Framework para determinar el contexto de los datos basado en ontologías de dominio

Santiago Acevedo, Oscar Muñoz

Tutores

Camila Sanz, Flavia Serra

Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Resumen

La información que se obtiene a partir de un conjunto de datos depende fuertemente del contexto en el cual estos son usados. Este proyecto presenta una propuesta para la obtención del contexto de un conjunto de datos. Los datos están determinados por un modelo de datos particular, que representa cierta porción del mundo real, por lo tanto el contexto puede estar basado en ontologías de dominio que representan y amplían la información sobre el dominio de ese modelo de datos. En este trabajo se tendrá en cuenta un modelo de datos relacional, específicamente una base de datos. Para esto se presenta un prototipo de *framework* que, permite la definición del contexto de los datos de una base de datos relacional, haciendo uso de ontologías de dominio.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Hoy en día el rol de los datos es muy importante, entenderlos nos da información valiosa que puede ser utilizada con un sinfín de propósitos, como por ejemplo mejorar el rendimiento deportivo de una persona, optimizar procesos de negocio, predecir comportamientos basados en datos históricos, entre otros. En este marco, también resulta importante poder determinar el contexto de los datos, es decir, la información de fondo que proporciona una comprensión más amplia de un evento, persona o elemento. Además, una vez conocido el contexto, resulta muy útil poder ampliarlo, para tener más conocimiento sobre el dominio de aplicación.

Existen muchas formas de representar el contexto, pero la más utilizada es mediante el uso de ontologías de dominio. Por su naturaleza, las ontologías proporcionan una forma sencilla de representar el conocimiento, además permiten mecanismos que facilitan la obtención de nueva información. Es por esto que se planteó el problema de determinar el contexto de los datos, haciendo uso de dichas ontologías de dominio.

El objetivo principal de este proyecto es entonces, modelar el contexto de una base de datos relacional haciendo uso de ontologías de dominio. A raíz de esto, y debido a que no se tenía una idea clara del estado del arte de los temas involucrados, se estableció como objetivo realizar un relevamiento sobre los temas Contexto y Ontologías. Además se realizó un análisis de las herramientas que se encontraban disponibles hasta el momento con el objetivo de establecer un punto de partida. Este proyecto se realizó en el marco de las tesis de Doctorado de Camila Sanz y Flavia Serra, lo que implicó otra tarea de análisis. Profundizaremos en dichas tesis en la Sección 2.1, referida a los Antecedentes del Proyecto.

Por otro lado, se tiene el objetivo de crear un prototipo de herramienta que cubra las necesidades planteadas. Finalmente, se considera el objetivo de presentar un caso de estudio que sirva de pun-

to de partida para poder modelar una realidad y sobre la cual sea posible aplicar la herramienta creada.

Para lograr la finalidad principal anteriormente descrita, se construye un *framework* que, dada una base de datos y una o varias ontologías de dominio, no sólo es capaz de determinar y modelar el contexto de los datos, sino que también lo amplía. Previo a la generación del contexto, se requiere que exista una relación entre las ontologías y la base de datos inicial, es por esto que se proponen mapeos para realizar dicha correspondencia. Los mapeos utilizados en este proyecto de grado son propuestos en la Sección de Mapeos 4.2. Para la realización del mapeo se debe contar con un usuario experto en el dominio, que pueda indicar la correspondencia entre los elementos de la base de datos con los de las ontologías. Con el mapeo no sólo se busca relacionar ambos sistemas de información, sino que, adicionalmente serán estos elementos mapeados los que permitirán ampliar el contexto, tanto como el usuario lo desee, y sobre los que cobrará valor el resultado devuelto por el *framework* desarrollado.

Poder determinar y ampliar el contexto de los datos sirve para conectar los puntos de referencia entre los mismos y poder transformar la realidad que se tiene modelada al momento, en una potencialmente más grande y con más valor. Para esto, el *framework* permite al usuario conectar la base de datos a la cual se le quiere extender su contexto, cargar ontologías de entrada y realizar mapeos o correspondencias que serán el punto de partida a la hora de generar el resultado. Adicionalmente, previo a la realización del mapeo, el usuario podrá indicar qué tan amplio es el contexto que desea generar.

El contexto se representará a través de una ontología y la misma podrá ser descargada por el usuario. A su vez, la ontología podrá ser visualizada en forma de grafo. La decisión de representar el contexto mediante ontologías se apoya en la posibilidad de explotar la información de contexto que ofrecen las ontologías de dominio ingresadas por el usuario.

Contribuciones

En este proyecto se realizan las siguientes contribuciones:

- Una propuesta para determinar el contexto de una base de datos relacional a partir de ontologías de dominio.
- Un conjunto de reglas definido para la realización de un mapeo que conecte elementos de una base de datos relacional con elementos de una o varias ontologías.
- Un conjunto de validaciones sobre las reglas mencionadas en el punto anterior.
- Una consulta SPARQL [53] iterativa que navega entre los componentes de una ontología para generar un contexto.

- Un prototipo de aplicación web que implementa la propuesta, incluyendo la visualización de los elementos en forma de grafo.

Organización del documento

El documento esta dividido en ocho capítulos, donde se comienza dando un marco teórico al proyecto en el capítulo 2. Posteriormente, se presenta un caso de estudio que se encuentra en el capítulo 3, el cual será de interés a lo largo del informe, ya que se utiliza como herramienta para explicar distintos conceptos. En el capítulo 4, se especifica cómo se define el contexto y se señala como realizar un mapeo entre bases de datos relacionales y ontologías. Luego, se continúa con el diseño y arquitectura en el capítulo 5 y posterior implementación en el capítulo 6. Por último, en el capítulo 7 se explica el funcionamiento del *framework* y se analiza los resultados obtenidos, para luego terminar con el trabajo a futuro y las conclusiones del proyecto en el capítulo 8.

Capítulo 2

Marco Teórico

En este capítulo se comienza hablando de los antecedentes del proyecto. Luego se presenta el marco teórico, en donde se abordan los principales temas del presente trabajo: ontologías y contexto. También se habla del uso de ontologías para el modelado de contexto y por último se abordan los conceptos relacionados a SPARQL.

2.1. Antecedentes del Proyecto

Este proyecto se encuentra enmarcado en dos tesis de doctorado. La primera es la realizada por Flavia Serra, quien llevó a cabo el articulo *Handling Context in Data Quality Management* [56], el cual se centra en principalmente dos temas: la calidad de datos y contexto. Partiendo del conjunto de tareas y técnicas que engloba el proceso *Data Quality Management*, el objetivo del trabajo es modelar el contexto, explotando la naturaleza contextual de los datos en cada fase de dicho proceso. Por otro lado, la otra tesis de doctorado en la que se enmarca el presente trabajo, es la de Camila Sanz, *Context based Data Quality Rules for Multidimensional Data* [54]. Este trabajo busca definir reglas de calidad de datos que consideren el contexto para sistemas de *data warehouse*. Se presenta el uso de ontologías de dominio para representar el contexto de los datos.

Se busca entonces que el *framework* desarrollado en el presente trabajo, partiendo de los temas desarrollados en los trabajos anteriormente mencionados, brinde una solución a una parte de los mismos. Para el caso de la tesis de Flavia Serra, el contexto generado por el *framweork* de este proyecto, puede ser utilizado como *input* en su proyecto para funcionar como herramienta para modelar el contexto en el ámbito de *Data Quality Management*. Por otro lado, para el caso de Camila Sanz, su contexto está modelado mediante el uso de ontologías, por lo que se puede utilizar este *framweork* para generar un contexto deseado, expresado en forma de ontología, y ser utilizado

directamente en el marco de su tesis.

2.2. Ontologías

Dado que en la Web es posible encontrar varias definiciones que contienen una amplia variedad de conceptos y matices, se buscó una definición de Ontología apropiada para este proyecto. En primer lugar la Real Academia Española [34] define una ontología como una parte de la metafísica que trata del ser en general y de sus propiedades trascendentales. Si bien esta definición es acertada, escapa de muchas aristas que ayudan a entender el concepto y su aplicación en el marco de este trabajo. En el año 2009, Nicola Guarino et al. [39], propusieron una definición de ontología que establece que si $\mathtt{C}$ es una conceptualización, y $\mathtt{L}$ un lenguaje lógico con el vocabulario $\mathtt{V}$ y un compromiso ontológico $\mathtt{K}$[1], entonces se tiene que una ontología $\mathtt{O_K}$ para $\mathtt{C}$ con el vocabulario $\mathtt{V}$ y el compromiso $\mathtt{K}$ es una teoría lógica consistente en un conjunto de fórmulas, diseñada para que el conjunto de sus modelos se aproxime tan bien como sea posible al conjunto de modelos previstos de $\mathtt{L}$ según $\mathtt{K}$. Sin embargo, según un análisis encontrado en [48], la definición anteriormente mencionada no funciona por muchos motivos. El primer motivo es que no hay una formalización acerca de qué se entiende por el término "compromiso ontológico", lo que podría generar discrepancias en su entendimiento. Además, se utiliza la frase "se aproxime tan bien como sea posible", sin embargo, no se especifica cuál debería ser la "aproximación"señalada, por lo tanto, debería ser asumida. El artículo aborda otros conceptos, pero dado que escapan al alcance de este proyecto, los mismos no son tenidos en cuenta.

Finalmente, se seleccionó la definición más citada dentro de la literatura relevada. Gruber [37] menciona que una ontología es una especificación formal, que es explícita de una conceptualización compartida. Por conceptualización se entiende una visión abstracta y simplificada de lo que se quiere representar. El hecho que sea una conceptualización compartida significa que es un consenso en la comunidad sobre ciertos objetos, es decir conceptos y entidades que son compartidos por todos. Por último, ser explícita y formal implica que puede ser procesable por una máquina, y que están explícitamente definidos sus conceptos y restricciones.

Una vez especificada la definición de ontologías, podemos ver que las mismas pueden clasificarse según diferentes tipos de criterios. Entre ellos se encuentra la clasificación de acuerdo al nivel de generalidad o de acuerdo al tipo de estructura de conceptualización, entre otros [30]. En esta primera, según Guarino [38], se establece que se cuenta con 4 tipos de ontologías:

- **Ontologías de Alto Nivel**: Describen conceptos generales como espacio, tiempo, materia, objeto. No están ligadas a un dominio o problema particular. Su intención es unificar criterios

[1] Un compromiso ontológico es el resultado de vincular los términos de un lenguaje a los conceptos de una conceptualización de un dominio dado [39].

entre grandes comunidades de usuarios.

- **Ontologías de Dominio**: Describen el vocabulario relacionado a un dominio genérico por medio de la especialización de los conceptos introducidos en las ontologías de alto nivel.
- **Ontologías de Tareas**: Describen el vocabulario relacionado a una tarea o actividad genérica por medio de la especialización de los conceptos introducidos en las ontologías de alto nivel.
- **Ontologías de Aplicación**: Describen conceptos que pertenecen tanto a un dominio como a una tarea particular, por medio de la especialización de los conceptos de las ontologías de dominio y de tareas. Generalmente corresponden a roles que juegan las entidades del dominio cuando ejecutan una actividad.

Para este trabajo es de especial interés la clasificación según nivel de generalidad, ya que el objetivo principal del proyecto es poder determinar el contexto de los datos haciendo uso de ontologías de dominio. Por esta razón, en primer lugar debemos definir qué se entiende por ontología de dominio. Como se mencionó anteriormente, una ontología de dominio se define como una ontología que define el vocabulario relacionado a un dominio genérico. El ejemplo más conocido se encuentra en el área de medicina, donde se cuenta con la ontología SNOMED sobre la cual se puede encontrar un análisis en [42].

