datos

Para poder hablar de la aplicación con mayor fluidez, explicando qué es lo que se va a implementar, primero presentaremos los conceptos modelados mediante el modelo entidad relación (MER) que se presenta en la Figura 5.2.

- **Modelo de calidad**: Define modelos de calidad.
- **Usuario**: Gestiona información de los usuarios que interactúan con el sistema, principalmente con los modelos de calidad.
- **Dimensión**: Representa una dimensión específica dentro del sistema.
- **Factor**: Los factores representan subcategorías dentro de una dimensión, permitiendo una clasificación más detallada de las métricas. Un factor está asociado a una dimensión.
- **Métrica**: Representa métricas específicas que permiten evaluar factores dentro de una dimensión. *tipo_resultado* puede ser "booleano" o "ratio" indicando que el resultado esperado de evaluar esa métrica.
- **Método**: Representa métodos de medición utilizados para calcular una métrica. Si bien en la definición una métrica puede tener muchos métodos, para este trabajo se define una relación 1 a 1 de métrica con método. Es decir que si queremos evaluar una misma métrica con dos métodos diferentes, es necesario definir dos métricas distintas, una para cada método. Esto se hizo con el objetivo de acotar el alcance de la aplicación. Los

[5] https://www.postgresql.org/

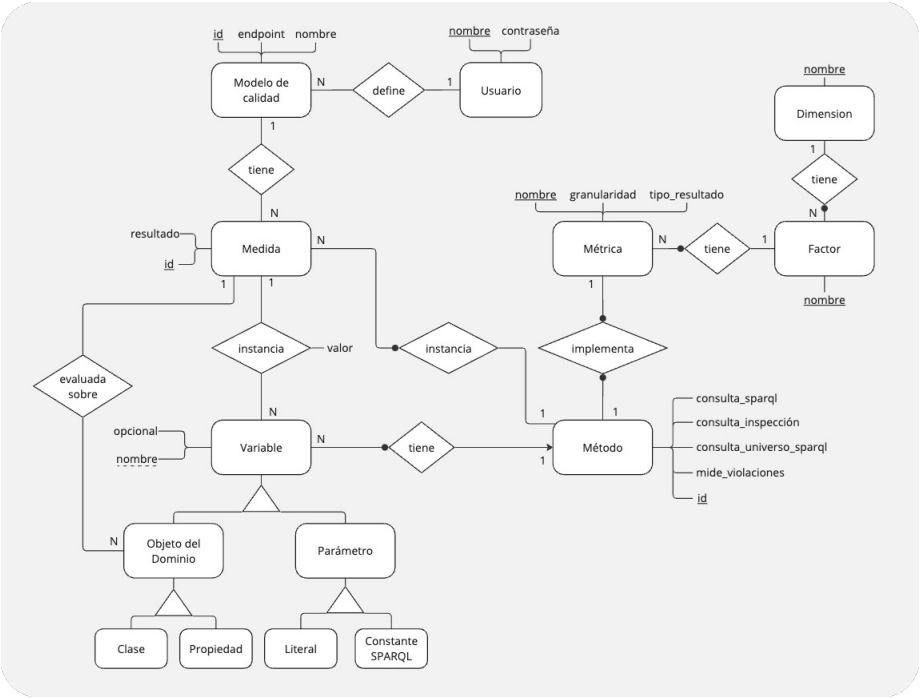



Figura 5.2: Modelo entidad-relación

atributos son usados para el calcular el resultado de aplicar el método a un grafo de conocimiento. Son explicados en detalle en 5.3.2.

- **Medida**: Es una instanciación del método de una métrica, con ciertas variables determinadas, conteniendo el resultado de hacer la consulta SPARQL del método instanciado con las variables instanciadas. Una métrica puede tener varias medidas para un modelo de calidad.

- **Variable**: Define las variables a instanciar para un método de medición específico. Vale la pena destacar que las variables cumplen dos funciones, por un lado, son los parámetros de los métodos, pero también son los objetos del dominio sobre los cuales se va a aplicar la métrica.

- **Objeto del dominio**: Son las variables sobre las que está aplicada la métrica. Además, son las propiedades o clases que utilizadas en las consultas SPARQL.

- **Parámetro**: Son utilizados en las consultas SPARQL para poder definir las Medidas. Algunos ejemplos pueden ser comparadores de SPARQL como `>`,`>=`,`<`,`<=`,`!=`, la negación `!`, o constantes en formato *string*, *int* o *float*.

Un detalle a destacar en el diseño presentado es que no se modela una relación entre la granularidad de la métrica y los objetos del dominio a los que refiere. En cambio, esta relación queda implícita en la definición del método asociado a la métrica, en particular, en las variables que utiliza. A modo de ejemplo, si una métrica tiene granularidad *Clase*, vamos a decir que está aplicada a al menos una de las clases a las que se hace referencia en el método de esta métrica.

Esto puede resultar extraño desde un punto de vista teórico, ya que da lugar a la pregunta "¿Sobre qué está aplicada la métrica?". Sin embargo, en la práctica no se encontró una forma intuitiva de conseguir esta información. Por un lado, no es algo que pueda deducirse automáticamente con un algoritmo (o inteligencia artificial) para una métrica o método. Por otro, no creemos que aporte valor solicitar que el usuario ingrese explícitamente sobre cuál o cuáles objetos de dominio debería aplicar la métrica. Basta con que el usuario sea capaz de señalar cuáles objetos de dominio necesitan ser instanciados para calcular esa métrica.

También vale la pena mencionar que en el código estos nombres están en inglés. Sin embargo, aquí se ponen en español para facilitar la lectura.

### 5.3.2. Backend

Tal como se mencionó en la Sección 5.3, se decidió crear un *backend* utilizando una base de datos *PostgreSQL* y *Django* para la API, en conjunto con herramientas estándar de *Django* como *Celery* y *RabbitMQ*. La ventaja de utilizar un *framework* como *Django* es la facilidad a la hora de crear *endpoints* y modelos para la herramienta, así como la posibilidad de utilizar su herramienta de administrador como sitio para el usuario administrador de la herramienta. A continuación, se describen los puntos principales de esta parte de la herramienta.

**Django admin**

Este es un componente integrado de *Django* que permite la gestión de datos a través de una interfaz administrativa lista para usar. Este módulo, diseñado principalmente para administradores y desarrolladores, proporciona una forma intuitiva de visualizar, añadir, editar y eliminar datos de las bases de datos relacionadas con una aplicación web. El administrador se adapta automáticamente a los modelos definidos, generando formularios y vistas basados en la estructura de datos subyacente.

En nuestro contexto, este módulo puede ser utilizado por los usuarios administradores para agregar nuevos usuarios a la plataforma, así como también nuevas dimensiones, factores y métricas. A su vez, los usuarios administradores que tengan acceso podrán visualizar los diferentes modelos de calidad definidos por el usuario y demás datos relativos al mismo.

**Cálculo de resultados para el modelo de calidad**

Este proceso comienza cuando el usuario finaliza el proceso de definir las *Medidas* dentro de un *Modelo de calidad* y el *frontend* envía la información. El *backend* comienza con la recepción de la solicitud que contiene los datos

necesarios para crear los registros de las *Medidas*. Estos datos son validados utilizando los serializadores correspondientes. Una vez validados, se generan instancias de los objetos *Medidas* y se almacenan en la base de datos. Además, se crean y almacenan los registros asociados de *VariableIntanciada*.

Tras la creación de los registros, se inicia un flujo de trabajo asíncrono para calcular los resultados de cada *Medida*. Cada uno de estos cálculos se realiza mediante tareas de *Celery*, las cuales son programadas para ejecutarse de manera independiente. El proceso de cálculo implica recuperar los detalles de la *Medida* y ejecutar diferentes acciones.

Primero, se identifican las definiciones de *Metodo* y *Variable Intanciada* asociadas a la *Medida*. Estas definiciones se utilizan para construir las consultas SPARQL generadas por sustituir las variables instanciadas en el método asociado la medida. Posteriormente, las consultas SPARQL se ejecutan contra la base de datos de grafos para obtener los resultados necesarios. El tiempo de respuesta al hacer la consulta SPARQL sobre grafos externos se considera el cuello de botella de la aplicación. Asumiendo que se tienen decenas de métodos y variables, el costo, en términos de tiempo de ejecución, asociado a ejecutar consultas sobre la base de datos `PostgreSQL` de la aplicación es despreciable cuando se compara al costo de efectuar consultas sobre un grafo de conocimiento potencialmente grande a través de HTTP. Todas las variables que influyen negativamente en los tiempos de respuesta (posición geográfica, estado de la red, etc.) a cualquier conexión cliente-servidor a través de este método en la web aplican en este contexto. Esto implica que puedan darse errores en la aplicación debido al estado de la red y del servidor que recibe las consultas, por ejemplo *timeouts*.

Dependiendo del tipo de métrica, el resultado puede ser un valor booleano o un cociente. En el caso de métricas booleanas, el resultado podrá ser 0 o 1. Para métricas de tipo cociente, se calcula como la división entre la cantidad obtenida de la consulta principal y el universo relevante de datos, redondeado a tres decimales. En ambos casos 1 indica la máxima calidad posible de la *Medida* y 0 la mínima.

Como esta herramienta busca calcular la calidad, y las métricas mencionadas en la Sección 4.3.1 están planteadas para encontrar violaciones, se utiliza un booleano llamado `mide_violaciones` para determinar si es necesario calcular el complemento del resultado. Por ejemplo, si al calcular una métrica del tipo ratio que busca violaciones se encuentran 5 instancias en la `consulta_universo_sparql` y 2 en `consulta_sparql` el resultado será calculado como

$$1 - \frac{\texttt{consulta\_sparql}}{\texttt{consulta\_universo\_sparql}} = 1 - 2/5 = 3/5$$

El objetivo de esto es que, a mayor cantidad de violaciones, la calidad disminuya, y, en consecuencia, el resultado se acerque más a cero.

Si el cálculo se realiza con éxito, el estado la *Medida* se actualiza a "Finalizado" y se guarda el resultado obtenido. En caso de que se produzca un

error, como un tiempo de espera excedido, la tarea se re intenta de forma automática hasta alcanzar el máximo de intentos configurado. Si todos los intentos fallan, el estado se actualiza a "Fallido". Esto luego es devuelto al *frontend* que lo utiliza para mostrar o no el resultado.

### 5.3.3. Frontend

La parte *frontend* de la aplicación fue creada utilizando la librería *React*, en conjunto con demás librerías de componentes prehechos como *radix-ui*[6] y herramientas como *Vite*[7] para tener un servidor de desarrollo local.

Tal como se describe en detalle en la Sección 5.4.1, la interfaz de usuario permite a los usuarios iniciar sesión e instanciar sus propios modelos de calidad, consumiendo las diferentes API que expone el *backend*.

Para la pantalla donde se visualizan los resultados, el *frontend* realiza solicitudes periódicas al *backend*, consultando por el estado del cálculo. En cada solicitud, el *backend* responde con el estado actual del cálculo de cada medida (pendiente, en progreso, finalizado o fallido). Si el estado recibido es "finalizado", el *frontend* obtiene los resultados calculados y los presenta al usuario en la interfaz. En el caso de un estado "fallido", se muestra un mensaje de error indicando que el cálculo no pudo completarse. Durante los estados "pendiente" o "en progreso", el *frontend* mantiene una animación de carga para informar al usuario que el cálculo sigue ejecutándose.

Esta estrategia permite al sistema gestionar procesos asíncronos de manera eficiente, proporcionando una experiencia fluida al usuario mientras espera los resultados. Al ser una aplicación que no se espera una alta carga de consultas al *backend*, no se optó por alguna solución más sofisticada como *web sockets*.

## 5.4. Flujo de la aplicación

En esta Sección se describen los flujos principales de la aplicación.