Hoy en día existen diversos lenguajes para trabajar con ontologías, los que han sido propuestos mayormente para describir recursos de la web. En este informe se hará foco en los principales, siendo estos RDF [35], RDFS [21] y OWL [13]. Primero se considera el lenguaje RDF, un lenguaje para expresar afirmaciones mediante el uso de triplas. Los otros dos lenguajes permiten restringir las afirmaciones de RDF en dominios de aplicación particulares, siendo RDFS bastante simple, mientras que OWL agrega un vocabulario más extenso, lo que permite un mayor detalle a la hora de describir recursos.

RDF (del inglés *Resource Description Framework*) proporciona un lenguaje simple para describir anotaciones sobre recursos web identificados por una URI (del inglés *Uniform Resource Identifier*), siendo esta una cadena de caracteres que contiene un nombre o una dirección que identifica a un objeto en la web. Estas anotaciones son afirmaciones. Por recurso se entiende cualquier cosa a la que se pueda hacer referencia, pudiendo ser una página web, un servicio web, un identificador de una entidad, un concepto, una propiedad, etc. Las restricciones sobre estas afirmaciones en dominios particulares se establecen en RDFS o OWL. Una afirmación RDF consta de una tripla. Una tripla está formada por un sujeto, un predicado y un objeto, y la misma expresa una relación denotada por el predicado entre el sujeto y el objeto. RDF brinda una representación sencilla, enfocada en

las instancias y en el mapeo a sus tipos. Es posible definir propiedades personalizadas para vincular datos y crear nuevas triplas. En la sección 2.5 se profundizará sobre la sintaxis de RDF.

Existen algunas situaciones donde RDF no es suficiente para poder modelarlas. En ocasiones, es interesante representar relaciones más complejas como por ejemplo, subclases (el tipo de un tipo). Son en estos casos en donde RDFS proporciona medios especiales para representar tales situaciones. Se considera RDFS como el lenguaje de esquema para RDF. Con las herramientas que brinda, permite especificar una serie de restricciones útiles sobre los elementos y las relaciones utilizadas en las triplas RDF. Con RDFS también es posible relacionar semánticamente elementos del conjunto de datos.

Finalmente, OWL (del inglés *Web Ontology Language*) amplía RDFS con la posibilidad de expresar restricciones adicionales. OWL sirve para construir ontologías o esquemas sobre conjuntos de datos RDF. Puede ser utilizado para presentar el significado de ciertos términos y las relaciones entre ellos. Su objetivo es facilitar un modelo sobre RDF que permita representar ontologías a partir de un vocabulario más amplio y una sintaxis más fuerte que la que permite RDF y RDFS.

2.3. Contexto

Hace años se estudia la noción de contexto en varias áreas de investigación [45, 58, 43]. Una definición de este término es la que se encuentra en *Oxford Dictionaries* [50], y la misma menciona que el contexto son las circunstancias que forman el escenario de un evento, declaración o idea, en términos en los que puede entenderse y evaluarse por completo.

En el año 1994, Schilit y Theimer [55] introdujeron el concepto contexto en el área de la computación, y lo definieron como la ubicación, la identidad de la gente y de los objetos cercanos, así como los cambios sobre estos objetos. A partir de este momento, existen varios tipos de definiciones contextuales en la literatura científica como también más específicamente en lo que respecta a la computación [55, 32, 29, 33]. La mayoría se enfocan en la información ambiental de un usuario [55, 32, 29]. Sin embargo, existe una definición particular que hace hincapié en el entorno dinámico. Dicha definición señala que el contexto no es simplemente el estado de un entorno predefinido con un conjunto fijo de recursos de interacción, sino que es parte de un proceso de interacción con un entorno en constante cambio compuesto de recursos reconfigurables, migratorios, distribuidos y de múltiples escalas [33].

Dado que el contexto es un concepto amplio, varias entidades pueden formar parte de él, por lo que se han propuesto diferentes formas de clasificarlo. El siguiente listado especifica algunas de las categorías planteadas por Poslad [51].

- **Contexto geográfico**: indica la ubicación del usuario, la que es obtenida mediante la ubicación del dispositivo utilizado por el usuario.
- **Contexto físico**: en este tipo de contexto se consideran las propiedades físicas de los usuarios, objetos, dispositivos, recursos y servicios que rodean la aplicación, como por ejemplo la presión arterial, temperatura, nivel de ruido, etc.
- **Contexto temporal**: cualquier tipo de información temporal, comprendiendo hora del día, fecha, estación del año, etc.

El concepto que engloba todo lo anteriormente especificado es el de Computación Ubicua, el cual se define como la integración de la computación en el contexto de la persona, de forma omnipresente y no percibida como productos u objetos informáticos diferenciados [36]. Tiene como característica principal la integración de diversas aplicaciones con distintos dispositivos, con el objetivo de beneficiar tareas cotidianas de los usuarios, en una forma transparente para los mismos. Lo mencionado anteriormente se enfoca en el contexto de un usuario, teniendo en cuenta dispositivos y aplicaciones. Otra perspectiva no menos importante, es cuando se habla del contexto enfocado en los datos. Como se menciona en la tesis de Flavia Serra [57], los datos deben ser contrastados con su contexto ya que hay una subjetividad intrínseca que depende del contexto. En adición, el mismo dato desde que es extraído de las fuentes, hasta que es recibido por el usuario, puede recorrer diferentes contextos.

A la hora de trabajar con grandes volúmenes de datos existen dos problemas muy comunes. El primero es la heterogeneidad de la semántica, la cual sucede cuando esquemas de bases de datos de un mismo dominio son desarrollados por diferentes partes, independientes entre sí. Esto provoca diferencias entre significados e interpretaciones. Es sabido que las ontologías han jugado un papel importante en su tarea de describir la semántica de los datos y abordar los problemas de heterogeneidad anteriormente descritos [44]. Por otra parte, el otro problema es el de la contextualización de los datos, ya que la captura del contexto requiere de modelos apropiados para la representación y el modelado del mismo.

Los datos tienen un valor limitado si no son considerados dentro de su contexto [49]. Esto se ve reflejado, por ejemplo, en las empresas que manejan sus datos internamente, sin conectarlos con el mundo exterior, acortando muchas veces la información que se podría obtener al considerar su contexto. Es por esto que abordar datos contextualizados se ha convertido en un gran desafío que aportaría mucho valor en el manejo de la información.

2.4. Ontologías para el Modelado de Contexto

Como se mencionó en las secciones anteriores, una ontología presenta una forma explícita y formal de definir una conceptualización. Esta es una ventaja que hace que las ontologías de dominio sean beneficiosas a la hora de modelar contexto. El hecho de soportar interoperabilidad y heterogeneidad (situaciones muy comunes cuando se trabaja con distintas fuentes de datos), hace posible la representación de relaciones complejas y dependencias entre los datos del contexto.

Hoy en día existe un común acuerdo con respecto a que la representación de un contexto debe abordar una comprensión compartida de algunos dominios, los cuales están formados por una colección de conceptos, objetos, propiedades, restricciones, etc. Es claro que este tipo de representaciones pueden tomar valor en distintos tipos de sistemas únicamente cuando las mismas comparten un formalismo común, que pueda ser utilizado por todos. El modelado de contexto requiere la manipulación de una variedad de tipos de información y sus relaciones. Este proceso también establece una fuerte conexión con el modelado conceptual, en el que las características principales son la reutilización y el consenso sobre las estructuras del modelado. De lo anterior se desprende que un marco ontológico parece ser un componente importante que puede mejorar la capacidad expresiva del modelo del contexto.

Según [31] se deben cumplir un conjunto de requerimientos para poder tener un modelo de contexto eficiente:

- **Generalidad del modelo**: El modelo tiene que ser suficientemente genérico para ser usado en diferentes dominios.
- **Modelado a nivel conceptual**: Debe abstraerse de las implementaciones.
- **Parámetros internos**: Considerar parámetros de contexto internos, que se refieren a atributos almacenados en la base de datos, como también parámetros de contexto externos, que involucran atributos fuera de la base de datos.
- **Semánticamente explícito**: El modelo tiene que dar una solución haciendo explícito el modelo semántico de los datos.
- **Técnica de explotación dedicada**: Debe ofrecer técnicas para explotar el modelo.

En [31] también se comparan diferentes modelos de contexto que carecen de los 5 requerimientos planteados anteriormente y se concluye que la mejor solución es modelar el contexto mediante una ontología.

2.5. SPARQL

En esta sección nos enfocaremos en el lenguaje SPARQL (del inglés *Protocol and RDF Query Language*) [53], para la consulta de grafos RDF.

Junto con el lanzamiento de RDF en 1998 como Recomendación de W3C, se planteó el problema de consultar datos que se encuentren modelados por RDF. Desde entonces, se han propuesto varios diseños e implementaciones de lenguajes de consulta RDF [40]. En 2004, el Grupo de Trabajo de Acceso a Datos RDF publicó un primer borrador de trabajo público de un lenguaje de consulta para RDF, llamado SPARQL. Desde enero de 2008, además de RDF también SPARQL se ha convertido en una Recomendación de W3C.

SPARQL es un lenguaje de consulta de coincidencia de grafos RDF. Dada una fuente de datos `D`, una consulta consta de un patrón que se compara con `D`, y los valores obtenidos de esta coincidencia se procesan para dar la respuesta. A su vez, la fuente de datos `D` que se va a consultar puede estar compuesta por una o varias fuentes.

Una consulta SPARQL consta de tres partes.

- La parte de **coincidencia de patrones**, que incluye varias funcionalidades interesantes de coincidencia de patrones de grafos, como partes opcionales, unión de patrones, anidamiento, filtrado de valores de posibles coincidencias y la posibilidad de elegir la fuente de datos para que coincida con un patrón.
- Los **modificadores de solución**, que una vez calculada la salida del patrón (en forma de tabla de valores de variables), permiten modificar estos valores aplicando operadores clásicos como proyección, distinto, orden, límite y *offset*.
- La **salida** de una consulta SPARQL puede ser de diferentes tipos: consultas binarias, selecciones de valores de las variables que coinciden con los patrones, construcción de nuevas triplas a partir de estos valores y descripciones de recursos.

Para definir la sintaxis de SPARQL, previamente es necesaria la definición formal de RDF para introducir algunos conceptos. Sean `I`, `B` y `L` conjuntos disjuntos infinitos de: IRIs [20], nodos en blanco y literales, respectivamente. Siendo una IRI una secuencia de caracteres del Conjunto de Caracteres Universal, un nodo en blanco un nodo que indica la existencia de un individuo con atributos específicos pero sin proveer un identificador o una referencia y por último un nodo literal un nodo que especifica valores de datos, como no tienen una existencia separada no precisan ser identificados de forma uniforme. Entonces, una tripla RDF es una tupla (s, p, o) de $(\mathtt{I} \cup \mathtt{B}) \times \mathtt{I} \times$

(I ∪ B ∪ L), donde s es el **sujeto**, p **predicado** y o **objeto**. Se tiene entonces que un grafo RDF es un conjunto de triplas RDF. Un ejemplo de grafo RDF es el de la figura 2.1. Como se puede observar, el grafo cuenta con dos triplas RDF. La primera tiene a `dbpedia:The_beatles` como sujeto, `foaf:made` como predicado y `dbpedia:Let_It_be` como objeto. La restante tripla RDF también tiene a `dbpedia:Let_It_be`, pero esta vez como sujeto, `rdfs:label` como predicado y el nodo literal `'Let It Be'` como objeto.