### 5.4.1. Flujo de creación y evaluación de modelo de calidad

Este es el flujo principal de la aplicación. Una vez ingresado en la aplicación web, el usuario se encontrará con una pantalla de inicio de sesión que se ve en la Figura 5.3.

Cuando el usuario inicia sesión, se le redirigirá a la pantalla que se muestra en la Figura 5.4, que consta de un listado de todos los modelos de calidad que el usuario ha instanciado previamente. El usuario puede optar por ingresar a la pantalla de detalle de estos modelos o crear uno nuevo.

Para crear un nuevo modelo de calidad, se le pedirá al usuario que ingrese la URL del *endpoint* SPARQL del grafo para realizar consultas, así como un nombre para la evaluación. Además, opcionalmente puede proveerse un grafo

[6] https://www.radix-ui.com/
[7] https://vite.dev/

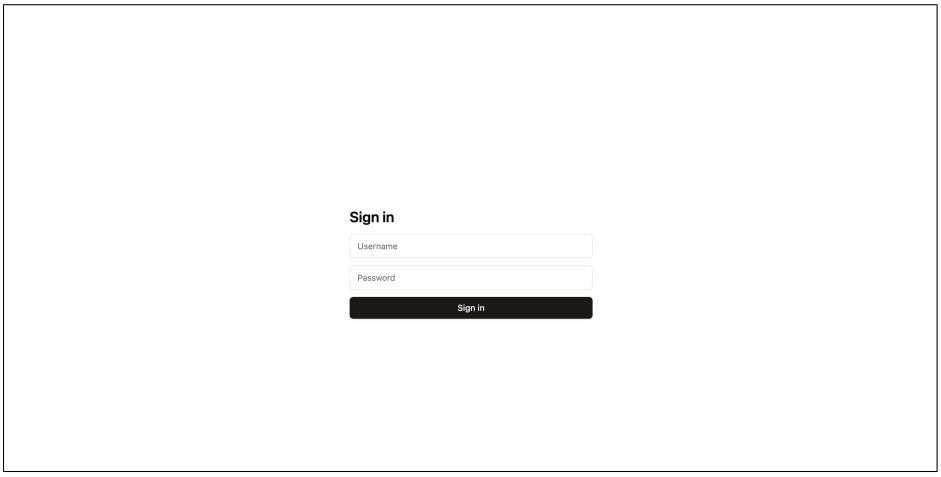



Figura 5.3: Pantalla de inicio de sesión de un usuario


Figura 5.4: Pantalla de listado de los modelos de calidad del usuario

nombrado sobre el cual ejecutar las consultas. El *endpoint* es el utilizado para luego realizar las consultas de las métricas. En este caso, a modo de ejemplo, se realizará la evaluación sobre DBpedia como se ve en 5.5.

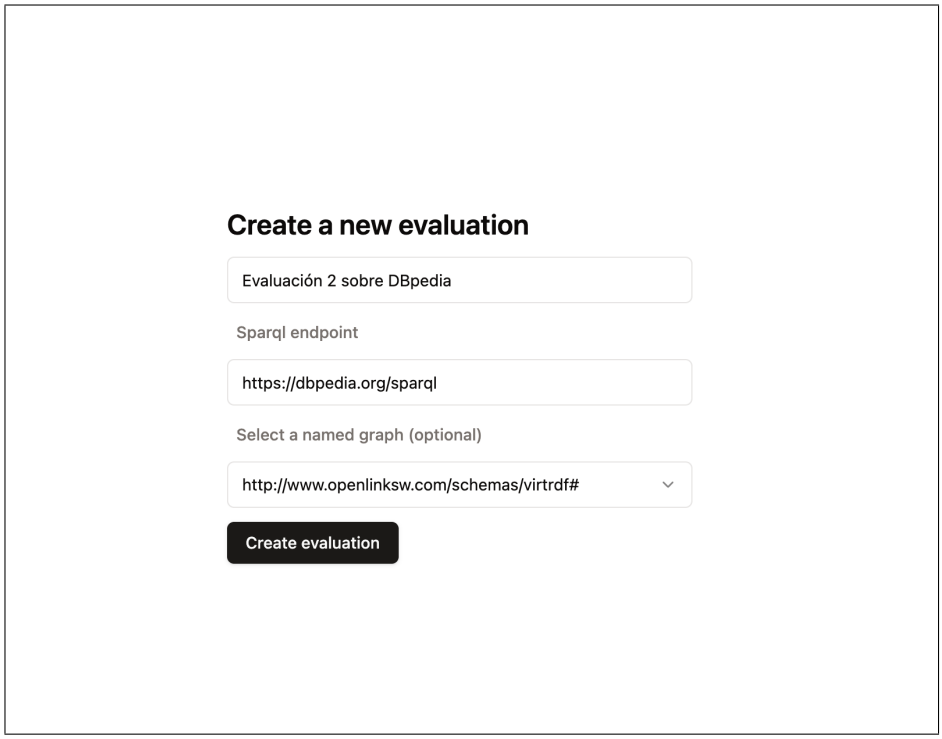



Figura 5.5: Pantalla para crear un nuevo modelo de calidad para un grafo

Una vez completado el paso anterior y validado que el *endpoint* ingresado es correcto, se procede a la pantalla de selección de métricas para el nuevo modelo de calidad. Aquí las métricas son presentadas agrupadas por dimensión y factor a modo de lista, con una breve descripción de cada dimensión, factor y métrica como se ve en 5.6. A su vez, para cada métrica se especifica el tipo del resultado devuelto al calcular la métrica. El usuario selecciona cada métrica por medio de *checkboxes* en las mismas.

Posteriormente a seleccionar las métricas del modelo, se procede a instanciar los parámetros de las mismas. Aquí, dependiendo de la métrica, se deberán seleccionar los operadores, propiedades y clases correspondientes para instanciar la métrica para calcular la calidad. Para el caso de entidades que son propias del grafo, como clases y propiedades, se muestran todas las entidades del mismo. Esto se puede ver en la Figura 5.7.

A su vez, en esta misma pantalla, se puede visualizar la consulta que se va a utilizar luego para calcular el resultado de la métrica como se ve en 5.8, y la posibilidad de agregar más de una instanciación para la métrica, funcionalidad

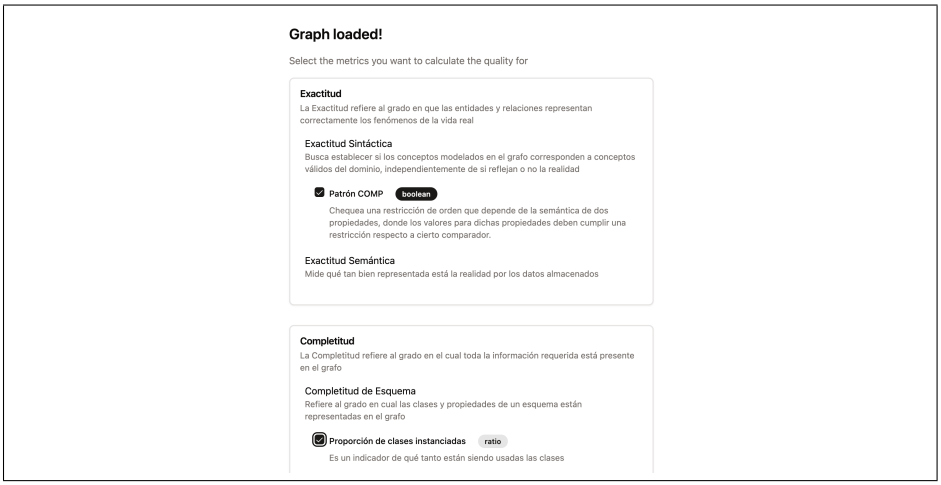



Figura 5.6: Pantalla para seleccionar métricas para el modelo

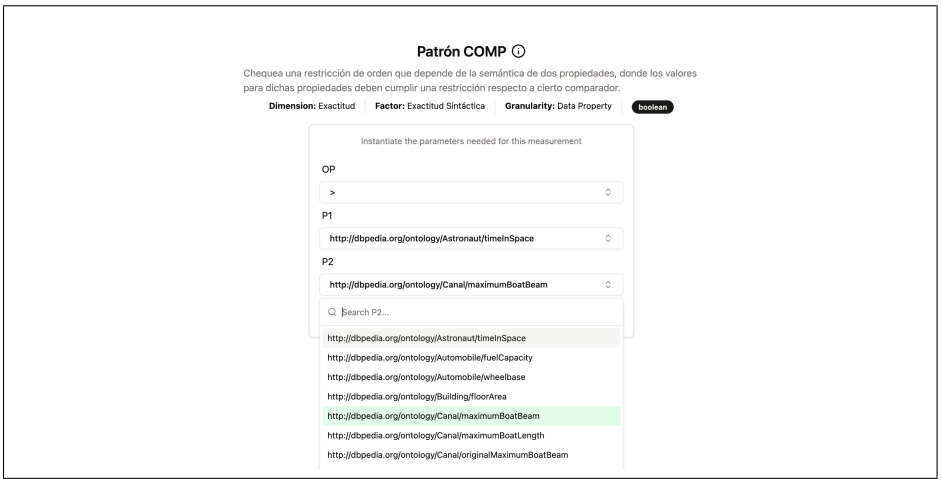



Figura 5.7: Seleccionar parámetros para la métrica seleccionada

que se muestra en 5.9. Este paso se repite para cada métrica seleccionada en el paso anterior.

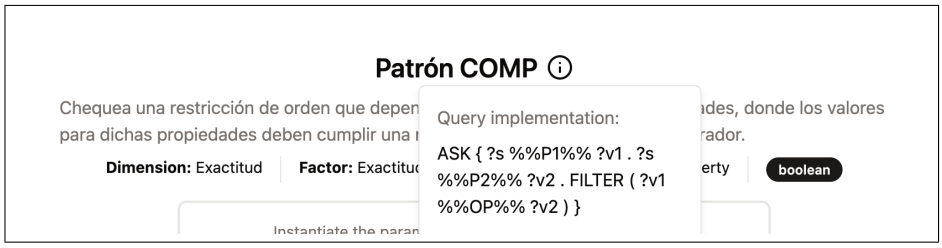



Figura 5.8: Consulta utilizada para calcular el resultado de la métrica

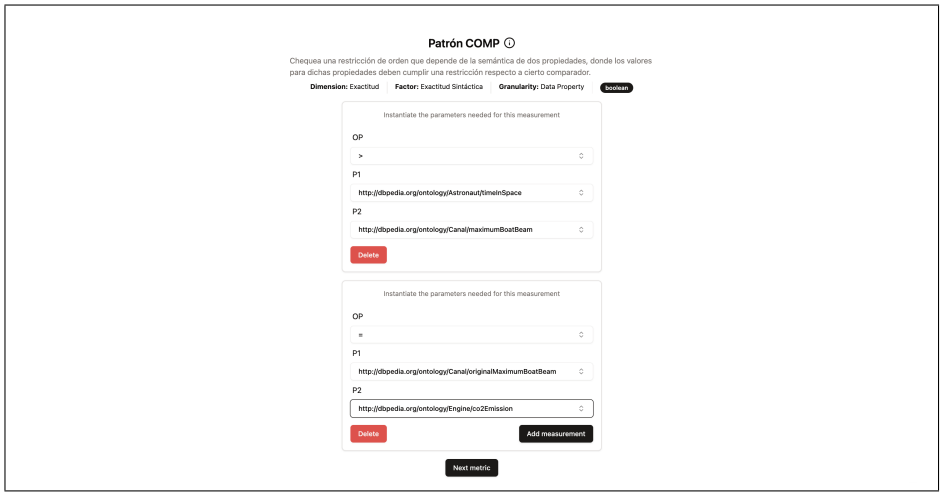



Figura 5.9: Realizar varias medidas para una métrica seleccionada

Una vez instanciados todos los parámetros para todas las métricas seleccionadas, el usuario es redirigido a una pantalla para visualizar un resumen de sus elecciones previo a calcular la calidad del modelo de calidad. A esta pantalla le llamamos pantalla de revisión y se puede ver en 5.10.