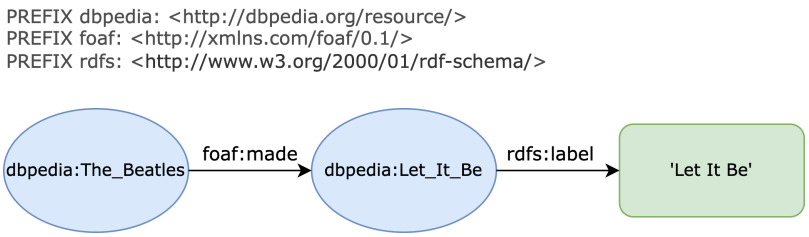

Figura 2.1: Ejemplo grafo RDF

Denotamos por IL la unión I ∪ L, y por T la unión I ∪ B ∪ L. También se asume la existencia de un conjunto infinito V de variables disjuntas de los conjuntos anteriores. La sintaxis oficial de las consultas SPARQL 1.1 [25] considera varios operadores como **OPTIONAL**, **UNION**, **FILTER**, **GRAPH** y la concatenación mediante el símbolo de punto (.) para construir lo que se conoce como **patrones de grafo**. Una expresión de patrón de grafo SPARQL se define de forma recursiva de la siguiente manera:

1. Una tupla de (IL ∪ V) × (I ∪ V) × (IL ∪ V) es un patrón de grafo (un patrón triple).
2. Si P1 y P2 son patrones de grafos, entonces las expresiones (P1 **AND** P2), (P1 **OPT** P2) y (P1 **UNION** P2) son patrones de grafos.
3. Si P es un patrón de grafo y R es una condición de filtro de SPARQL, entonces la expresión (P **FILTER** R) es un patrón de grafo. Donde las condiciones de filtro SPARQL son restricciones de la forma:
   - ?x = u, ?x =?y, isBlank(?c), entre otras, para ?x,?y ∈ V y u ∈ I
   - ¬R, R1 ∧ R2, o R1 ∨ R2, para condiciones de filtro R, R1 y R2.

La mayor fortaleza de este lenguaje de consulta es la de poder navegar por las relaciones de los grafos RDF haciendo uso de la coincidencia de patrones. En este proceso, los patrones simples se pueden combinar en otros más complejos, que exploran relaciones más elaboradas en los datos.

SPARQL tiene cuatro formas de consulta. Estas formas de consulta utilizan las soluciones de la coincidencia de patrones para formar conjuntos de resultados o grafos RDF. Las formas de consulta son:

- `SELECT`: Devuelve todas o un subconjunto de las variables vinculadas en una concordancia con un patrón de búsqueda.

- `CONSTRUCT`: Devuelve un grafo RDF construido mediante la sustitución de variables en un conjunto de plantillas de triplas RDF.

- `ASK`: Devuelve un valor booleano indicando si se encuentra o no una concordancia para un patrón de consulta.

- `DESCRIBE`: Devuelve un grafo RDF que describe los recursos encontrados.

El conjunto de patrones de grafos SPARQL, así como el tipo de consulta `SELECT`, ha atraído mucha atención, principalmente en la comunidad de la Web Semántica [46]. Sin embargo, teniendo en cuenta que el *framework* devolverá al usuario el contexto correspondiente a la base de datos ingresada, no sería útil únicamente coincidir patrones de grafo para obtener resultados, sino que se busca además construir un nuevo grafo con el contexto asociado a los elementos mapeados. Teniendo en cuenta que la consulta `SELECT` sólo devuelve mapeos como resultado y no grafos RDF a los que se le pueda realizar una consulta en cada iteración, se considera otro tipo de consulta, la forma de consulta `CONSTRUCT`.

Una consulta SPARQL `CONSTRUCT`, es una expresión de la forma:

CONSTRUCT H WHERE P,

donde `H` es un conjunto de triplas $(\mathtt{T} \cup \mathtt{V}) \times (\mathtt{I} \cup \mathtt{V}) \times (\mathtt{T} \cup \mathtt{V})$, llamado *template*, y `P` es un patrón de grafo. La forma de consulta `CONSTRUCT` devuelve un único grafo RDF especificado por el *template*. El resultado es un grafo RDF formado al tomar cada solución hallada por el patrón de grafo, sustituyendo las variables en el *template* del grafo, y combinando las triplas en un único grafo RDF mediante la unión de conjuntos. Esto permitirá al *framework* devolver el contexto requerido en forma de grafo RDF.

A modo de resumen, se presenta un ejemplo de consulta SPARQL donde se puede observar algunas aplicaciones de los conceptos antes presentados. Estos conceptos serán ampliados y explicados con más profundidad en la sección 6.3, donde se aborda en detalle la consulta SPARQL utilizada

por el *framework*. Se considera la fuente de datos expresada por el grafo RDF que se visualiza en la figura 2.2, en el cual se observan varias triplas RDF que especifican algunos de los álbumes de la banda The Beatles. Lo que se quiere lograr mediante el uso de una consulta del tipo `CONSTRUCT`, es poder crear un nuevo grafo RDF en donde se haga uso del predicado `dc:creator` y establezca el creador de los álbumes que se muestran en el grafo.

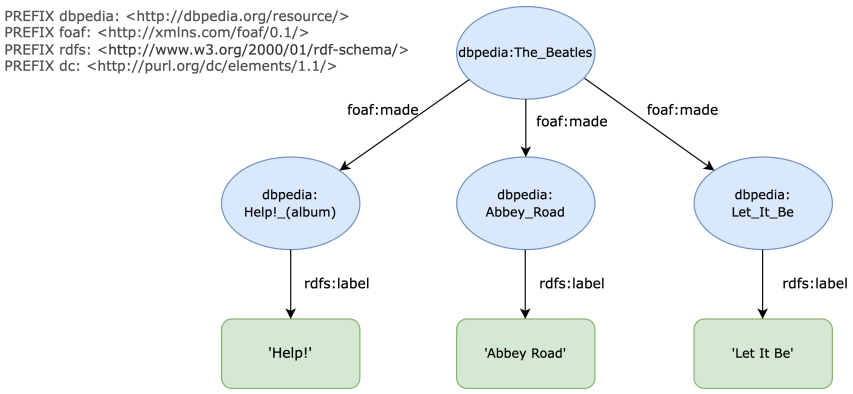

Figura 2.2: Grafo RDF

La consulta, que haciendo uso de la forma de consulta `CONSTRUCT`, crea el grafo RDF es la siguiente:

```
CONSTRUCT {
        ?album dc:creator dbpedia:The_Beatles . }
WHERE {
        dbpedia:The_Beatles foaf:made ?album . }
```

Dado el patrón de grafo establecido en el bloque del `WHERE`, se observa que en la variable `?album` se almacenan todos los álbumes que fueron creados por la banda The Beatles. Luego, en el bloque del `CONSTRUCT` se especifica el *template* que toma la variable `?album`. Por cada uno de estos, se crea la tripla RDF que los toma como sujetos, luego como predicado el elemento `dc:creator` y como objeto `dbpedia:The_Beatles`. Como resultado, se obtiene el grafo RDF que se visualiza en la figura 2.3.

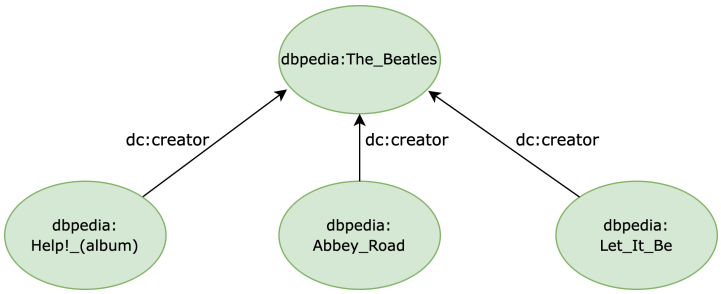

Figura 2.3: Grafo RDF resultado

Capítulo 3

Caso de Estudio

Una vez entendidos y formalizados los conceptos necesarios, se presenta un caso de estudio que se utiliza como guía en el transcurso del informe. Se considera una base de datos relacional que representa una universidad, la cual es presentada en 3.1, y una ontología de dominio que busca darle contexto a la base de datos, la cual es presentada en 3.2.

3.1. Base de Datos

En la figura 3.1 se muestra el esquema de la base de datos. La base de datos busca modelar una realidad sencilla en el contexto de una universidad. Se cuenta con información de estudiantes que pueden estar asociados a varios cursos. Al mismo tiempo, estos cursos pueden estar asociados a varios estudiantes. Cada curso se corresponde a una única asignatura, la cual pertenece a un único instituto. Los institutos pueden tener varios profesores asociados, los cuales pueden pertenecer a varios cursos. Por último se observa que un estudiante pertenece a una carrera y puede contar con una dirección y/o una beca.

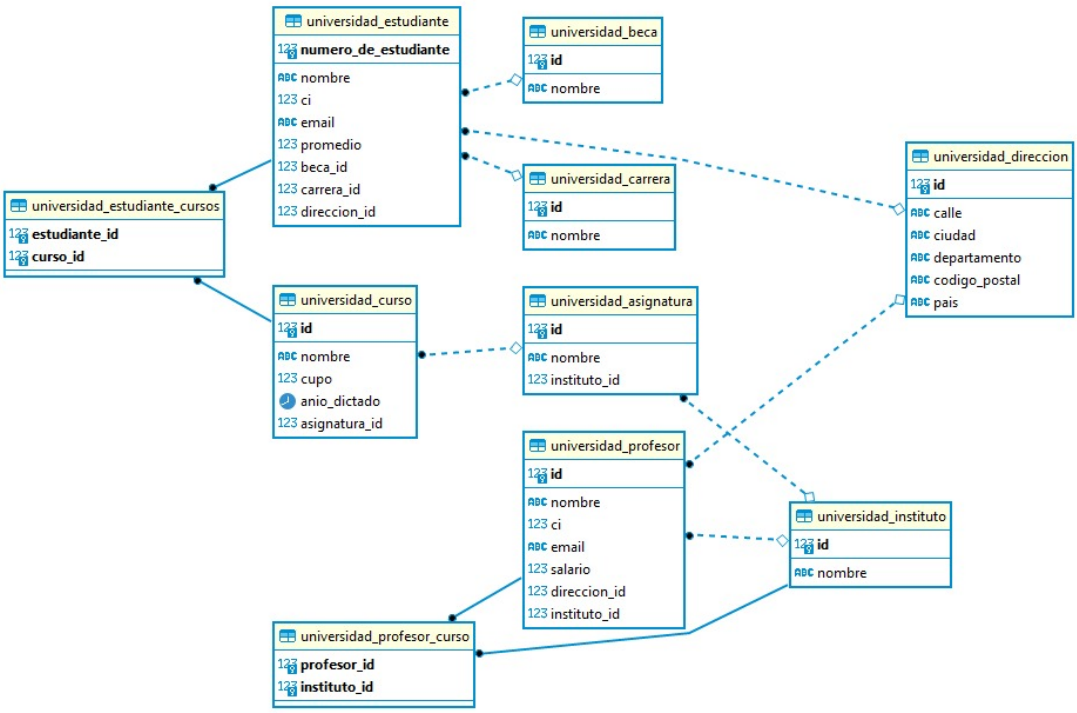

Figura 3.1: Esquema de la base de datos del caso de estudio

El ejemplo busca ser concreto pero intentando contemplar todos los casos que luego aparecerán en la sección 4.3, de forma de poder demostrar todas las reglas en práctica con la aplicación.

3.2. Datos de Contexto

Para el contexto se creó una ontología con algunas similitudes con la base de datos anteriormente presentada, ya que son sobre el mismo dominio, pero introduciendo nuevas relaciones y clases, de tal forma de que el contexto obtenido sea de valor. Con este objetivo, como se observa en la imagen 3.2, se agregan clases como: `TipoCurso`, `Postgrado`, `Grado`, `Biblioteca`, entre otras, *object properties* como: `LibroPertenece`, `InscripcionEstudiante` y *data properties* como: `Carrera_CreditosNecesarios`, `Asignatura_SemestreCursado`, etc. Para la visualización de las ontologías durante todo el flujo se utilizó una representación en forma de grafo, que se pudo obtener de la solución desarrollada.

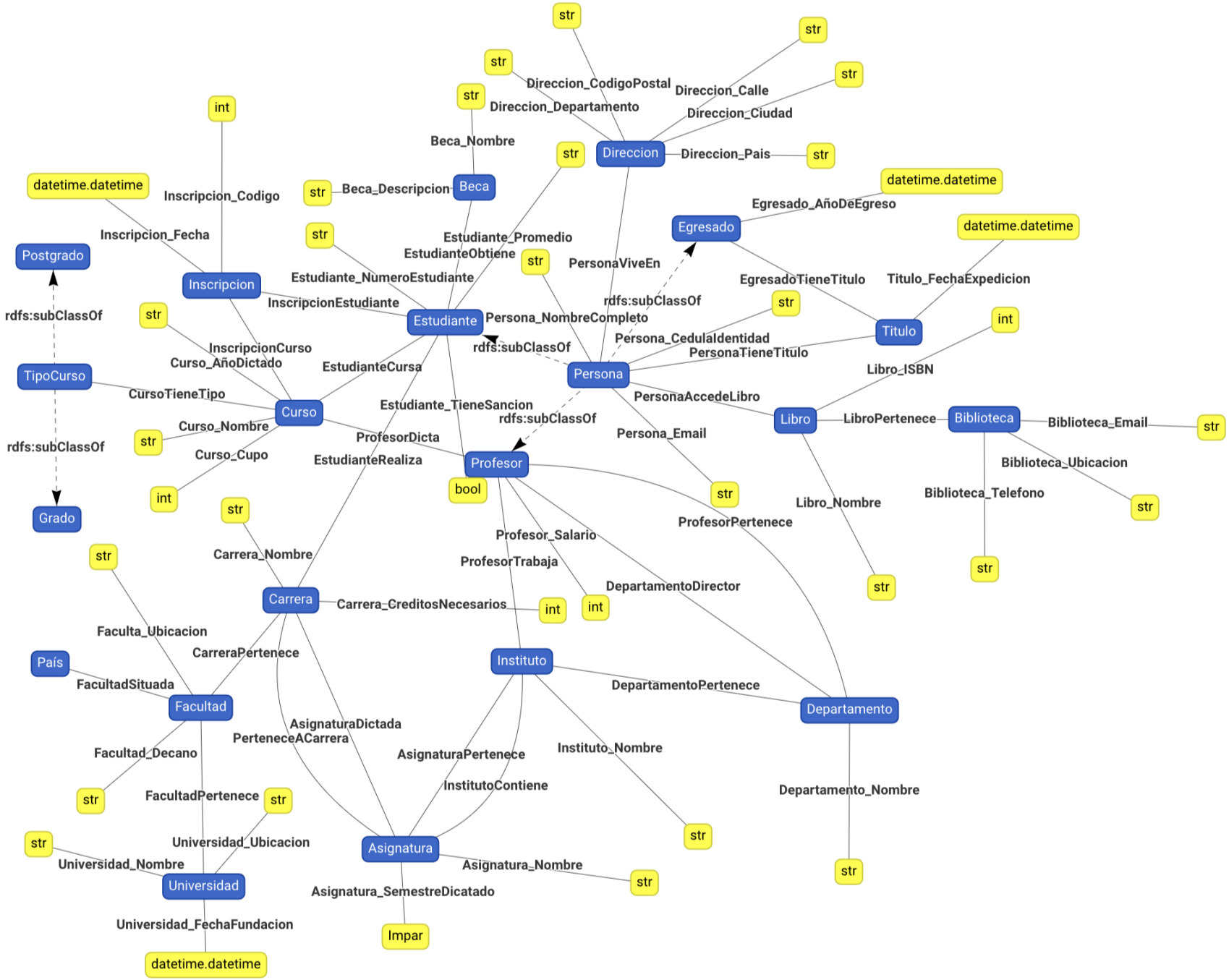

Figura 3.2: Ontología del caso de estudio

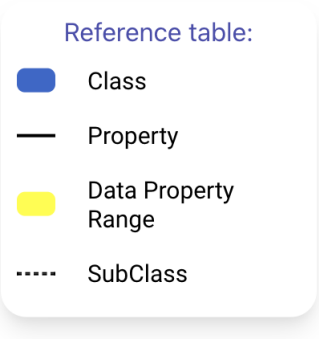

Figura 3.3: Referencias de la ontología

Capítulo 4

Definición del Contexto

En este capítulo se define el mecanismo de mapeo entre una base de datos y una (o varias) ontología(s) de dominio y en 4.3 se presentan las reglas aceptadas por el *framework* para realizar dichos mapeos, ejemplificando cada una con un grafo conteniendo el contexto resultado. El capítulo finaliza con la descripción de la solución propuesta en la sección 4.4.

4.1. Distancia y Mapeos Parciales y Totales

Previo a la definición del mecanismo de mapeo, es importante definir términos tales como la distancia entre dos elementos de una ontología y los mapeos parciales y totales, siendo estas herramientas fundamentales a la hora de definir el contexto. Estos conceptos fueron extraídos de la tesis de doctorado de Camila Sanz [54], y adaptados al marco de este proyecto. Se introduce entonces el concepto de distancia. La distancia entre dos elementos conectados e_1 y e_2 en una ontología, es la longitud de la ruta no dirigida más corta entre ellos. Es de interés también dar una definición de mapeos parciales y totales, siendo estos términos que el *framework* considera a la hora de definir el contexto. Se les llama mapeos totales a los mapeos donde la correspondencia entre el elemento de la base de datos y el de la ontología es total, estos son considerados abstracciones que representan el mismo elemento de la realidad. Por otro lado, de no ser total la correspondencia, entonces se tiene un mapeo parcial. Por ejemplo, si se tuviera un elemento en una base de datos que represente a una persona, la misma se podría mapear totalmente con un elemento que también represente una persona en una ontología, y por otro lado se podría mapear parcialmente a un elemento que represente a un estudiante en la ontología.

4.2. Mecanismo de Mapeo

Como se explicó en el capítulo 2 correspondiente al Marco Teórico, se entiende por contexto a los elementos que están cercanos a un objeto de estudio. En adición, este contexto es dinámico y abarca el proceso de interacción de un elemento con su entorno. En el marco de este proyecto, el objeto de estudio son los distintos elementos de la base de datos para los que el usuario busca determinar su contexto. Como recurso para generar el contexto, se utilizan las ontologías de dominio provistas por el usuario. En este proceso, primeramente es necesario identificar la correspondencia de los elementos de la base de datos con los elementos de las ontologías. Los elementos de la base de datos para los cuales el usuario especifica su correspondiente en las ontologías, son los que finalmente el *framework* tomará como base para generar el contexto. Una vez identificadas las correspondencias entre los elementos, se procede a ampliar la visión sobre los elementos iniciales, siendo estos los que el usuario mapea en las ontologías, y considerar todos los elementos que se encuentren en la cercanía de los mismos. La cercanía depende del tamaño de contexto que el usuario desee. El tamaño busca determinar qué tanto contexto el *framework* debe devolver y cuáles son los elementos que son considerados cercanos y de relevancia para el contexto resultante. Por lo que, si el usuario desea obtener el contexto con tamaño n, ingresará n como entrada y el *framework* devolverá todos los elementos de la ontología que se encuentran a distancia n de los elementos mapeados. Dicho esto, se podría considerar como el contexto de menor tamaño a todos los elementos que se encuentran a distancia 1 de los elementos iniciales de las ontologías, los cuales fueron seleccionados por el usuario en la etapa de mapeo. A mayor distancia, mayor será la cantidad de información devuelta por el *framework*.

Como se explicó en el párrafo anterior, para que el *framework* pueda cumplir con su finalidad principal, es necesario realizar el mapeo que asocie el conjunto de datos, de la base de datos, a los cuáles se les quiere determinar el contexto, con sus correspondientes en las ontologías. Para reflejar este proceso de mapeo y representar la definición del contexto se cuenta con la figura 4.1.

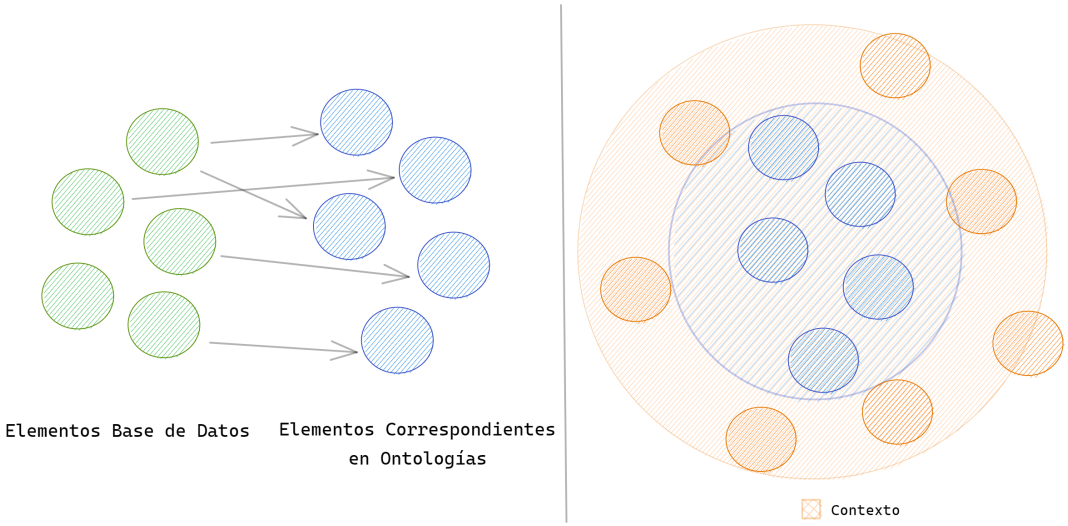

Figura 4.1: Representación gráfica de contexto

Teniendo en cuenta la definición que se presenta en [47], un mapeo es la especificación de un mecanismo para transformar los elementos de un modelo (conforme a un metamodelo particular), en elementos de otro modelo que se ajusta a otro metamodelo (posiblemente el mismo). De acuerdo con los autores, un mapeo puede ser expresado como asociaciones, restricciones, reglas, plantillas con parámetros que deben asignarse durante el mapeo u otras formas aún por determinar. En el caso de este trabajo, el mapeo será entre elementos de una base de datos y los elementos de una o varias ontologías.

El problema que se presenta al realizar el mapeo entre una base de datos y una ontología es considerado un problema de integración de datos [52]. Un sistema de integración de datos es una tripla $\langle G, S, M \rangle$ donde G es el esquema global, S es el esquema fuente y M es un conjunto de afirmaciones que relacionan los elementos del esquema de origen con los elementos del esquema global [52]. El objetivo de estos tipos de sistemas es definir una interfaz común a varias fuentes de datos, permitiendo a los usuarios consumir de la misma, sin preocuparse en los modelos de los distintos sistemas fuentes. Llevando estos términos a nuestro caso, se observa que el esquema fuente es el esquema relacional de la base de datos y el esquema destino, es el esquema de la(s) ontología(s).

Existen diversas dificultades a la hora de realizar un mapeo entre ambas herramientas de almacenamiento de información. Dichas dificultades radican, fundamentalmente, en la heterogeneidad entre ellas. Un clara diferencia se puede observar a la hora de obtener la semántica de los datos, algo que las ontologías proporcionan de una manera explícita y formal, a diferencia de un esquema de base de datos, el cual no provee dicha información. Además, en una ontología no se distingue entre

atributos y las relaciones que hay entre clases cuando se describe un concepto, sino que los atributos y las relaciones se refieren a propiedades. Por lo tanto, se presenta la necesidad de definir reglas y restricciones para normalizar el proceso. En este trabajo se considerarán algunas de las reglas y restricciones descritas en [41].