Finalmente, el usuario es redirigido a la pantalla de resultado que se muestra en la Figura 5.11 del modelo de calidad. Aquí el usuario puede visualizar el resultado para cada métrica del modelo de calidad en tiempo real. Si el cálculo todavía no concluyó, se mostrará un elemento gráfico que lo indique.

### 5.4.2. Flujos de usuario administrador

A grandes rasgos, los flujos de usuario administrador se pueden separar en dos: crear usuarios, y editar dimensiones, factores, métricas o métodos.

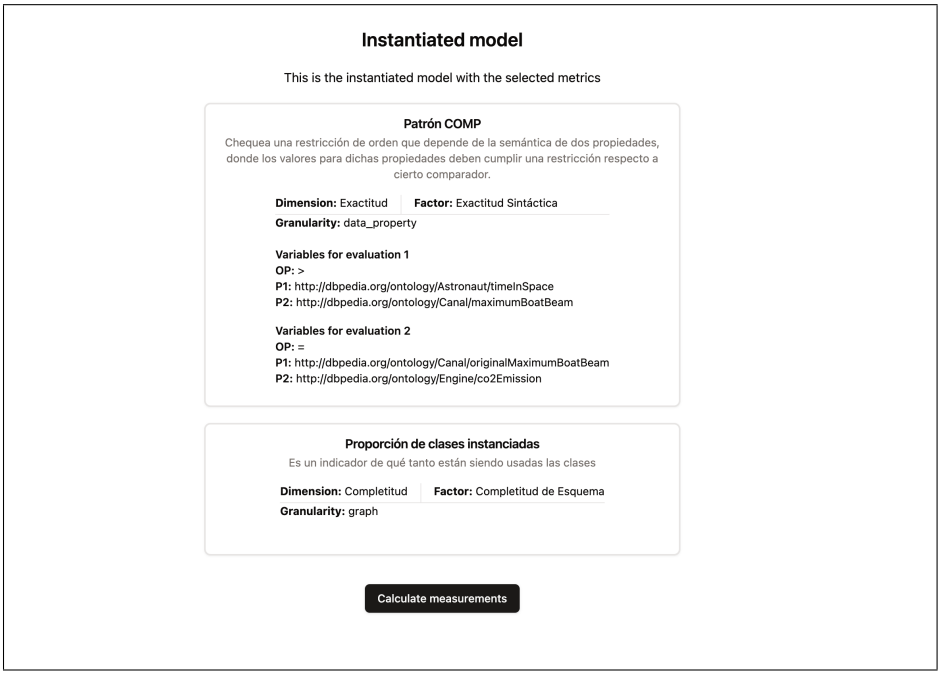



Figura 5.10: Pantalla de revisión del modelo de calidad previo a calcular su calidad

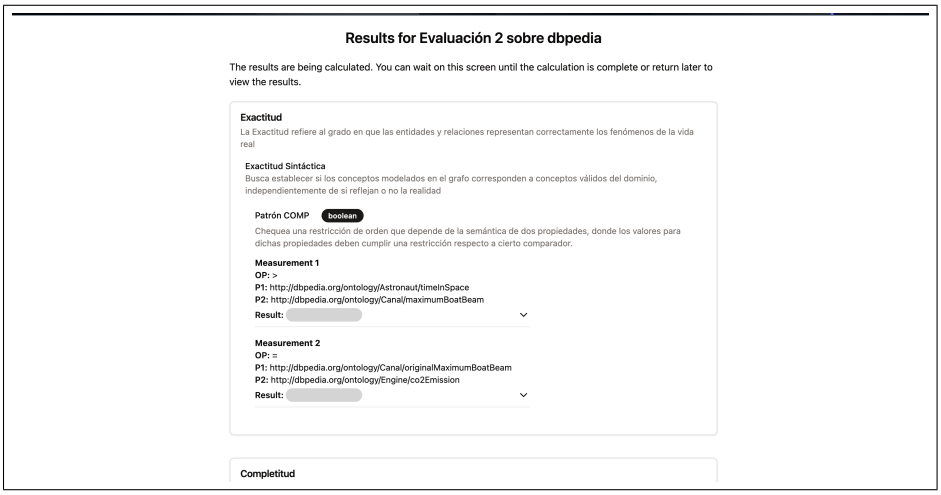



Figura 5.11: Pantalla de resultados con métricas todavía en proceso

La Figura 5.12 muestra un listado de todos los usuarios del sistema. Aquí, el usuario administrador puede crear, eliminar o editar usuarios.

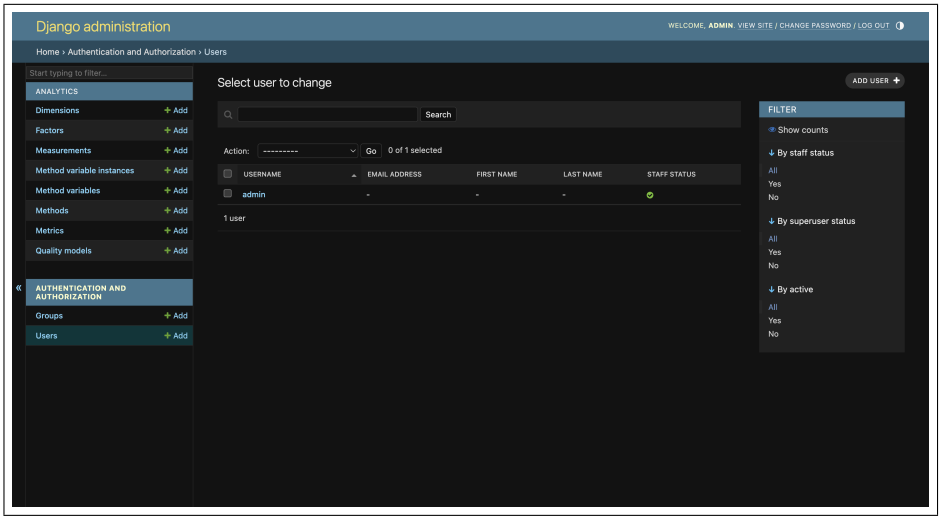



Figura 5.12: Pantalla de gestión de usuarios para el usuario administrador

En la Figura 5.13 se ve un listado de las dimensiones. Luego, al hacer clic en alguna de ellas, se va a la pantalla 5.14 donde se ve cómo se pueden editar las propiedades de la dimensión. Esto se puede hacer para todos los modelos que se ven en la lista de la izquierda.

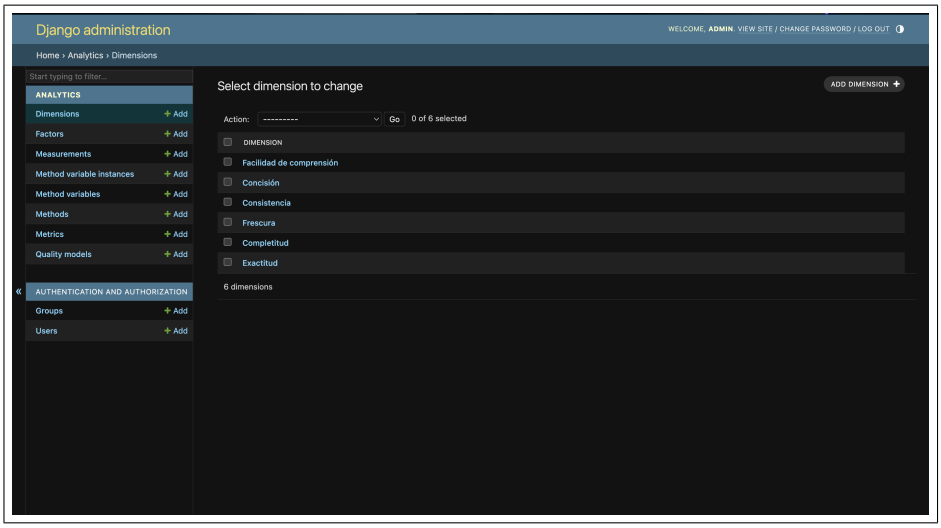



Figura 5.13: Pantalla de listado de dimensiones

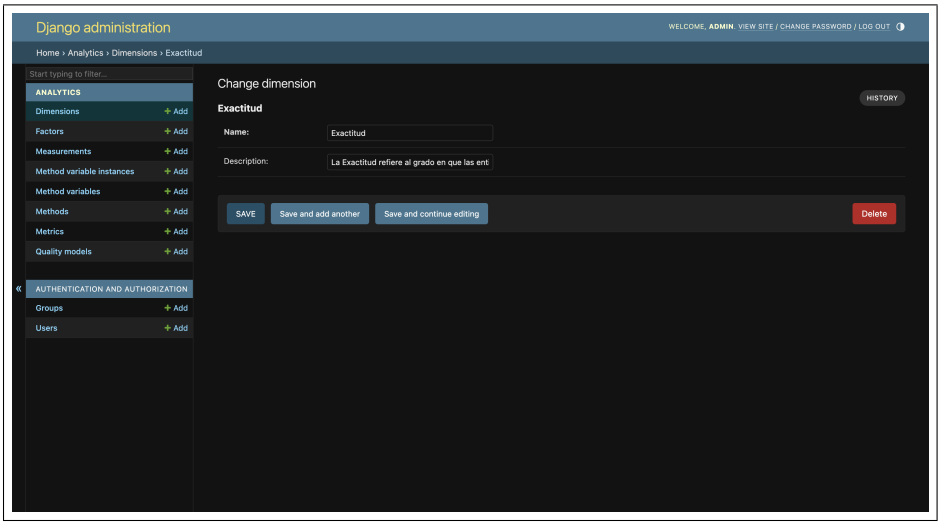



Figura 5.14: Pantalla de detalle para las dimensiones

# Capítulo 6

# Pruebas y experimentos

Para realizar la experimentación de la herramienta, se ha diseñado un proceso automatizado que permite aplicar un modelo de calidad a partir de las métricas disponibles sobre bases de datos conocidas. La metodología implementada aplica el modelo de calidad mencionado sobre dos bases de conocimiento, generando así mediciones que pueden ser utilizadas para detectar inconsistencias o evaluar la coherencia de la información almacenada. Estas bases de conocimiento fueron DBpedia e Hito[1]; también fue considerado utilizar Yago, pero finalmente no fue incluido debido a que emplea un esquema basado en Schema.org[2], el cual es externo. Actualmente, nuestra herramienta solo admite esquemas definidos dentro de la propia ontología, por lo que Yago queda fuera del alcance del análisis realizado. No obstante, se prevé incorporar en trabajos futuros la capacidad de evaluar bases de conocimiento que utilicen esquemas externos, lo que permitiría incluir Yago en futuras experimentaciones (tal como se menciona más a detalle en el Capítulo 7).
En la Tabla 6.1 se presentan las especificaciones del equipo utilizado para realizar las evaluaciones.

| | |
|---|---|
| Procesador | Apple M2 Pro |
| Memory | 16 GB |
| Sistema operativo | Mac Sonoma 14.5 |

Tabla 6.1: Especificaciones del equipo utilizado en las evaluaciones

## 6.1. Evaluación sobre DBPedia

DBpedia es un proyecto de extracción y estructuración de datos procedentes de Wikipedia[3], permitiendo su reutilización en aplicaciones semánticas y de

[1] https://hitontology.eu/
[2] https://schema.org/
[3] https://es.wikipedia.org/wiki/Wikipedia:Portada

datos enlazados. Su objetivo principal es transformar el contenido de Wikipedia en un formato legible por máquinas, facilitando el acceso y la interconexión con otras fuentes de datos.
Este proyecto extrae información en formatos como RDF y SPARQL, lo que permite a los desarrolladores y sistemas acceder a datos estructurados. La base de conocimiento abarca múltiples dominios, incluyendo geografía, ciencia, cultura y economía, y se actualiza periódicamente con los cambios en Wikipedia.
DBpedia se utiliza en inteligencia artificial, procesamiento del lenguaje natural, motores de búsqueda y sistemas de recomendación. Además, es una pieza clave en la web semántica, permitiendo la interconexión con otras bases de datos como Wikidata y OpenCyc.

Al momento de evaluar este grafo de conocimiento, se utilizaron todas las métricas definidas, con excepción de los patrones IPR (*Instanced Property Ratio*) e ICR (*Instanced Class Ratio*). Lo primero se debe a que la consulta asociada al patrón es compleja y al momento de ejecutarse el volumen de los datos que cumplen el patrón de grafo buscado es muy alto, por lo que se genera un `timeout`. Para el caso del segundo, lo que sucede es que en la ontología de DBPedia se definen y utilizan clases OWL (`owl:Class`) y no clases RDF (`rdfs:Class`), por lo que se utiliza la métrica OWLICR (*Instanced class ratio for OWL*) en vez de ICR.
En cuanto a las medidas para cada métrica, las variables elegidas representan las presentadas en los ejemplos de la Sección 4.3, así como también otras medidas que surgieron a partir de explorar DBpedia mediante consultas SPARQL.

En la Tabla 6.2 se presenta el modelo de calidad definido y evaluado sobre este grafo, así como los resultados individuales de cada medida y el tiempo de ejecución correspondiente. Tal como se menciona en la Subsección 5.3.2 de la presentación de la herramienta, el tiempo de ejecución es del orden del tiempo de respuesta del servicio al cual se le realiza la consulta SPARQL. El *endpoint* SPARQL consultado es `https://dbpedia.org/sparql` y el grafo *default* sobre el que se evalúa es `http://dbpedia.org`.

Para mejorar la legibilidad en la tabla se utilizan los siguientes prefijos:

- dbo: `http://dbpedia.org`
- foaf: `http://xmlns.com/foaf/0.1/`
- geo: `http://www.w3.org/2003/01/geo/wgs84_pos#`

| Dimensión | Factor | Métrica | Variables instanciadas | Resultado | Tiempo de ejecución (s) |
|---|---|---|---|---|---|
| Completitud | Completitud de Esquema | Instantiated Class Ratio for OWL | | 0.315 | 8.10 |
| Consistencia | Consistencia | Patrón INVFUNC | P1: foaf:homepage | 0 | 1.37 |
| Consistencia | Consistencia | Patrón INVFUNC | P1: dbo:capital | 0 | 1.16 |
| Consistencia | Consistencia | Patrón LITRAN | T1: dbo:Person<br>P1: dbo:height<br>Vmin: 0.4<br>Vmax: 2.5 | 1 | 1.98 |
| Consistencia | Consistencia | Patrón MATCH | P1: dbo:isbn<br>NOP: !<br>REGEXP: ^([iIsSbBnN 0-9-])*$ | 0 | 1.04 |
| Consistencia | Consistencia | Patrón MATCH | P1: dbo:postalCode<br>NOP: !<br>REGEXP: ^[0-9]5$ | 0 | 1.04 |
| Consistencia | Consistencia | Patrón OWLASYMP | P1: dbo:child | 0 | 1.06 |
| Consistencia | Consistencia | Patrón OWLDISJC | T1: dbo:Person<br>T2: dbo:Place | 0 | 0.97 |
| Consistencia | Consistencia | Patrón OWLDISJCG | | 0.855 | 8.95 |
| Consistencia | Consistencia | Patrón OWLDISJP | P1: dbo:bandMember<br>P2: dbo:birthPlace | 1 | 1.07 |
| Consistencia | Consistencia | Patrón OWLIRREFL | P1: dbo:child | 0 | 1.07 |
| Consistencia | Consistencia | Patrón OWLIRREFL | P1: dbo:parent | 0 | 1.42 |
| Consistencia | Consistencia | Patrón TYPEDEP | T1: foaf:Person<br>T2: dbo:Person | 1.000 | 7.53 |
| Consistencia | Consistencia | Patrón TYPRODEP | T1: dbo:Person<br>P1: dbo:birthDate | 0.436 | 11.41 |
| Consistencia | Consistencia | Patrón TYPRODEP | T1: dbo:Place<br>P1: geo:Lat | 0.891 | 3.29 |

| **Dimensión** | **Factor** | **Métrica** | **Variables instanciadas** | **Resultado** | **Tiempo de ejecución (s)** |
|---|---|---|---|---|---|
| Exactitud | Exactitud Sintáctica | Patrón COMP | P1: dbo:deathDate<br>P2: dbo:birthDate<br>OP: ¡ | 0 | 4.16 |
| Exactitud | Exactitud Sintáctica | Patrón COMP | P1: dbo:latestReleaseDate<br>P2: dbo:releaseDate<br>OP: ¿ | 0 | 1.09 |
| Exactitud | Exactitud Sintáctica | Patrón COMP | P1: dbo:demolitionDate<br>P2: dbo:buildingStartDate<br>OP: ¡ | 0 | 1.20 |
| Exactitud | Exactitud Sintáctica | Patrón ONELANG | P1: rdfs:label<br>V1: en | 1 | 0.99 |
| Exactitud | Exactitud Sintáctica | Patrón ONELANG | P1: foaf:name<br>V1: en | 0 | 32.42 |

Tabla 6.2: Modelo de calidad y resultados para la evaluación de DBpedia con tiempos de ejecución

### Análisis de los resultados

En base a los resultados obtenidos en la Tabla 6.2, se pueden identificar diversas implicancias sobre la completitud, consistencia y exactitud de los datos presentes en DBpedia.

En cuanto a la completitud, la proporción de clases OWL instanciadas se encuentra en un valor de 0.315, lo que sugiere que aproximadamente un tercio de las clases definidas en el esquema son instanciadas. Este valor podría indicar una cobertura limitada del esquema en los datos actuales, lo que podría ser una señal de que algunas clases definidas en el modelo de datos no están siendo suficientemente representadas en las instancias.
Al ejecutar la consulta de la cantidad de clases del universo del Listing 4.51, obtenemos que la cantidad total de clases es 1568. Luego, al ejecutar la consulta de inspección presentada en el Listing 6.1, la cual permite obtener la cantidad de clases que no están instanciadas, obtenemos que la cantidad es 494 (coincidiendo con el resultado de la métrica).

```
SELECT (COUNT(DISTINCT ?classWithInstances) AS ?
    classesWithInstances)
FROM <http://dbpedia.org> WHERE {?instance a ?classWithInstances .
    ?classWithInstances a owl:Class .}
```

Listing 6.1: Consulta que devuelve la cantidad de clases no instanciadas en DBpedia

Respecto a la consistencia, se observan tanto patrones que se cumplen correctamente como otros que no lo hacen. Por ejemplo, se logró el cumplimiento completo de la restricción de la función inversa sobre `foaf:homepage` (valor 1) mediante el patrón INVFUNC. Sin embargo, en otros patrones como INVFUNC sobre `dbo:capital`, LITRAN sobre `dbo:height`, MATCH sobre `dbo:isbn` y `dbo:postalCode`, no se cumple la restricción. A pesar de estos fallos, hay ciertos patrones que muestran un cumplimiento alto, como OWLDISJCG (0.855) y TYPRODEP sobre `geo:Lat` (0.891), lo cual indica que, aunque algunas restricciones no se cumplen en su totalidad, muchas de ellas son respetadas en gran medida.
Si tomamos por ejemplo el caso de OWLDISJCG, podemos ejecutar la consulta presentada en Listing 6.2, la cual nos devuelve instancias que violan la restricción de `owl:disjointWith`. Aquí podemos observar que la consulta retorna instancias que violan esta restricción, en particular tenemos el caso de https://dbpedia.org/page/Benjamin_Marsh, el cual tiene como `rdf:type` tanto a `dbo:Person` como a `dbo:Place`.

```
SELECT DISTINCT ?s
FROM <http://dbpedia.org> WHERE { ?s rdf:type <http://dbpedia.org/
    ontology/Person> . ?s rdf:type <http://dbpedia.org/ontology/
    Place> . }
```

Listing 6.2: Consulta que devuelve las instancias que violan la restricción de `owl:disjointWith`

En términos de exactitud, se observa que las comparaciones de fechas, como las de los patrones COMP sobre `dbo:deathDate` y `dbo:birthDate`, presentan inconsistencias (valores 0). Además, en el patrón ONELANG, se encontró que el valor de `rdfs:label` está correctamente definido en inglés, mientras que `foaf:name` no lo está.
Si tomamos por ejemplo el caso de COMP, podemos ejecutar la consulta presentada en Listing 6.3, la cual nos devuelve instancias que violan la restricción de que la fecha de nacimiento sea menor que la de muerte. Aquí podemos observar que la consulta retorna instancias que violan esta restricción, en particular tenemos el caso de https://dbpedia.org/page/Robert_Opron, el cual tiene como `dbo:birthDate` 1932-02-22 (22 de febrero de 1932) y como `dbo:deathDate` 1922-02-21 (21 de febrero de 1922).

```
SELECT ?s
FROM <http://dbpedia.org>
WHERE { ?s <http://dbpedia.org/ontology/deathDate> ?v1 . ?s <http
    ://dbpedia.org/ontology/birthDate> ?v2 . FILTER ( ?v1 < ?v2 ) }
```

Listing 6.3: Consulta que devuelve instancias cuya `deathDate` sea menor que su `birthDate`

## 6.2. Evaluación sobre Hito

Hito es una ontología diseñada para describir sistemáticamente los sistemas de aplicación y productos de software en el ámbito de la tecnología de la información en salud (Health IT). Su objetivo principal es proporcionar una terminología coherente que facilite la comunicación y comparación de diferentes soluciones de software en el sector sanitario (Hito, 2025).

Esta ontología permite describir productos de software de código libre y abierto (LIFOSS)[4] en medicina y atención sanitaria, así como estudios que los evalúan, utilizando un conjunto definido de clases y sus relaciones (Hito, 2025).

Hito se publica como Linked Open Data (LOD)[5], lo que permite que las descripciones de LIFOSS sean consultadas y visualizadas mediante herramientas de la Web Semántica, como navegadores RDF y consultas SPARQL (Hito, 2025).

El proyecto Hito es una colaboración entre equipos de investigación de Austria y Alemania, y se ha desarrollado para apoyar descripciones sistemáticas de software médico (Hito, 2025).

Al tratarse de un grafo de conocimiento de menor volumen que DBpedia, la cantidad de métricas seleccionadas para evaluar Hito es más limitada. Esto se debe, por ejemplo, a cuestiones como falta de definición de restricciones. En este caso, las métricas evaluadas para el grafo fueron Proporción de clases OWL instanciadas, Métrica basada en *INVFUNC*, Métrica basada en *LITRAN*, Métrica basada en *OWLASYMP*, Métrica basada en *OWLDISJC*, Métrica basada en

[4] https://efmi.org/workinggroups/lifoss-libre-free-and-open-source-software/
[5] https://lod-cloud.net/

*OWLIRREFL*, Métrica basada en *TYPEDEP*, Métrica basada en *TYPRODEP* y Métrica basada en *ONELANG*.