Antes de definir las reglas es necesario introducir los principales elementos tanto de una base de datos relacional como de una ontología. Los elementos de una base de datos relacional son: Relación, Atributo, Tupla, Dominio, *Primary Key* y *Foreign Key*, y están descritos en la tabla 4.1. Los elementos de una ontología son: Clase, *Object Property, Data Property*, Individuo y Axioma, y están descritos en la tabla 4.2. Cabe destacar que no todos los elementos anteriormente mencionados se tendrán en cuenta al momento de realizar el mapeo, como por ejemplo un Individuo de una ontología. Esto es así debido a que en el marco de este proyecto se tendrán en cuenta únicamente mapeos a nivel de esquema, y no de instancias.

Elemento	**Descripción**
Relación	Tabla con un conjunto de columnas, filas y restricciones.
Atributo	Columna de una tabla.
Tupla	Registro o fila de una tabla.
Dominio	Tipo de dato de un atributo, por ej. *integer*, *character*, *date*, etc.
Primary Key	Un atributo clave para identificar de forma única cada fila de la tabla para mantener la integridad de la entidad.
Foreign Key	Mantiene relaciones entre tablas de la base de datos para garantizar la integridad referencial. Ej. *one-to-one*, *one-to-many*, *many-to-many*.

Tabla 4.1: Elementos de una base de datos relacional.

Elemento	**Descripción**
Clase	Representa un concepto. Puede referir a cualquier elemento de la realidad.
Object Property	Define las relaciones entre las clases basadas en dominios y rangos, que corresponden a las clases que están relacionadas entre sí.
Data Property	Representa los atributos de clases de ontología según dominios y rangos; el dominio representa una clase a la que pertenece la propiedad y el rango representa el tipo y límite de datos de la propiedad.
Individuo	Una instancia de una Clase u *Object Property.*
Axioma	Los axiomas representan afirmaciones lógicamente verdaderas. Son restricciones y reglas definidas sobre conceptos y atributos.

Tabla 4.2: Elementos de una ontología.

Respecto a los tipos de mapeos definidos en la Sección 4.1, la solución desarrollada tiene en cuenta únicamente los mapeos totales. La razón detrás de esto es que estos son los que proporcionan información valiosa y los que permiten conectar varios nodos de diferentes ontologías de forma de poder navegar entre ellos y ampliar el contexto. Se podría dar el caso en donde un elemento de la base de datos se corresponda totalmente a varios elementos dentro del conjunto de ontologías que se tiene como entrada, ya que teniendo en cuenta la definición anterior, estos serían abstracciones de un mismo elemento de la realidad. En este caso, se crea una relación que una dichos elementos de las ontologías, con el objetivo de representar la equivalencia de estos, para luego a la hora de darle contexto a los elementos de la base de datos, tratar a los equivalentes de las ontologías de la misma forma.

4.3. Listado de Reglas

Las reglas que se tendrán en cuenta en el proceso de mapeo definen cómo los distintos componentes de la base de datos pueden ser mapeados a un elemento de una ontología. Para cada regla se presentará el contexto resultante en forma de ontología, de un mapeo aceptado por la misma. El contexto devuelto se expresa con distancia 1. A diferencia del grafo presentado anteriormente en la Figura

3.2, en las siguientes figuras, se cuenta con mapeos del usuario, por lo que el grafo y sus referencias cambian ya que estas involucran dichos mapeos del usuario. Para poder representar los mapeos, se utiliza una convención con colores como se muestra en la figura 4.2.

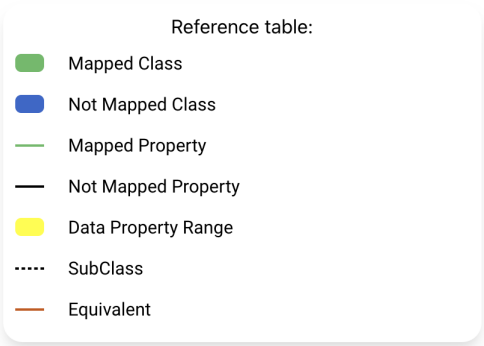

Figura 4.2: Referencia para grafos con mapeos

Los términos que OWL agrega, y que son tenidos en cuenta por el *framework* a la hora de la definición de las reglas son: `owl:Class`, `owl:ObjectProperty` y `owl:DatatypeProperty`, los cuales de aquí en adelante serán referenciados como clases OWL, *object properties* OWL y *data properties* OWL, respectivamente.

Regla #1: Mapeo de tablas a clases OWL

Sea `T` una tabla de la base de datos, entonces `T` se puede mapear a `C`, siendo `C` una clase OWL de la ontología.

Para este ejemplo se tiene en cuenta el mapeo de la tabla `universidad_curso` a la clase `Curso`, teniendo como resultado el contexto que se presenta en la figura 4.3:

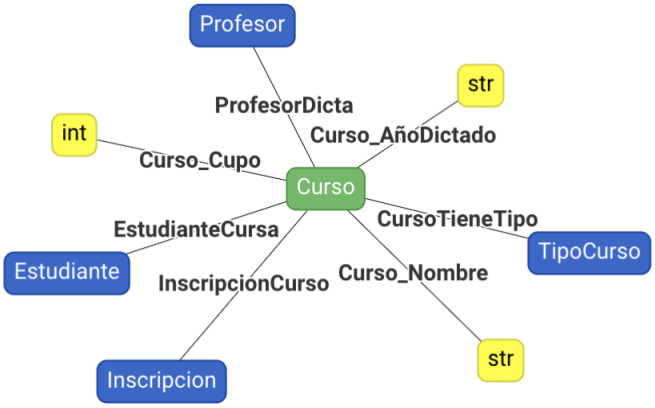

Figura 4.3: Contexto para la tabla `universidad_curso`

Regla #2: Manejo de tablas puente

Tablas puentes: tienen *foreign keys* de tablas que participan en una relación *many-to-many*, siendo estas su propio *primary key*. Representan relaciones N a N entre tablas. En el caso de estudio se tiene la tabla `universidad_estudiante_cursos`, la cual representa una relación N a N entre las tablas `universidad_estudiante` y `universidad_curso`.

Sea `T` una tabla puente de la base de datos, entonces `T` se puede mapear a `OP`, siendo `OP` una *object property* OWL de la ontología. Sea `C` cualquier clase OWL de la ontología, no esta permitido mapear la tabla puente `T` a la clase `C`. Una pre-condición para este mapeo es que el dominio y rango de la *object property* `OP` hayan sido mapeados previamente a clases.

Para ilustrar este caso se mapea la tabla puente `universidad_estudiante_cursos` con la *object property* `EstudianteCursa`, la cual tiene como dominio la clase `Estudiante` y rango `Curso`, teniendo como resultado el contexto que se muestra en la figura 4.4:

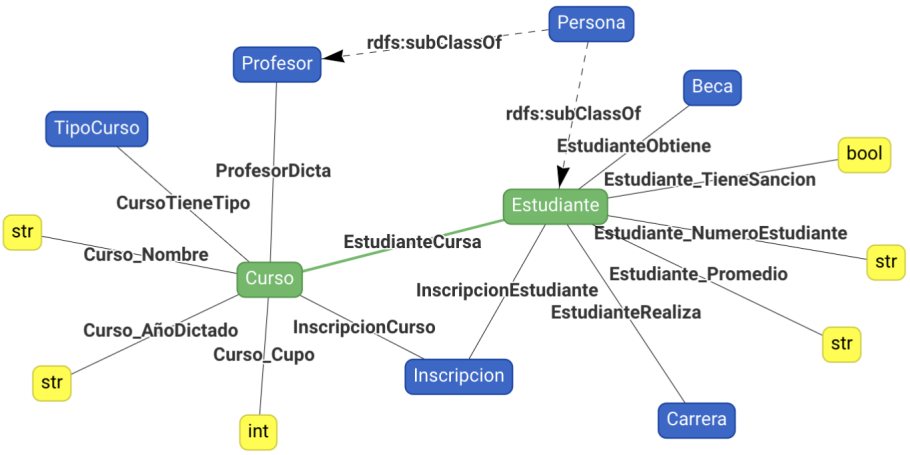

Figura 4.4: Contexto para la tabla `universidad_estudiante_cursos`

Regla #3: Mapeo de columnas (no *foreign keys*) con *data properties* OWL

Sea `c` una columna de la base de datos, siendo `c` no *foreign key*, entonces `c` se puede mapear a `DP`, siendo `DP` una *data property* OWL de la ontología.

Un mapeo posible teniendo en cuenta esta regla es el de la columna no *foreign key* `nombre`, de la tabla `universidad_carrera`, con la *data property* `Carrera_Nombre`. Como se especifica en la regla, se debe mapear también el dominio de la *data property*, el cual en este caso es la clase `Carrera`. Se obtiene entonces el contexto de la figura 4.5:

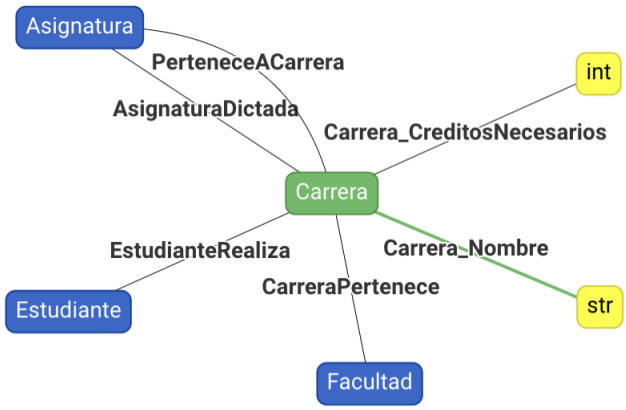

Figura 4.5: Contexto para la columna `nombre` de la tabla `universidad_carrera`.

Regla #4: Mapeo de columnas (no *foreign keys*) con clases OWL

Sea `c` una columna de la base de datos, siendo *c* no *foreign key*, entonces `c` se puede mapear a `C`, siendo `C` una clase OWL de la ontología.

Para ejemplificar la aplicación de esta regla se tiene en cuenta el mapeo de la columna `País` de la tabla `universidad_dirección` con la clase `País`, teniendo como resultado el contexto de la figura 4.6:

Figura 4.6: Contexto para la columna `País` de la tabla `universidad_dirección`.

Regla #5: Mapeo de *foreign keys* a *object properties* OWL

Sea `c` una columna de la base de datos, siendo `c` *foreign key*, entonces `c` se puede mapear a `OP`, siendo `OP` una *object property* OWL de la ontología. Una pre-condición para este mapeo es que el dominio y rango de la *object property* `OP` hayan sido mapeados previamente a clases.