En la Tabla 6.3 se presenta el modelo de calidad definido y evaluado sobre este grafo, así como los resultados individuales de cada medida y el tiempo de ejecución de la misma. Tal como se menciona en la Subsección 5.3.2 de la presentación de la herramienta, el tiempo de ejecución es del orden del tiempo de respuesta del servicio al cual se le realiza la consulta SPARQL. El *endpoint* SPARQL consultado es `https://hitontology.eu/sparql` y el grafo sobre el que se evalúa es `http://hitontology.eu/ontology`.

| Dimensión | Factor | Métrica | Variables instanciadas | Resultado | Tiempo de Ejecución (s) |
|---|---|---|---|---|---|
| Completitud | Completitud de Esquema | Instantiated Class Ratio for OWL | | 0.767 | 1.70 |
| Consistencia | Consistencia | Patrón INVFUNC | P1: hitOntology:internalId | 0 | 0.73 |
| Consistencia | Consistencia | Patrón INVFUNC | P1: hitOntology:spUsedInOuCitSctid | 0 | 0.81 |
| Consistencia | Consistencia | Patrón LITRAN | T1: hitOntology:Study<br>P1: hitOntology:publishedInYear<br>Vmin: 1970<br>Vmax: 2025 | 1 | 1.43 |
| Consistencia | Consistencia | Patrón OWLASYMP | P1: hitOntology:softwareProductComponent | 1 | 0.73 |
| Consistencia | Consistencia | Patrón OWLDISJC | T1: hitOntology:QuasiExperimentalStudy<br>T2: hitOntology:ValidationStudy | 1 | 0.83 |
| Consistencia | Consistencia | Patrón OWLIRREFL | P1: hitOntology:softwareProductComponent | 1 | 0.86 |
| Consistencia | Consistencia | Patrón TYPEDEP | T1: hitOntology:ValidationStudy<br>T2: hitOntology:Study | 1 | 1.49 |
| Consistencia | Consistencia | Patrón TYPEDEP | T1: hitOntology:FeatureCitation<br>T2: hitOntology:EnterpriseFunctionCitation | 0 | 1.43 |
| Consistencia | Consistencia | Patrón TYPRODEP | T1: hitOntology:FeatureCitation<br>P1: hitOntology:fCitClassifiedAs | 0.657 | 1.53 |
| Consistencia | Consistencia | Patrón TYPRODEP | T1: hitOntology:Client<br>P1: hitOntology:client | 0 | 1.60 |
| Exactitud | Exactitud Sintáctica | Patrón ONELANG | P1: rdfs:label<br>V1: en | 1 | 0.72 |

Tabla 6.3: Modelo de calidad y resultados para la evaluación de Hito

### Análisis de los resultados

En base a los resultados obtenidos en la Tabla 6.3, se pueden identificar diversas implicancias sobre la completitud, consistencia y exactitud de los datos analizados.

En cuanto a la completitud del esquema, la proporción de clases OWL instanciadas se encuentra en un valor de 0.767, lo que indica que aproximadamente el 76.7 % de las clases definidas en el esquema están representadas en los datos. Este valor es significativamente superior al observado en el análisis previo en DBpedia (0.315), lo que sugiere que el esquema en este caso tiene una cobertura más amplia y es más representativo en términos de instanciación de clases.
Al ejecutar la consulta de la cantidad de clases del universo del Listing 4.51, obtenemos que la cantidad total de clases es 43. Luego, al ejecutar la consulta de inspección presentada en el Listing 6.1, la cual permite obtener la cantidad de clases que no están instanciadas, obtenemos que la cantidad es 33 (coincidiendo con el resultado de la métrica).

Respecto a la consistencia, se identifican distintos patrones con diferentes niveles de cumplimiento. Por un lado, el cumplimiento del patrón INVFUNC sobre `internalId` y `spUsedInOuCitSctid` presenta valores de 0, lo que indica que estas relaciones no respetan la restricción. Sin embargo, en otros patrones, como LITRAN sobre `hitOntology:publishedInYear`, se observa un cumplimiento total (valor 1), asegurando que los valores se encuentran dentro del rango definido entre 1970 y 2025. De manera similar, los patrones OWLASYMP sobre `softwareProductComponent`, OWLDISJC sobre `QuasiExperimentalStudy` y `ValidationStudy`, y OWLIRREFL sobre `softwareProductComponent` muestran valores de 1, indicando que estas restricciones se respetan completamente. En términos de dependencias tipológicas, se observan diferencias en el cumplimiento de los patrones TYPEDEP y TYPRODEP. En particular, TYPEDEP muestra un cumplimiento total entre `ValidationStudy` y `Study` (valor 1), mientras que entre `FeatureCitation` y `EnterpriseFunctionCitation` no se cumple (valor 0). Para TYPRODEP, el cumplimiento varía: en `FeatureCitation` y `fCitClassifiedAs` se observa un valor de 0.657, indicando un cumplimiento parcial, mientras que en `Client` y `client` no se cumple la restricción (valor 0).
Si tomamos por ejemplo el caso de TYPRODEP, podemos ejecutar la consulta presentada en Listing 6.4, la cual nos retorna instancias tales que son instancias de `hito:FeatureCitation` y no tienen la propiedad `hito:fCitClassifiedAs`. Aquí podemos observar que la consulta retorna instancias que violan esta restricción, en particular, tenemos el caso de
https://hitontology.eu/ontology/3LGM2FeatMultilayerView, el cual tiene como `rdf:type` `hito:FeatureCitation` y no tiene la propiedad `hito:fCitClassifiedAs`.

```
SELECT DISTINCT ?s
FROM <http://hitontology.eu/ontology>
WHERE { ?s rdf:type <http://hitontology.eu/ontology/FeatureCitation
    > . FILTER NOT EXISTS { ?s <http://hitontology.eu/ontology/
```

```
    fCitClassifiedAs> ?o} }
```

Listing 6.4: Consulta que devuelve las instancias que violan la restricción evaluada en TYPRODEP

En términos de exactitud sintáctica, el patrón ONELANG se cumple completamente en `rdfs:label` con el idioma inglés especificado (valor 1), lo que asegura la coherencia lingüística en los datos, al igual que en DBpedia.

# Capítulo 7

# Conclusiones y Trabajo Futuro

En este capítulo se hará un análisis entre los objetivos propuestos y los resultados obtenidos, haciendo una comparación de la aplicación presentada con las herramientas mencionadas en la Sección 3. Luego, se hablará de aspectos a mejorar de la herramienta y posible trabajo futuro.

## 7.1. Conclusiones

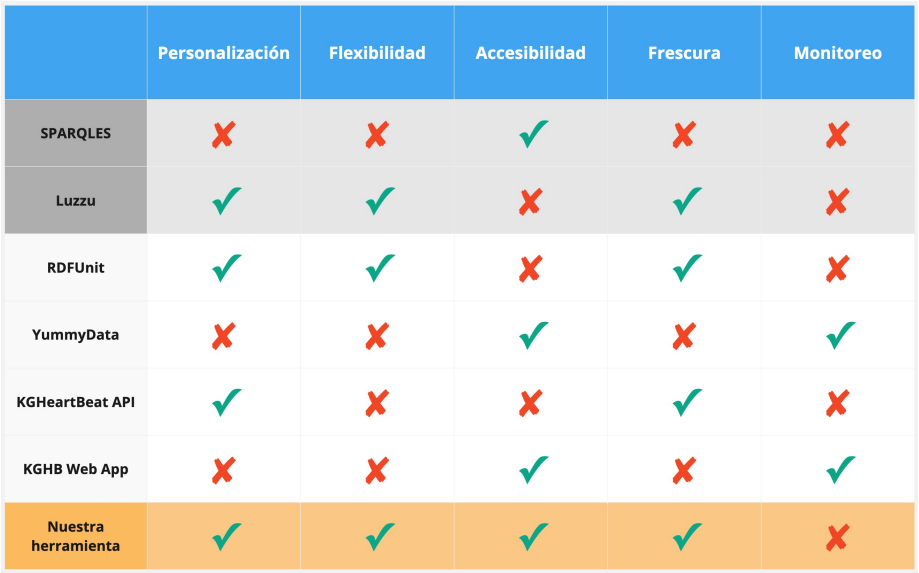
Se concluye que se cumplieron los tres objetivos planteados en la Sección 1.2.

Primero, en la Sección 3 se realizó un análisis comparativo de las herramientas que también buscan evaluar la calidad de datos en KG disponibles en la literatura, cumpliendo con el primer objetivo planteado de investigar el estado del arte.

También se cumplió el segundo objetivo, ya que en la Sección 4 se analizan dimensiones, factores, granularidades, métricas y métodos disponibles en la literatura actual, y se definen las que se van a usar en este trabajo. Se realizó el trabajo de clasificar todas las métricas utilizadas, asignándole dimensión, factor y granularidad a cada una, teniendo además que corregir y modificar algunas consultas SPARQL.

Por último, luego de presentar la aplicación en la Sección 5 y los resultados de instanciar un modelo de calidad en DBPedia y calcular los resultados en la Sección 6, estamos en condiciones de hacer un análisis comparativo de nuestra herramienta con las mencionadas en la Sección 3. Como se muestra en la tabla 7.1, nuestra herramienta cumple con cuatro de las cinco cualidades planteadas, siendo la única en alcanzar este número y, por lo tanto, la que cumple con más cualidades entre las evaluadas.

Más allá de la comparación numérica de cualidades, vemos valor en tener una aplicación que conste de una aplicación web que pueda ser usada por todo



| | Personalización | Flexibilidad | Accesibilidad | Frescura | Monitoreo |
|---|---|---|---|---|---|
| SPARQLES | ✘ | ✘ | ✓ | ✘ | ✘ |
| Luzzu | ✓ | ✓ | ✘ | ✓ | ✘ |
| RDFUnit | ✓ | ✓ | ✘ | ✓ | ✘ |
| YummyData | ✘ | ✘ | ✓ | ✘ | ✓ |
| KGHeartBeat API | ✓ | ✘ | ✘ | ✓ | ✘ |
| KGHB Web App | ✘ | ✘ | ✓ | ✘ | ✓ |
| Nuestra herramienta | ✓ | ✓ | ✓ | ✓ | ✘ |

Figura 7.1: Tabla comparativa de las herramientas analizadas y nuestra aplicación

tipo de usuarios, no solo programadores. Que se puedan editar, crear y borrar dimensiones, factores, métricas y métodos para poder personalizar qué es lo que se quiere medir y cómo. Que se puedan evaluar los KG a demanda, permitiendo a un usuario evaluar su KG, encontrar errores, arreglar los errores, volver a evaluar su grafo y ver los errores resueltos en tiempo real. Y que todo esto se pueda hacer para cualquier *endpoint* SPARQL de acceso público. Por esto, podemos decir que se construyó una aplicación que llena un vacío en la literatura, cumpliendo con el tercer objetivo planteado.

## 7.2. Trabajo futuro

Como sucede con todo producto de software, durante el proceso de implementación se detectaron posibles nuevas funcionalidades, así como mejoras de las que se plantearon originalmente, que por una cuestión de alcance quedaron por fuera de esta primera versión de la aplicación. En esta sección se pretende dar cuenta y dejar registro de ellas para facilitar futuras iteraciones del producto.

Dividimos dichas mejoras en dos categorías, en relación a la importancia e impacto en el producto final que consideramos que tienen. Por un lado tenemos las mejoras necesarias, o *must-have*, aquellas que consideramos que deberían agregarse en futuras iteraciones. Por otro, las funcionalidades opcionales, o *nice-to-have*, que mejoran algún aspecto de la aplicación y/o la interacción de los usuarios con ella, pero no resultan imprescindibles.

Una aclaración pertinente respecto de las categorizaciones es que las mejoras

que se mencionan no tienen un orden entre las que integran la misma categoría. Lógicamente, las mejoras necesarias son de mayor prioridad que las opcionales, pero no se define un orden de prioridad entre miembros de la misma categoría.

### 7.2.1. Mejoras necesarias

**Monitoreo:** Una funcionalidad de la cual la aplicación carece con respecto a otras herramientas existentes es la de **Monitoreo** (ver la Figura 7.1). Dado que esto marca una brecha respecto de lo ofrecido en las demás herramientas, consideramos que es una de las principales funcionalidades a tener en cuenta para futuras iteraciones. Dicho esto, vale destacar que la comparación puede efectuarse en nuestra aplicación de manera manual, instanciando el mismo modelo de calidad sobre el mismo grafo en diferentes momentos y comparando los resultados.

**Simplificar el despliegue:** Con el objetivo de facilitar la adopción de la herramienta y mejorar su portabilidad entre sistemas, se plantea implementar una solución totalmente basada en el uso de tecnologías de contenedores (por ejemplo, Docker) como una mejora necesaria. Actualmente, la base de datos, el *broker* de *rabbit* y los *workers* de *celery* se encuentran dockerizados, pero no así el *backend* y la aplicación *React*. Una vez que se tenga la aplicación desplegada, se puede crear un usuario administrador y, utilizando la interfaz de *Django Admin*, gestionar usuarios adicionales y personalizar las dimensiones, factores, métricas y métodos. La idea es que cada grupo de investigación que esté de acuerdo con un conjunto de dimensiones, factores, métricas y métodos despliegue su propia instancia de la aplicación.

**Validación de código SPARQL y seguridad** : Actualmente la herramienta utiliza la librería SPARQLWrapper para interactuar con los *endpoints* SPARQL y enviar las consultas desde el *backend* de la aplicación. Sin embargo, una funcionalidad que esta librería no ofrece es la validación de la sintaxis de las consultas que recibe. Idealmente, esto es necesario no sólo para validar las consultas previo a enviarlas, sino para defenderse de ataques de inyección en las variables instanciadas por los usuarios. La funcionalidad de validación es un recurso importante al momento de que un usuario quiera definir sus propias consultas, ya que ahorra tiempo perdido en errores de validación (que ahora sólo se detectan una vez que se interactúa con un *endpoint*) y su *debugueo*.

**Esquema y vocabulario:** Existen esquemas y vocabularios predefinidos que pueden ser incluidos y utilizados en un grafo sin necesidad de ser definidos explícitamente. Un ejemplo es Schema.org, que es el esquema que una de las bases de conocimiento abierto más grandes disponibles, YAGO utiliza. El uso de esquemas de este tipo dificulta la tarea de poder decidir cómo instanciar las variables de muchos de los métodos de medición de nuestra herramienta, dado

que el vocabulario utilizado no puede obtenerse directamente, sino que está embebido en el esquema externo. Por ejemplo, una consulta para obtener todas las clases `rdfs:Class`, como la que se utiliza para poder desplegar opciones para las variables de tipo `Clase` de los métodos en DBPedia, no devolverá resultados para YAGO. Esto se debe a que en dicho *dataset* no hay una definición explícita de triplas con la forma `<?pred rdf:type rdfs:Class>`. Para atacar este problema, convendría implementar mecanismos para que un usuario pueda definir el esquema utilizado en un grafo, por ejemplo, mediante un archivo *Turtle* que lo implemente.

### 7.2.2. Mejoras opcionales

**Permitir archivos como grafo de entrada para la aplicación:** Actualmente, la aplicación solo permite ingresar un enlace a un *endpoint* SPARQL para interactuar con un KG. Sin embargo, resultaría conveniente incorporar también la funcionalidad de subir archivos, por ejemplo en formato *Turtle*, que sean un *dump* de un KG.

**Representación de los resultados:** Una de las cualidades interesantes de KGHeartBeat (Pellegrino y cols., 2024), que podrían replicarse en nuestra aplicación, es la posibilidad de asociar una gráfica o representación visual característica a cada métrica. Ahora mismo, en la pantalla de resultados, solo se ve un resultado numérico que no es tan ilustrativo.

**Exportar resultados a CSV:** La funcionalidad es auto-descriptiva. Implementar la posibilidad de descargar el resultado de una evaluación en formato CSV facilitaría su posterior uso en herramientas de visualización de datos más potentes.

**Grafos de esquema nombrados:** Un patrón comúnmente utilizado por *datasets* RDF es definir el esquema utilizado como un grafo nombrado. En estos casos pueden surgir problemas al intentar proveer de opciones para instanciar las variables de los métodos de medición de ciertas métricas. Por ejemplo, si el *grafo default* del *dataset* está configurado como el *merge* de todos los grafos nombrados, y el vocabulario RDFS se encuentra embebido en el *dataset*, una consulta que busca obtener clases RDFS (`rdfs:Class`) sobre dicho *grafo default* obtendría `rdf:Property` como uno de los resultados. Sucede entonces que, en ciertos contextos, el usuario puede querer restringir su vocabulario a elementos de un esquema dado. Con este objetivo en mente, convendría tener la posibilidad de asociar a una evaluación de la herramienta el nombre del grafo nombrado que contiene el esquema, para evitar este tipo de problemas.

**Granularidad de la métrica asociada al patrón COMP:** La métrica 4.4 se definió con un resultado booleano, pero podría considerarse una implementación en forma de ratio. Para ello, basta definir la consulta de universo 7.1 y

redefinir la consulta del patrón para que cuente la cantidad de incumplimientos encontrados, de manera análoga a las demás métricas de tipo ratio.

```
SELECT COUNT DISTINCT ?s AS ?universe
WHERE {
    ?s %%P1%% ?v1 .
    ?s %%P2%% ?v2 .
}
```

Listing 7.1: Patrón METRIC con resultado de ratio

# Referencias


Abicht, K. (2023). Owl reasoners still useable in 2023. *IEEE Access*, *9*, 31322–31339.

Baker, T., y Prud'hommeaux, E. (2022). *Shape expressions (shex) primer.* https://shexspec.github.io/primer/.

Carlo Batini, M. S. (2016). *Data and information quality - dimensions, principles and techniques.* Springer.

Debattista, J., Auer, S., y Lange, C. (2016). Luzzu–a framework for linked data quality assessment. En *2016 ieee tenth international conference on semantic computing (icsc)* (pp. 124–131).

Debattista, J., Lange, C., Auer, S., Cortis, y Dominic. (2018). Evaluating the quality of the lod cloud: An empirical investigation. *Semantic Web*, *9*(6), 859–901.

Emilio Cristalli, F. S., y Marotta, A. (2018). Data quality evaluation in document oriented data stores. En *Advances in conceptual modeling* (p. 309-318).

Hito. (2025). *Hito project.* https://hitontology.eu/. (Accessed: 2025-20-02)

Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutiérrez, C., ... Zimmermann, A. (2021). *Knowledge Graphs* (n.º 22). Springer. Descargado de https://kgbook.org/ doi: 10.2200/S01125ED1V01Y202109DSK022

Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J., Cornelissen, R., y Zaveri, A. (2014). Test-driven evaluation of linked data quality. En *Proceedings of the 23rd international conference on world wide web* (pp. 747–758).

Lorena Etcheverry, Adriana Marotta, Camila Sanz, Sebastián García. (2025). *Curso: Calidad de datos e información.* https://eva.fing.edu.uy/course/view.php?id=1073. (Accessed: 2025-13-02)

Lorena Etcheverry, A. V. (2012). Views over rdf datasets: A state-of-the-art and open challenges. , *9*, 31322–31339.

Mohammed, S., Ehrlinger, L., Harmouch, H., Naumann, F., y Srivastava, D. (2024). Data quality assessment: Challenges and opportunities. *arXiv preprint arXiv:2403.00526*.

Pellegrino, M. A., Rula, A., y Tuozzo, G. (2024). Kgheartbeat: An open source tool for periodically evaluating the quality of knowledge graphs. En *International semantic web conference* (pp. 40–58).

Seo, S., Cheon, H., Kim, H., y Hyun, D. (2022). Structural quality metrics to evaluate knowledge graph quality. *Semantic Web*.

Shearer, R. D., Motik, B., y Horrocks, I. (2008). Hermit: A highly-efficient owl reasoner. En *Owled* (Vol. 432, p. 91).

Tim Berners-Lee. (2006). *Linked data.* https://www.w3.org/DesignIssues/LinkedData.html. (Accessed: 2025-25-02)

Vandenbussche, P.-Y., Umbrich, J., Matteis, L., Hogan, A., Buil-Aranda, y Carlos. (2017). Sparqles: Monitoring public sparql endpoints. *Semantic web*, *8*(6), 1049–1065.

Vollmers, D., Jalota, R., Moussallem, D., Topiwala, H., Ngomo, A. N., y Usbeck, R. (2021). Knowledge graph question answering using graph-pattern isomorphism. *CoRR*, *abs/2103.06752*. Descargado de https://arxiv.org/abs/2103.06752

W3C. (2004). *Owl web ontology language.* https://www.w3.org/TR/owl-features/. (Accessed: 2025-18-01)

W3C. (2012a). *Owl 2 web ontology language.* https://www.w3.org/TR/owl2-overview/. (Accessed: 2025-18-01)

W3C. (2012b). *Owl 2 web ontology language rdf-based semantics (second edition).* https://www.w3.org/TR/owl2-rdf-based-semantics/. (Accessed: 2025-20-02)

W3C. (2013). *Sparql 1.1 overview.* https://www.w3.org/TR/2013/REC-sparql11-overview-20130321/. (Accessed: 2025-12-02)

W3C. (2014a). *Rdf 1.1 semantics.* https://www.w3.org/TR/rdf11-mt/. (Accessed: 2025-20-02)

W3C. (2014b). *Rdf 1.1 turtle.* https://www.w3.org/TR/turtle/. (Accessed: 2025-17-02)

W3C. (2014c). *Rdf schema 1.1.* https://www.w3.org/TR/rdf-schema/. (Accessed: 2025-17-02)

Wang, X., He, X., Cao, Y., Liu, M., y Chua, T.-S. (2019). Kgat: Knowledge graph attention network for recommendation. En *Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining* (pp. 950–958).

Xue, B., y Zou, L. (2022). Knowledge graph quality management: a comprehensive survey. *IEEE Transactions on Knowledge and Data Engineering*, *35*(5), 4969–4988.

Yamamoto, Y., Yamaguchi, A., y Splendiani, A. (2018). Yummydata: providing high-quality open life science data. *Database*, *2018*, bay022.

Zha, D., Bhat, Z. P., Lai, K.-H., Yang, F., Jiang, Z., Zhong, S., y Hu, X. (2025). Data-centric artificial intelligence: A survey. *ACM Computing Surveys*, *57*(5), 1–42.

# Anexo A

# Métricas consideradas de Evaluating the Quality of the LOD Cloud: An Empirical Investigation

### Keeping URIs short

Es una métrica basada en el largo de las URIs del dataset, evaluando la calidad del grafo en base a este concepto.

En el trabajo no se presenta una consulta para definir la métrica, sino que se define una ecuación. La misma es la siguiente:

$$RC1(D) = \frac{\text{size}\left(\{u \in \mathcal{U} \mid (\text{len}(u) \leqslant 80) \wedge (\prime\prime \notin u)\}\right)}{\text{size}(\text{dlc}(D) \cap \mathcal{U})}$$

donde $u$ es un URI que tiene una longitud (definida por la función len) de 80 o menos y no está parametrizado (el URI no contiene '?'). Por lo tanto, este valor métrico mide la proporción de URIs cortos en un conjunto de datos.

| | |
|---|---|
| Dimensión | Facilidad de comprensión |
| Factor | Facilidad de comprensión |
| Granularidad | Grafo |
| Variables | No necesita |

Tabla A.1: Keeping URIs short

**Motivo de descarte:** su cálculo requiere recorrer y filtrar todas las URIs del esquema, lo que puede ser costoso en grafos grandes. Aunque es una métrica

viable en términos de implementación SPARQL, su enfoque está en la estructura de los recursos para las consultas y no en la calidad de los datos, lo que la hace menos relevante para los objetivos de la aplicación. Por esta razón, se descarta su inclusión.