Un mapeo que ejemplifica la aplicación de esta regla es el de la *foreign key* `beca_id` de la tabla `universidad_estudiante` con la *object property* `EstudianteObtiene`. Como especifica la regla, es necesario mapear al menos un elemento del dominio y otro del rango de la *object property* mapeada, por lo que se mapea también la tabla `universidad_estudiante` con la clase OWL *Estudiante* y la tabla `universidad_beca` con la clase OWL `Beca`. Luego de estos mapeos, se tiene como resultado el contexto de la figura 4.7:

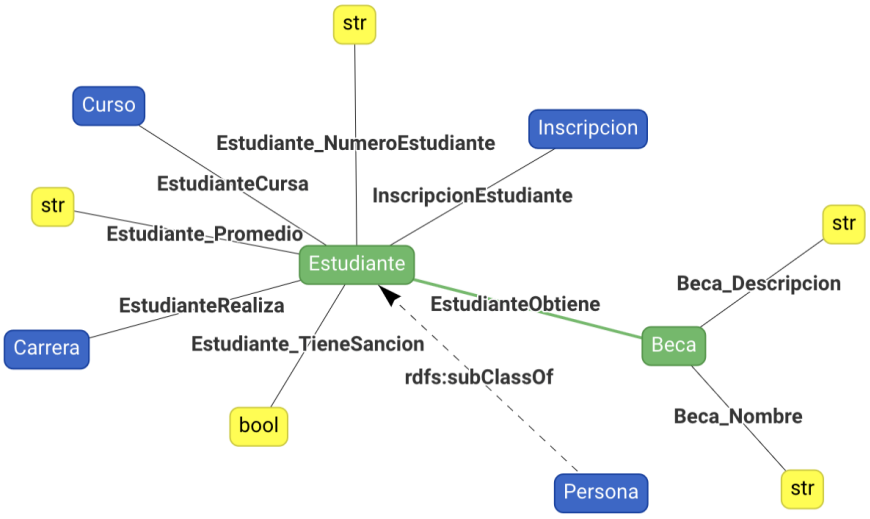

Figura 4.7: Contexto para la *foreign key* `beca_id` de la tabla `universidad_estudiante`.

4.4. Descripción de la Solución

Una vez presentada la herramienta de mapeo y las respectivas reglas, se explicará a alto nivel la solución propuesta.

Para lograr definir y generar un contexto para los datos, es necesario, como ya se mencionó anteriormente, definir los elementos y las reglas a mapear. Este conjunto de elementos es el utilizado a la hora de generar el contexto. Para esto, se creó una consulta en SPARQL para poder construir el contexto para los datos provistos como entrada. La implementación de la consulta sera abordada en la sección 6.3. A grandes rasgos, la consulta tiene como entrada la lista de los elementos de las ontologías mapeados y el tamaño del contexto requerido, representado por un entero que especifica la distancia, y retorna un grafo RDF.

La consulta tuvo como dificultad el hecho de que SPARQL no posee ningún tipo de recursión que tenga en cuenta la distancia ingresada por el usuario, por lo que la consulta únicamente contempla el caso en el cual la distancia es 1. Por este motivo, se buscó lograr una iteración por fuera de las herramientas provistas por la tecnología. Siendo `N` la distancia ingresada, el mecanismo empleado es el siguiente: se genera una iteración de `N` pasos, en donde en cada una se trata de generar un grafo RDF. La lista de elementos mapeados en el paso base sera la lista de clases ingresada por el usuario, y para los pasos siguientes, serán las clases generadas por la consulta del paso anterior. Es así como en cada iteración el grafo que se obtiene suma 1 a la distancia del contexto que se quiere obtener.

Para ilustrar este procedimiento y mostrar gráficamente el significado de la distancia, se presenta un ejemplo que obtiene en cada paso de la iteración mediante el uso de imágenes, suponiendo que el usuario quiere obtener el contexto para la clase `Facultad` con distancia 3.

En el primer paso la consulta tiene como entrada únicamente la clase mapeada por el usuario, siendo esta `Facultad`, por lo que la lista de elementos mapeados que la consulta SPARQL tiene en cuenta, en este caso contiene sólo la clase `Facultad`. Luego de ejecutar la consulta, se obtienen los elementos que se encuentran a distancia 1 de la clase `Facultad`. Se observa el contexto resultado de la iteración en la Figura 4.8.

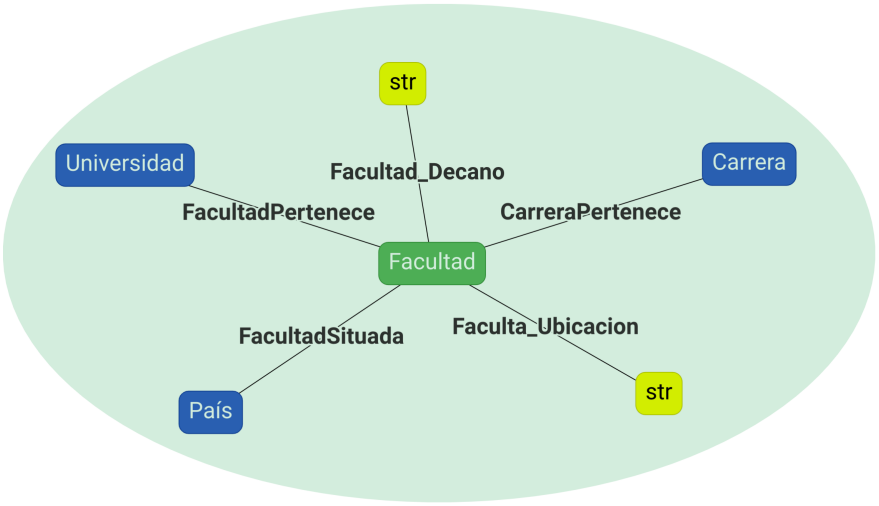

Figura 4.8: Iteración paso 1

Como se mencionó anteriormente, la distancia se puede ver como la cantidad de aristas (propiedades) que se recorren entre dos clases. Por esto la distancia entre `Facultad` y `Universidad` es 1, ya que se navegó por la *object property* `FacultadPertenece`. Esta navegación a través de las propiedades se hace sin tener en cuenta la dirección de las mismas, ya que se toman en cuenta las triplas donde las clases son objeto, así como también cuando son sujeto, contemplando cuando las propiedades son incidentes a la clase.

En el paso número dos de la iteración, esta vez la lista de elementos mapeados tiene las clases: `Facultad`, `Universidad`, `País` y `Carrera`. Al realizar la consulta SPARQL con dichos elementos, se construirá un RDF con los elementos que se encuentren a distancia 1 de los mismos. Se observa el

contexto resultado de la iteración en la Figura 4.9.

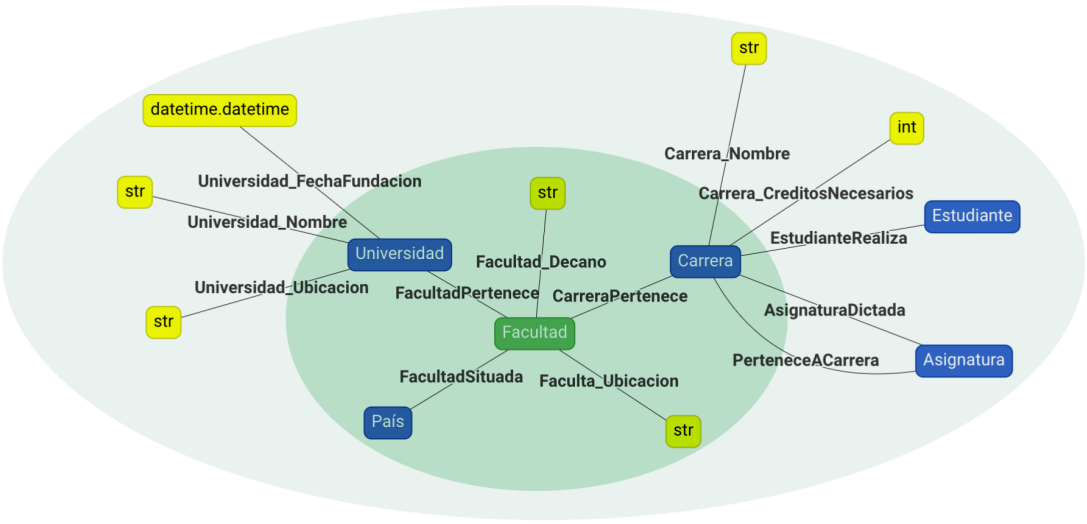

Figura 4.9: Iteración paso 2

En el último paso, a la lista de elementos mapeados se le suman las clases `Estudiante` y `Asignatura`. Luego, al realizar la consulta SPARQL con estos valores, se obtendrán los elementos que se encuentren a distancia 1 de las clases de la lista de clases mapeadas, obteniendo como resultado el contexto final, que se aprecia en la Figura 4.10.

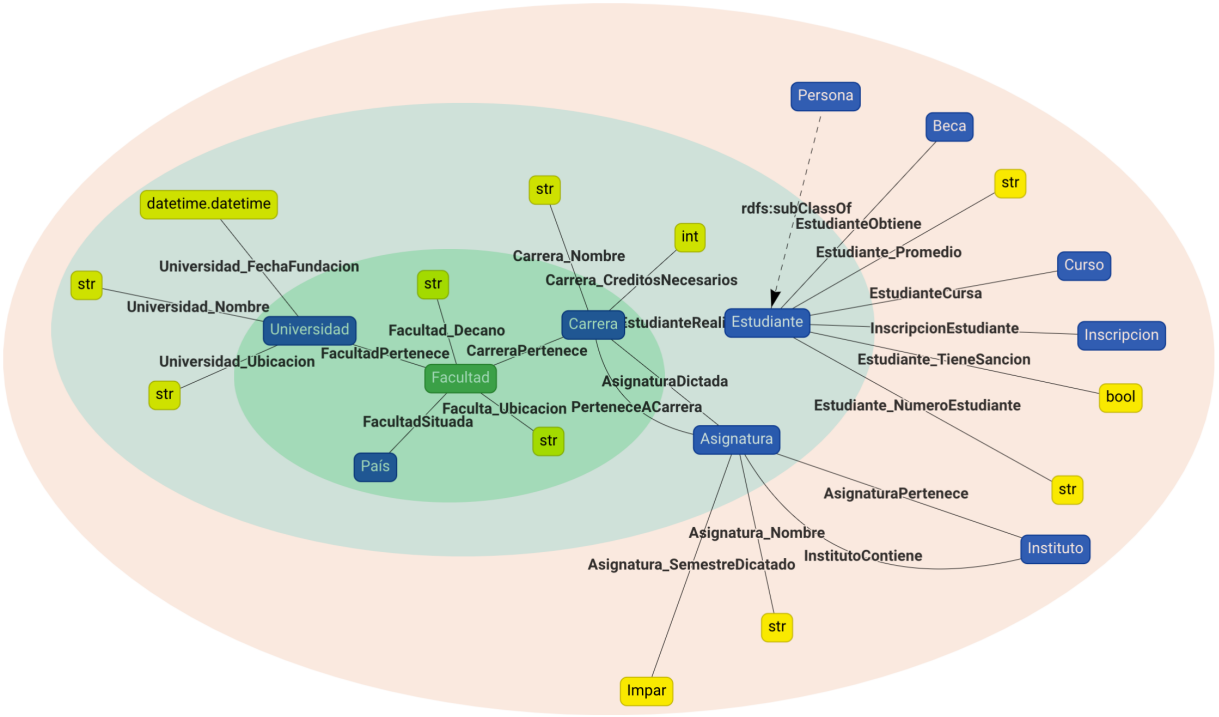

Figura 4.10: Iteración paso 3

Como se puede observar, la idea principal de la consulta SPARQL desarrollada es, dada una lista de clases, siempre obtener los elementos que se encuentran a distancia 1 de las mismas, construyendo un nuevo grafo RDF, cuyas clases serán tomadas como entrada en la siguiente iteración. De esta manera, se itera hasta llegar a la distancia requerida por el usuario.

Capítulo 5

Análisis y Diseño del *Framework* Propuesto

Primeramente, en este capítulo se hará un breve análisis de los requerimientos que se tuvieron al momento de realizar el proyecto. Luego, se describen los diferentes componentes del diseño y la arquitectura del *framework* implementado. La arquitectura se dividirá en 2 módulos principales, uno de *frontend* para interactuar con el usuario final y otro de *backend* para el procesamiento y generación del contexto, los cuales interactúan entre ellos mediante una API.

5.1. Análisis

Como principal requerimiento, se solicitó poder trabajar sobre una base de datos relacional dada, y partiendo de esta, buscar datos en una o varias ontologías de dominio existentes que le den contexto y puedan ampliarla. Es decir, un usuario debería definir en qué ontologías y en qué elementos de estas ontologías se encuentran los conceptos de la base de datos ingresada. En base a esto, el *framework* solicitado debería poder buscar en las ontologías todo lo que esté relacionado con estos elementos, hasta una distancia n. Luego, se tuvo el requerimiento de poder contar con un manejo usuarios, el cual debería ser capaz de, para cada usuario del sistema, poder almacenar un historial de las ontologías usadas, como también los contextos generados. En adición, como se mencionó en la sección 2.1, se tuvo en cuenta ambas tesis de doctorado para poder servir como insumo, y poder aportar una herramienta más a dichos trabajos. Finalmente, fue requerida una funcionalidad que permita al usuario visualizar todas las ontologías, tanto las ingresadas por el usuario, como también las generadas para representar el contexto devuelto por el *framework*.

5.2. Componente *Frontend*

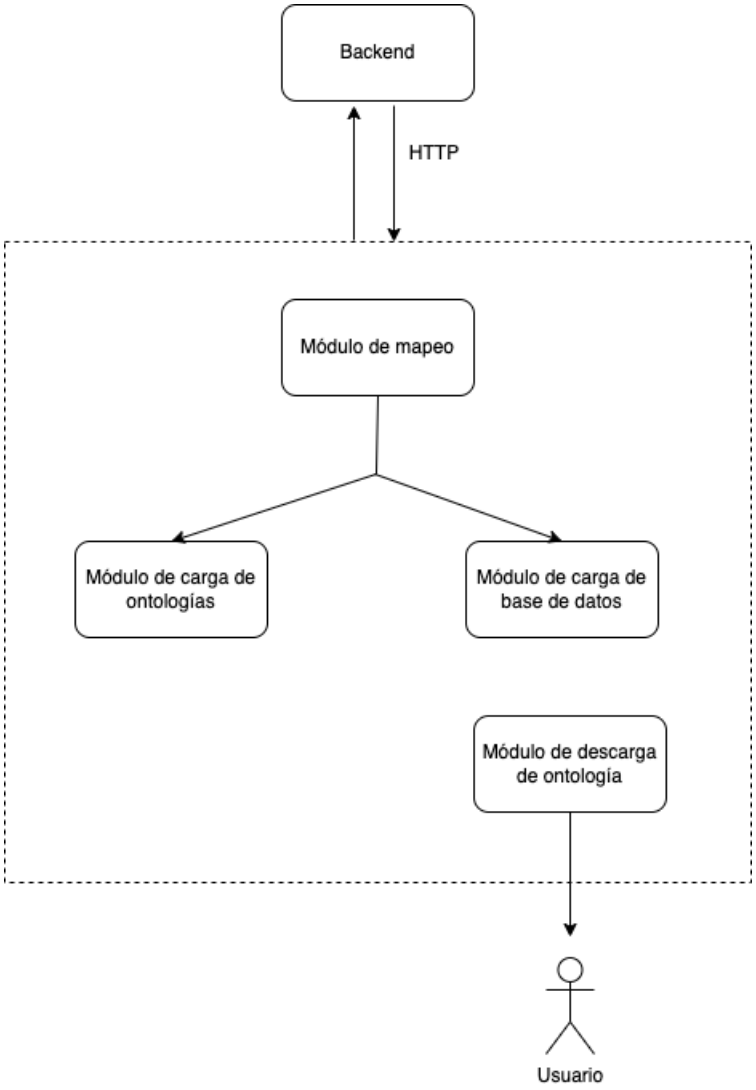

Figura 5.1: Arquitectura alto nivel *Frontend*

Como se muestra en la Figura 5.1, el componente *frontend* está dividido en cuatro módulos principales. Primeramente, se cuenta con los módulos `de carga de ontologías` y de `carga de base de datos`, siendo estos los encargados de procesar las entradas del usuario que posteriormente proveerán como entrada los elementos a mapear al `Módulo de mapeo`. Este último se encargará de procesar y guardar las elecciones del usuario para luego enviarlas al componente *backend* y validarlas. Por último se cuenta con un `Módulo de descarga` que se encarga de proveer al usuario la posibilidad

de descargar el contexto resultado. También es el encargado de generar un grafo para previsualizar el resultado. En lo que resta de esta subsección se dará una breve descripción de cada módulo.

Módulo de carga de ontologías: Este módulo es el encargado de procesar y enviar las ontologías ingresadas por el usuario al componente *backend*. Brinda la posibilidad de ingresar una o varias ontologías, en forma de archivo o IRI.

Módulo de carga de base de datos: Este módulo se encarga del procesamiento y al igual que el módulo descrito anteriormente, del envío de los datos de la conexión a la base de datos ingresada. Se cuenta con un formulario donde el usuario debe ingresar los datos correspondientes para realizar una conexión al manejador de bases de datos.

Módulo de mapeo: Este módulo es el encargado de brindar al usuario la posibilidad de realizar el mapeo de los elementos de la base de datos con los de las ontologías. Además, es el responsable de mostrar las ontologías que el usuario ingreso como entrada, en forma de grafo, para facilitar dicho proceso de mapeo.

Módulo de descarga de ontología: Este módulo es quien, una vez generada la ontología por el componente *backend*, recibe el archivo .owl como también los elementos del grafo RDF para poder presentárselo al usuario.

5.3. Componente *Backend*

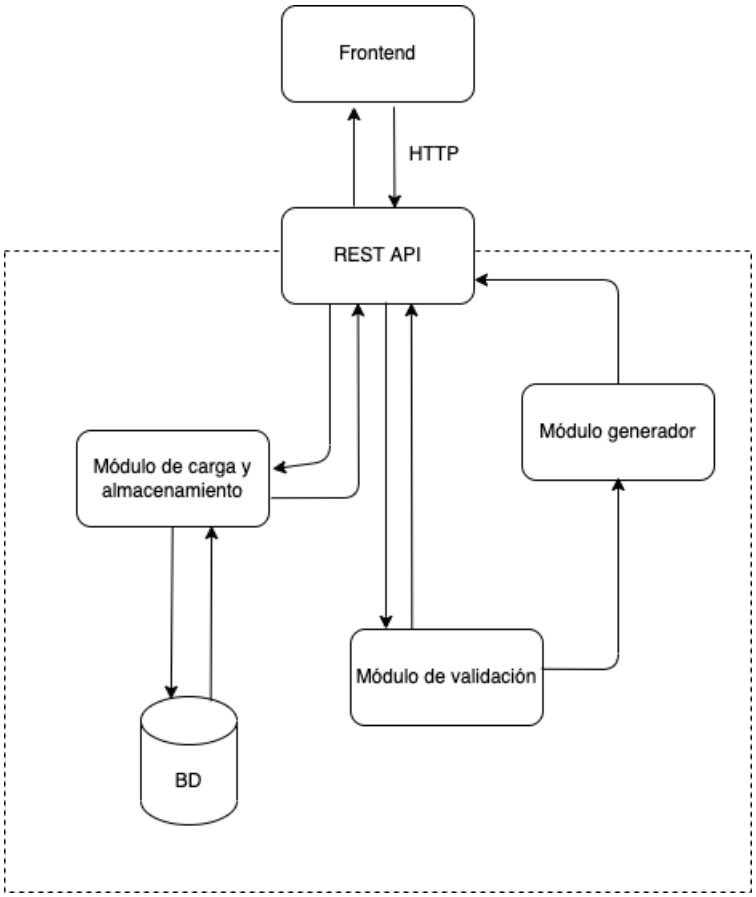

Figura 5.2: Arquitectura de alto nivel del *backend*

El componente *backend* contiene tres módulos principales. Primeramente, se tiene el **Módulo de carga y almacenamiento**, el cual se encarga de recibir las ontologías y los datos para la conexión de la base de datos, para validarlos y posteriormente almacenarlos en la base de datos local. De esta manera, se crea un modelo que almacena los datos del proceso del mapeo. Luego se tiene el **Módulo de validación**, donde se recibe el mapeo realizado por el usuario y se lo valida teniendo en cuenta las reglas definidas en 4.3, que se utilizan para dicho propósito. En caso de ser válido, el módulo encargado de la generación de la ontología final es el **Módulo generador**. Por último se cuenta con una base de datos PostgreSQL [16] para el almacenamiento de los datos.

El *framework* utilizado para el desarrollo del *backend* es Django [4]. Dicha tecnología sigue una arquitectura Modelo-Vista-Controlador (MVC), que se divide en tres capas diferentes:

- El **Modelo** es la capa de acceso a la base de datos. Esta capa contiene toda la información sobre los datos, cómo acceder a estos, cómo validarlos, cuál es el comportamiento que tiene, y las relaciones entre los datos.
- La **Vista** es la capa de presentación. Esta capa contiene las decisiones relacionadas a la presentación, es la responsable de decidir qué información se envía al *frontend* para su posterior visualización.
- El **Controlador** es la capa de la lógica de negocios. Esta capa contiene la lógica que accede al modelo y luego de obtener los datos correspondientes los envía a la vista apropiada. Se lo puede ver como un puente entre la Vista y el Controlador.

En el caso del *framework* desarrollado, cada módulo se comporta como un contenedor independiente que es capaz de implementar el MVC, teniendo su propio Modelo, Controlador, como también su propia Vista. En lo que resta de esta subsección se dará una breve descripción de cada módulo.

Módulo de carga y almacenamiento: Este módulo tiene como objetivo validar las ontologías y los datos para la conexión con la base de datos, para su posterior almacenamiento. Es necesario persistir estos datos ya que serán necesarios para consultas en etapas siguientes, y también podrán ser editados. Se almacena la información de la conexión a la base de datos ingresada, las ontologías ingresadas por el usuario, el nombre que identifica al resultado y por último el número de pasos a iterar en la generación del contexto. Luego, se genera una conexión a la base de datos, de forma de poder extraer la información de la misma. En cuanto a las ontologías, se almacenan las mismas y se asocian al modelo que almacena la información del proceso. Por último, se hace una extracción de los datos de cada una, como las clases, *objects properties* y *data properties*. Estos datos, como también la información extraída de la base de datos ingresada, son enviados al *frontend* para que posteriormente el usuario pueda ingresar un mapeo.

Módulo de validación: Este módulo tiene como objetivo validar el mapeo ingresado por el usuario. Nuevamente se crea una conexión a la base con los datos anteriormente ingresados, para obtener información de tablas, tablas puentes, *foreign keys*, etc. Se tiene en cuenta también las ontologías ingresadas en el proceso. En base a esta información se realiza la validación, teniendo en cuenta las reglas definidas en la sección 4.3. Por último se informa el resultado de dicha validación al *frontend*.

Módulo generador: Luego de que el mapeo ingresado por el usuario haya sido validado, es en este módulo donde se genera el contexto resultado mediante un grafo RDF. Este módulo también es responsable de devolver el archivo con la ontología generada, para que pueda ser descargada en el *frontend*.

5.4. Flujo de la Aplicación:

Un posible diagrama de secuencia del sistema que muestra cómo funciona la interacción entre el usuario y los principales módulos puede ser encontrado en la figura 5.3.