### Minimal usage of RDF data structures

Esta métrica evalúa los aspectos relacionados con la complejidad sintáctica y semántica del uso de las características de las estructuras de datos RDF, como la reificación, los contenedores y las colecciones, las cuales se desaconseja utilizar debido a dicha complejidad.

| | |
|---|---|
| Dimensión | Concisión |
| Factor | Concisión representacional intencional |
| Granularidad | Grafo |
| Variables | No necesita |

Tabla A.2: Minimal usage of RDF data structures

**Motivo de descarte:** dado que la aplicación se enfoca en métricas evaluables mediante consultas SPARQL, esta métrica no resulta adecuada, ya que su cálculo requiere un análisis estructural más allá de lo que puede obtenerse únicamente a partir de consultas sobre los datos.

### Re-use of existing terms

Esta métrica evalúa si un conjunto de datos reutiliza términos relevantes de un vocabulario específico de un dominio. En particular, se examina si una propiedad o clase (en el caso de que el predicado sea $rdf : type$) utilizada en una tripla hace referencia a un término de otro vocabulario. La identificación de vocabularios relevantes para distintos dominios se puede realizar de manera manual o automática. Por ejemplo, mediante el uso de servicios como el portal Linked Open Vocabulary ($LOV$). Además, se considera la superposición de vocabularios recomendados, como $SKOS$, $FOAF$, $DCMI\ Terms$ y otros, que se han convertido en estándares de facto en más del 15 % de los conjuntos de datos $LOD$. El objetivo es aumentar la interoperabilidad entre aplicaciones, facilitando el procesamiento de datos vinculados y mejorando la coherencia entre dominios.

| | |
|---|---|
| Dimensión | Facilidad de comprensión |
| Factor | Facilidad de comprensión |
| Granularidad | Grafo |
| Variables | No necesita |

Tabla A.3: Re-use of existing terms

**Motivo de descarte:** la métrica requiere identificar vocabularios de dominio y determinar su relevancia en distintos contextos, lo cual introduce un nivel de análisis que va más allá de lo que puede resolverse eficientemente mediante consultas SPARQL. Además, la identificación de vocabularios recomendados y su superposición entre dominios requiere mecanismos adicionales, como servicios externos o procesamiento manual, lo que la hace poco práctica para su integración en la aplicación.

### Usage of undefined classes and properties

Esta métrica evalúa la calidad de la definición de clases y propiedades en un conjunto de datos, identificando aquellos casos donde no se sigue una lógica formal adecuada. El uso incorrecto o ambiguo de clases y propiedades puede generar problemas de interpretación y dificultar el razonamiento automatizado, ya que los razonadores pueden enfrentar dificultades al tratar de inferir relaciones y restricciones entre los elementos del conjunto de datos.

| | |
|---|---|
| Dimensión | Facilidad de comprensión |
| Factor | Facilidad de comprensión |
| Granularidad | Grafo |
| Variables | No necesita |

Tabla A.4: Usage of undefined classes and properties

**Motivo de descarte:** No se incluirá la métrica debido a que su cálculo implica evaluar la correcta definición de clases y propiedades desde una perspectiva lógica y formal. Esta tarea requiere validar la existencia y correcta especificación de las clases y propiedades en un esquema externo o una ontología de referencia, lo cual no es directamente medible mediante consultas SPARQL. Además, la métrica introduce una dependencia en la verificación de definiciones fuera del conjunto de datos analizado, lo que complica su implementación dentro del alcance de la aplicación.

### Usage of blank nodes

Esta métrica mide el impacto de los nodos en blanco (blank nodes) en un conjunto de datos de Linked Data, identificando los casos en los que su uso puede ser problemático. Los nodos en blanco son indeseables porque no pueden ser referenciados externamente, lo que contradice las mejores prácticas de Linked Data relacionadas con la interconexión y reutilización. En términos simples, el alcance de los nodos en blanco está "limitado al documento en el que aparecen". Su presencia puede generar inconvenientes durante el consumo de Linked Data y en tareas específicas como determinar si dos gráficos RDF son isomorfos.

| | |
|---|---|
| Dimensión | Facilidad de comprensión |
| Factor | Facilidad de comprensión |
| Granularidad | Grafo |
| Variables | No necesita |

Tabla A.5: Usage of blank nodes

**Motivo de descarte:** su cálculo requiere identificar nodos en blanco dentro del conjunto de datos, lo cual no es directamente medible mediante consultas SPARQL estándar sin asumir conocimiento previo sobre la estructura del grafo. Además, aunque el uso de nodos en blanco puede afectar la interoperabilidad y reutilización de los datos, su impacto varía según el contexto y las necesidades específicas de cada aplicación.

## Different serialisation formats

Esta métrica evalúa la disponibilidad de un conjunto de datos en múltiples formatos de serialización compatibles con el modelo de datos RDF, como RDF/XML, Turtle, N-Triples, N-Quads, JSON-LD, entre otros. Cada formato presenta características con ventajas y desventajas específicas que pueden afectar su uso. Por ejemplo, aplicaciones web suelen preferir el formato JSON-LD debido a su compatibilidad nativa con entornos JavaScript, lo que elimina la necesidad de parsers adicionales. Garantizar la disponibilidad del conjunto de datos en varios formatos facilita su consumo y mejora su versatilidad, promoviendo un uso más amplio y adaptado a diferentes necesidades y plataformas.

**Motivo de descarte:** evalúa la disponibilidad del conjunto de datos en distintos formatos de serialización, un aspecto más relacionado con la accesibilidad y distribución de los datos que con su calidad intrínseca. Además, la relevancia de esta métrica depende del contexto de uso, y su medición no puede realizarse exclusivamente mediante consultas SPARQL sobre los datos, sino que requiere información externa sobre los formatos en los que se ofrece el dataset.

## Usage of multiple languages

Esta métrica evalúa la capacidad de un conjunto de datos para atender a usuarios de diferentes lenguas, asegurando así un alcance global más amplio. Por ejemplo, un conjunto de datos con literales únicamente etiquetados en inglés no será adecuado para usuarios que hablen español. En cambio, si los literales están disponibles en inglés y español, es más probable que el conjunto de datos sea reutilizado. Los literales textuales pueden combinarse con etiquetas de idioma (por ejemplo, @es), lo que facilita su localización y comprensión. Además, las mejores prácticas de datos en la web sugieren que los parámetros de idioma deben explicitarse en los metadatos, permitiendo a los usuarios evaluar cómo trabajar con los datos y habilitar servicios de traducción automatizados. El

uso de etiquetas de idioma mejora la expresión de información lingüística, favoreciendo una mejor localización y aumentando la versatilidad del conjunto de datos.

La ecuación calcula la proporción de triplas RDF en un conjunto de datos que contienen etiquetas de idioma en sus literales textuales. Primero, se define el conjunto de sujetos que tienen al menos un objeto con una forma léxica etiquetada con un idioma. Luego, se considera el subconjunto de esas tripletas donde los literales efectivamente incluyen una etiqueta de idioma. Finalmente, la métrica se obtiene dividiendo el tamaño de este subconjunto entre el tamaño total de sujetos con literales textuales y redondeando el resultado al entero más cercano. Esto permite evaluar en qué medida el conjunto de datos soporta múltiples idiomas mediante el uso adecuado de etiquetas de idioma en los literales.

**Motivo de descarte:** evalúa la disponibilidad de datos en múltiples idiomas, un aspecto más relacionado con la accesibilidad y usabilidad que con la calidad estructural o semántica del grafo. Aunque la presencia de etiquetas de idioma en los literales puede verificarse mediante consultas SPARQL, la métrica no captura directamente problemas de calidad en los datos, sino más bien su adaptabilidad a distintos públicos. Además, su relevancia depende del dominio y del propósito del conjunto de datos, por lo que no resulta esencial dentro del enfoque de evaluación definido para la aplicación.

## Provision of basic provenance information

Esta métrica evalúa la presencia de información sobre la procedencia de los datos en un conjunto de datos de Linked Data. La procedencia es un aspecto clave, especialmente cuando los datos son compartidos entre colaboradores que no tienen contacto directo o cuando los conjuntos de datos permanecen disponibles más allá del ciclo de vida de los proyectos u organizaciones que los generaron. Conocer el origen y el productor de los datos permite a los consumidores evaluar su credibilidad e integridad, facilitando la confianza en su uso y reutilización.

**Motivo de descarte:** se enfoca en la calidad de los metadatos y no en la calidad de los datos en sí. Aunque la información de procedencia es valiosa para evaluar la credibilidad y trazabilidad de un conjunto de datos, su presencia o ausencia no afecta directamente la corrección, coherencia o completitud de los datos contenidos en el grafo.

## Traceability of the data

Esta métrica evalúa la incorporación de metadatos de procedencia en un conjunto de datos, permitiendo a los consumidores rastrear el origen y la historia de cada recurso. En entornos donde múltiples editores contribuyen dentro del mismo espacio de nombres, es fundamental identificar la fuente de cada pieza de información. La ontología PROV-O proporciona un marco estándar para describir esta procedencia, definiendo agentes responsables, entidades representadas

y actividades relacionadas con la generación o modificación de los datos. Incluir esta información mejora la transparencia, facilita la confianza en los datos y permite una mejor gestión de su uso y reutilización.

**Motivo de descarte:** si bien la trazabilidad es un aspecto relevante para la gestión y confianza en los datos, no existe un estándar universalmente adoptado para representarla en los grafos RDF. La métrica depende de la presencia y estructura de metadatos específicos, como los definidos en PROV-O, cuya adopción varía entre conjuntos de datos. Esta falta de estandarización dificulta su evaluación de manera generalizada mediante consultas SPARQL.

## Human readable labelling and comments

Esta métrica evalúa la presencia de etiquetas y comentarios en los datos enlazados para garantizar su comprensibilidad tanto por humanos como por aplicaciones. Dado que las aplicaciones de Linked Data aún no son lo suficientemente inteligentes para inferir el significado de los recursos, es fundamental proporcionar etiquetas y descripciones explícitas. Estos elementos no solo mejoran la interpretabilidad de los datos, sino que también facilitan su uso en búsquedas basadas en palabras clave o en lenguaje natural. Se recomienda el uso de predicados como $rdfs:label$, $foaf:name$, $skos:prefLabel$ y $dcterms:title$ para etiquetar recursos, mientras que $dcterms:description$ y $rdfs:comment$ son adecuados para proporcionar descripciones textuales. La inclusión de estas etiquetas mejora la accesibilidad y usabilidad del conjunto de datos, promoviendo una mejor comprensión dentro del ecosistema de Linked Data.

**Motivo de descarte:** se enfoca en la documentación y comprensibilidad de los datos más que en su calidad intrínseca. Si bien la presencia de etiquetas y comentarios facilita la interpretación y reutilización de los datos, su ausencia no necesariamente implica problemas en la exactitud, coherencia o completitud del grafo. Además, la selección de predicados para etiquetas y descripciones varía según el modelo de datos utilizado, lo que dificulta una evaluación estandarizada mediante consultas SPARQL.

## Regular expression definition of a URI

Esta métrica evalúa la presencia de metadatos estructurales que describan el formato de las URI en un conjunto de datos, facilitando su interpretación y consulta. Definir una expresión regular para la estructura de las URI permite a los agentes comprender mejor los recursos, extrayendo información relevante como el nombre local o filtrando datos según patrones predefinidos. Esto no solo mejora la exploración del conjunto de datos, sino que también optimiza la capacidad de los consumidores para formular consultas más eficientes.

En el trabajo no se presenta una consulta para definir la métrica, sino que se define una ecuación. La misma es la siguiente:

$$U3(D) = \begin{cases} 1 & \text{if has pattern} \\ 0 & \text{otherwise} \end{cases}$$

**Motivo de descarte:** evalúa aspectos estructurales de las URI que no están directamente relacionados con la calidad de los datos en el grafo. La definición de una expresión regular para las URI puede ser útil para la exploración y optimización de consultas, pero no afecta la integridad, coherencia o relevancia de los datos en sí. Además, este tipo de evaluación requiere conocimiento adicional sobre los patrones de las URI, lo cual no puede ser directamente medido mediante consultas SPARQL sobre los datos.

## Indication of used vocabularies

Esta métrica evalúa la especificación de los vocabularios utilizados en un conjunto de datos, lo que facilita la comprensión y consulta de su estructura. Dado que los vocabularios describen los recursos dentro del dataset, su declaración explícita forma parte de los metadatos estructurales y permite a los consumidores, tanto humanos como máquinas, interpretar y formular consultas de manera más eficiente. Al proporcionar esta información, se mejora la accesibilidad del conjunto de datos y se optimiza su integración dentro del ecosistema de Linked Data.

**Motivo de descarte:** evalúa aspectos relacionados con los metadatos y la estructura del grafo, más que con la calidad intrínseca de los datos. Aunque la especificación de los vocabularios utilizados puede facilitar la interpretación y consulta de los datos, no es un indicador directo de la calidad de los mismos. Además, la evaluación de esta métrica requiere conocimiento adicional sobre los vocabularios utilizados, lo cual no puede ser medido directamente mediante consultas SPARQL sobre los datos.

## Extensional conciseness

Esta métrica evalúa la concisión de un conjunto de datos, entendida como la ausencia de redundancia en las instancias locales. En el contexto de Linked Data, la redundancia se refiere a la existencia de dos instancias con identificadores diferentes pero con el mismo conjunto de propiedades y valores de datos correspondientes. La redundancia puede aumentar el tamaño de un conjunto de datos y complicar su curación, ya que se deben actualizar correctamente las instancias replicadas. Sin embargo, en algunos casos, la redundancia puede ser útil, especialmente para mejorar la reescritura de consultas en el acceso a datos basado en ontologías. Esta métrica, por lo tanto, promueve la eficiencia del dataset al eliminar duplicados innecesarios y facilitar su integración en el ecosistema de Linked Data.

**Motivo de descarte:** depende de la identificación externa de duplicados

en el conjunto de datos, lo cual no puede ser realizado de manera sencilla con consultas SPARQL. La definición de redundancia y la necesidad de usar aproximaciones para determinar si dos recursos son idénticos se escapan del alcance de la evaluación mediante consultas sobre los datos. Además, la implementación de esta métrica requiere un proceso más complejo que involucra comparar instancias y aplicar técnicas de hashing, lo que la hace poco viable para el modelo de evaluación de calidad de datos que estamos utilizando.

### Entities as members of disjoint classes

Esta métrica evalúa el uso adecuado de restricciones de disjointness dentro de un conjunto de datos basado en OWL (Web Ontology Language), que extiende la expresividad de RDFS al modelar primitivas difíciles de expresar en este último. El uso correcto de características como $owl : disjointWith$ garantiza que un individuo que pertenece a una clase no pueda ser simultáneamente un miembro de otra clase especificada, lo que mejora la consistencia del modelo. Un ejemplo de clases disjuntas es el de $foaf : Person$ y $foaf : Document$, que en el vocabulario FOAF se definen como disjuntas, asegurando que un recurso no pueda ser tanto una persona como un documento.

**Motivo de descarte:** aunque la detección de violaciones a las restricciones de disjointness puede contribuir a evaluar la consistencia del modelo de datos, esta métrica no mide directamente la calidad intrínseca de los datos, sino más bien la corrección del esquema ontológico utilizado. Además, la evaluación de esta métrica requiere un razonador OWL para inferir y verificar las relaciones disjuntas, lo que escapa del alcance de un modelo de evaluación basado exclusivamente en consultas SPARQL.

### Misplaced classes or properties

Esta métrica evalúa la consistencia en el uso de clases y propiedades dentro de un conjunto de datos RDF, asegurando que las propiedades estén correctamente definidas y aplicadas en función de las clases de recursos a las que corresponden. RDF Schema permite describir las propiedades en términos de las clases a las que se aplican, lo cual es esencial para una correcta interpretación de los datos. La métrica verifica que las triplas sigan la estructura estándar (sujeto, predicado, objeto), asegurando que el predicado sea una propiedad que describa el recurso en la posición del sujeto, y el objeto sea el valor correspondiente. Sin embargo, se excluyen de la evaluación aquellos triplas que involucren axiomas OWL como $owl : equivalentProperty$ o $owl : inverseOf$, ya que requieren que la propiedad se ubique en la posición del objeto. El uso adecuado de estas estructuras previene posibles problemas de interpretación y garantiza la consistencia del modelo.

**Motivo de descarte:** esta métrica evalúa la coherencia en el uso de clases y propiedades dentro del dataset, lo que está más relacionado con la corrección

del esquema ontológico que con la calidad intrínseca de los datos. Además, la métrica se basa en una ecuación matemática en lugar de una consulta SPARQL, lo que sugiere que su implementación requiere un procesamiento adicional fuera del alcance de una evaluación puramente basada en consultas. La detección de clases o propiedades mal ubicadas también puede depender de la interpretación de los espacios de nombres y de reglas específicas del dominio, lo que introduce una complejidad adicional y dificulta su aplicación en un modelo genérico de evaluación de calidad de datos.

### Misused OWL datatype or object properties

Esta métrica evalúa el uso adecuado de las propiedades en OWL, diferenciando entre las que se refieren a individuos `owl:ObjectProperty` y las que se refieren a valores de datos `owl:DatatypeProperty`. Un uso incorrecto de estas propiedades podría generar un funcionamiento inadecuado de los agentes, como cuando un usuario de Linked Data utiliza características de `owl:ObjectProperty` y `owl:DatatypeProperty` para crear hipervínculos entre objetos, lo que puede causar inconsistencias en la interpretación de los datos.

**Motivo de descarte:** esta métrica evalúa la correcta aplicación de propiedades en OWL, lo que está más relacionado con la coherencia del modelo ontológico que con la calidad intrínseca de los datos. Además, su cálculo se basa en una ecuación matemática en lugar de una consulta SPARQL, lo que sugiere la necesidad de un procesamiento adicional fuera del alcance de una evaluación basada exclusivamente en consultas. La detección de un mal uso de propiedades OWL requiere interpretar correctamente los tipos de datos y objetos, lo que introduce una complejidad adicional y puede depender de definiciones específicas del vocabulario utilizado, dificultando su aplicación en un modelo genérico de evaluación de calidad de datos.

### Usage of deprecated classes or properties

Esta métrica evalúa el uso de clases y propiedades obsoletas dentro de un conjunto de datos OWL. Cuando se eliminan clases y propiedades de los esquemas, los datos que las utilizan se vuelven incoherentes. OWL introduce las clases `owl:DeprecatedClass` y `owl:DeprecatedProperty` para identificar estas situaciones. Cualquier clase o propiedad declarada como instancia de estas clases ya no se recomienda para su uso en datos publicados, ya que su uso puede comprometer la coherencia y la integridad del conjunto de datos. Esta métrica ayuda a asegurar que los datos se mantengan actualizados y alineados con las mejores prácticas de publicación en el ecosistema de Linked Data.

**Motivo de descarte:** esta métrica evalúa el uso de clases y propiedades obsoletas, lo que está más relacionado con la evolución del esquema ontológico que con la calidad intrínseca de los datos. Además, su cálculo se basa en una ecuación matemática en lugar de una consulta SPARQL, lo que indica la nece-

sidad de un procesamiento adicional fuera del alcance de una evaluación basada exclusivamente en consultas. La detección de elementos obsoletos depende de la disponibilidad y actualización de los esquemas de referencia, lo que introduce una dependencia externa y puede dificultar su aplicación en un modelo genérico de evaluación de calidad de datos.

### Valid usage of the inverse functional property

Esta métrica evalúa el uso correcto de propiedades inversamente funcionales en un conjunto de datos OWL. En el mundo real, ciertos identificadores, como una clave pública de cifrado, son únicos para cada individuo. Para representar esta unicidad en un documento de Linked Data, debe existir exactamente un recurso que describa dicho identificador. Las propiedades que garantizan esta unicidad se denominan inverse functional properties, lo que significa que si dos recursos diferentes comparten el mismo valor para una de estas propiedades, durante el razonamiento, se interpretan como la misma entidad mediante owl:sameAs. OWL proporciona la clase `owl:InverseFunctionalProperty` para definir estas propiedades. Ejemplos comunes incluyen `foaf:mbox` y `foaf:homepage`.

**Motivo de descarte:** esta métrica evalúa la unicidad de los valores asociados a propiedades inversamente funcionales, lo que depende del razonamiento OWL para inferir equivalencias entre recursos mediante `owl:sameAs`. Sin embargo, el modelo de evaluación basado en consultas SPARQL no incluye razonamiento avanzado, lo que impide una verificación directa de estas inferencias. Además, la detección de violaciones a la unicidad requiere una comparación global de los valores de la propiedad en el dataset, lo que introduce una complejidad computacional significativa y dificulta su aplicación en un modelo eficiente de calidad de datos.

### Ontology hijacking

Esta métrica evalúa la posible presencia de ontology hijacking en un conjunto de datos, una práctica que ocurre cuando una fuente no autorizada redefine o extiende un concepto previamente establecido en una ontología oficial. Un recurso es considerado una fuente autorizada de un concepto si su espacio de nombres coincide con el del concepto en cuestión. Por ejemplo, $http://xmlns.com/foaf/0,1/$ es la fuente oficial de `foaf:Person`. El ontology hijacking puede generar inferencias incorrectas en los datos y afectar la consistencia del modelo. Sin embargo, su restricción también puede ser vista como una limitación al principio de open world assumption de Linked Data, ya que impide que terceros amplíen o modifiquen conceptos existentes.

**Motivo de descarte:** esta métrica evalúa la redefinición de conceptos dentro de una ontología, lo que está más relacionado con la gobernanza y el control del esquema que con la calidad intrínseca de los datos. Además, su cálculo se basa en una ecuación matemática en lugar de una consulta SPARQL, lo que

sugiere la necesidad de un procesamiento adicional fuera del alcance de una evaluación basada exclusivamente en consultas. La detección de ontology hijacking depende de la identificación de fuentes autorizadas y la comparación de espacios de nombres, lo que introduce una dependencia externa y puede ser difícil de evaluar de manera sistemática en un modelo genérico de calidad de datos.

### Usage of incorrect domain or range types

Esta métrica evalúa la coherencia en el uso de dominios y rangos en un conjunto de datos RDF. En un esquema, una propiedad puede definir opcionalmente un dominio y un rango: el dominio indica la clase esperada para el sujeto de una tripla, mientras que el rango define la clase esperada para el objeto (ya sea un recurso o un tipo de dato literal). El uso incorrecto de dominios o rangos puede generar inconsistencias en los datos, dificultando la consulta por parte de los consumidores que dependen de los esquemas subyacentes para formular consultas sin inspeccionar directamente los datos. Garantizar la correcta aplicación de dominios y rangos mejora la precisión de las consultas y la interoperabilidad del conjunto de datos.

**Motivo de descarte:** esta métrica evalúa la coherencia en la aplicación de dominios y rangos en un conjunto de datos RDF, lo que está más relacionado con la validación del esquema ontológico que con la calidad intrínseca de los datos. Además, su cálculo se basa en una ecuación matemática en lugar de una consulta SPARQL, lo que sugiere la necesidad de un procesamiento adicional fuera del alcance de una evaluación basada exclusivamente en consultas. La verificación de dominios y rangos requiere conocer y aplicar reglas definidas en ontologías externas, lo que introduce una dependencia adicional y puede dificultar su aplicación en un modelo genérico de evaluación de calidad de datos.