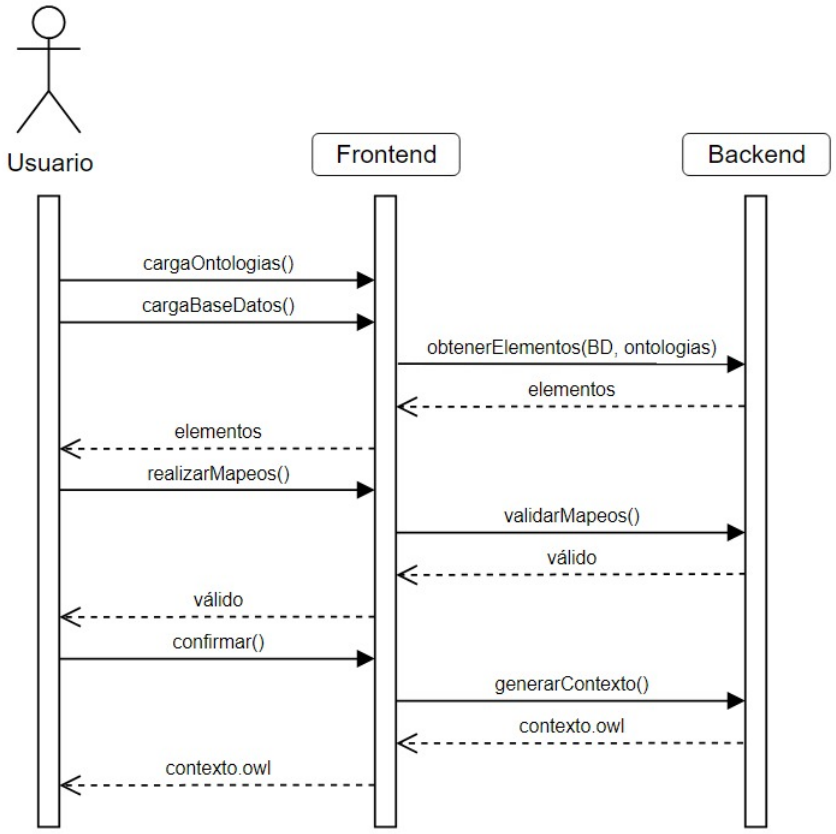

Figura 5.3: Diagrama de secuencia del sistema

El usuario ingresa los archivos iniciales mediante las operaciones `cargaOntologías` y `cargaBaseDatos`. Además, en este paso el usuario ingresa el número (entero mayor a 0) de iteraciones, que semánticamente se corresponden con el concepto de distancia. Luego el componente *frontend*, mediante la operación `obtenerElementos`, se comunica con el componente *backend* para parsear y obtener los elementos que se encuentran en dichos archivos. Una vez hecho esto, se le presentan los elementos al usuario de una forma amigable, para que posteriormente este realice el mapeo siguiendo las reglas previamente establecidas. El usuario realiza los mapeos correspondientes mediante la operación `realizarMapeo`. Luego, haciendo uso de la operación `validarMapeos` el *frontend* envía los mapeos al *backend* para realizar la validación correspondiente. El *backend* luego comunica el resultado al

frontend y este se lo comunica al usuario. En caso de ser válido el mapeo, el componente *frontend*, haciendo uso de la operación `generarContexto`, le solicita al *backend* la generación del contexto correspondiente a los mapeos. Este contexto puede ser visualizado por el usuario en forma de grafo RDF. Finalmente, se genera un archivo .owl y el usuario tiene la posibilidad de descargarlo.

Capítulo 6

Implementación

Este capítulo presentará las decisiones con respecto a la implementación, así como los diferentes componentes y dependencias con los que cuenta cada uno de los principales módulos.

En lo que respecta a las grandes tecnologías utilizadas en la solución, se decidió usar React [23] con JavaScript [8] para todo lo relacionado al módulo de *frontend* y Django [4] con Python [19] para lo relacionado al módulo de *backend*. Esta decisión se basó en la experiencia de los autores en ambas tecnologías y por el hecho que dichas tecnologías son flexibles, adaptables, existe una gran comunidad *open source* de ambas, y por último, son tecnologías populares y que funcionan en conjunto de forma estrecha.

El proyecto, el cual se encuentra alojado en un repositorio [24] de la plataforma GitHub [7], esta disponible para ejecutar localmente mediante el uso de Docker [5] y Docker Compose [6]. Con la primera, se generan contenedores para cada uno de los módulos (*frontend* y *backend*) y con la última se realiza toda la configuración de comunicación entre ambos servicios. Además, en el repositorio se cuenta con un archivo del tipo README en donde se especifican tanto los requerimientos y el proceso de instalación de la solución, como también un manual de usuario [9].

A continuación se detallarán los principales desafíos y componentes de la aplicación, tanto para el *frontend* como para el componente *backend*. Se hace una mención aparte para SPARQL ya que es uno de los puntos principales en el trabajo realizado.

6.1. *Frontend*

Las grandes decisiones en el componente de *frontend* se suscitaron alrededor de como estructurar la aplicación y las principales dependencias a utilizar. Para esto, se decidió por utilizar una estructura de páginas y componentes. Los componentes permiten separar la interfaz de usuario en piezas independientes, reutilizables y pensar en cada pieza de forma aislada. Luego, el conjunto de componentes son los que forman una página. De esta forma, se tiene una modularización clara sobre las páginas que serán accesibles por lo usuarios de la aplicación. Esto facilitará el trabajo a futuro y evitará cierto grado de retrabajo necesario con componentes que ya están creados y podrán ser reutilizados.

Para resolver ciertos problemas específicos en la aplicación se utilizaron dependencias externas. La premisa del *frontend* es intentar utilizar la menor cantidad posible de dependencias de forma de minimizar el peso de la aplicación y que sea más performante. De todas formas, hay algunas dependencias que actualmente se encuentran mantenidas y testeadas por la comunidad, lo que garantiza su buen funcionamiento, que ayudan a resolver problemas comunes, tales como la comunicación entre *frontend* con el *backend*. A continuación, se mencionarán las principales dependencias externas que se utilizaron para el desarrollo del *frontend*.

La dependencia Styled Components [26] se utilizó buscando mantener el código javascript de React lo más claro y legible posible, y extraer el código HTML y CSS hacia otros archivos de estilos. Se tienen archivos con extensión .js, donde se implementa la lógica, y archivos .styled.js, donde se le da el estilo.

Se decidió usar la librería Axios [2] que tiene resuelta la comunicación entre la aplicación y el *backend*. Dicha librería facilita la realización de los distintos pedidos HTTP hacia el *backend* y permite el manejo de errores. También permite editar los cabezales de los pedidos, de forma de poder manejar autenticación en la aplicación.

Otra dependencia que se utilizó, fue la necesaria para implementar la visualización de grafos. Este es un componente importante de la aplicación ya que permite al usuario tener una representación clara y visual del contexto obtenido como salida. Se realizó una investigación de las herramientas existentes que cubren las necesidades de la solución. Una de ellas fue NVD3 [28], que ofrece una variedad de modos de visualización de grafos, pero tiene la desventaja de no contar con una buena documentación. Otra herramienta que se investigó es Plotly [15], que ofrece una gran experiencia de usuario pero requiere de muchos recursos del sistema. Por último, se analizó la herramienta Vis.js [27], una librería de redes de código abierto que se puede utilizar para una gran variedad de representaciones, incluyendo la necesaria en el marco de este proyecto. Además, permite ele-

gir entre una variedad de formas para nodos o grupos de nodos particulares, o reemplazarlos por completo con iconos o imágenes. En base a esto, se decidió optar por el uso de esta última dependencia.

Todas estas librerías fueron utilizadas en los módulos descritos en la sección 5.2. Al contrario del *backend*, la modularización del *frontend* conlleva una dificultad mayor, lo que hace que los módulos estén interconectados y compartan algunos componentes entre si.

6.2. *Backend*

En lo que respecta al *backend*, al utilizar Django como *framework* de desarrollo, la dificultad no se localizó en crear la REST API para proveer servicios al *frontend*, si no que estuvo en la generación del contexto.

Ya que Django es un *framework* para el desarrollo web que utiliza el lenguaje Python, se tuvo que hacer una evaluación de las distintas librerías basadas en este lenguaje que se podrían utilizar al momento de trabajar con ontologías, como también con bases de datos relacionales. Con respecto a esto último, se decidió hacer uso de la librería Psycopg2 [17], siendo esta el adaptador de base de datos PostgreSQL más popular para el lenguaje en cuestión. En el proceso de evaluación se consideró algunas herramientas que brindaban la misma solución, tales como PyMySQL [18] o asyncpg [1], pero se tuvo en cuenta la recomendación de la comunidad del *framework* Django, quien sugiere el uso de Psycopg a la hora de trabajar con una base de datos PostgreSQL.

En cuanto al manejo de ontologías, se tuvo como requerimiento utilizar dos librerías. Esto debido a que ambas son compatibles con las tecnologías utilizadas en la tesis realizada por Camila Sanz. La primera es Owlready2 [14], la cual ofrece la posibilidad de trabajar con ontologías mediante objetos de Python, permitiendo su modificación, almacenamiento y realización de razonamientos. Otra de las funcionalidades importantes y necesarias en este sistema son las consultas a grafos RDF. Es aquí donde la librería Owlready2 no es de gran ayuda, ya que existen varios modos de consulta que no soporta. Es por esto que se usó la librería RDFLib [22], la cual se centra más en el manejo de RDF con Python. Esta librería brinda la posibilidad de usar la forma de consulta `CONSTRUCT`, siendo esta la que se se utiliza en la implementación de la consulta de SPARQL al momento de generar el contexto.

Como se especificó en la sección 5.3, el módulo *backend* consta de tres submódulos, el Módulo de carga y almacenamiento, el Módulo de validación y el Módulo generador. En cada uno de estos, es necesario el uso de las librerías antes mencionadas para el manejo tanto de los elementos de las ontologías como los de la base de datos. A continuación se describirá cómo se hace uso de cada una de éstas en los distintos módulos.

Módulo de carga y almacenamiento: Como ya se menciono en la sección 5, este módulo es el encargado de validar las ontologías y los datos para la conexión con la base de datos, para su posterior almacenamiento. En él, se utiliza la librería Psycopg, de manera de generar una conexión a la base de datos y extraer la información de la misma. Por otro lado, para el manejo de las ontologías, se hace uso de la librería Owlready2 para la extracción de los datos de cada una de las ingresadas por el usuario, como las clases, *objects properties* y *data properties*.

Módulo de validación: En este módulo, como ya se menciono en 5, se busca validar el mapeo ingresado por el usuario. Para esto no se necesita ninguna de las librerías anteriormente mencionadas, ya que únicamente se debe realizar una validación teniendo en cuenta reglas definidas. Este módulo le responde al *frontend* el resultado de la validación, pudiendo ser o que el mapeo fue validado con éxito o en caso contrario, por cada mapeo que el usuario ingresa y no cumple con alguna regla, un mensaje que especifica la regla que no se cumple.

Módulo generador: Como ya se mencionó anteriormente, este módulo es el responsable de, teniendo en cuenta las ontologías ingresadas por el usuario, realizar la consulta SPARQL que genera el RDF con el contexto resultado, haciendo uso de la librería RDFLib. En la sección 6.3 se detalla como se lleva a cabo la misma. Una vez realizada la consulta, se genera y retorna al *frontend* un archivo *.owl* con el grafo RDF, que podrá ser descargado por el usuario. Además, utilizando la librería OWLReady2 se extraen las clases, *objects properties* y *data properties* de la ontología generada, para que el *frontend* pueda mostrarla en pantalla de una forma amigable para el usuario.

6.3. Consulta SPARQL

En esta sección se describirá la consulta SPARQL a la que se llegó, teniendo en cuenta lo especificado en la sección 2.5.

La misma tiene únicamente una entrada, la cual es una lista de clases mapeadas, en la variable `list_mapped_elements`. Es importante señalar que únicamente se tiene en cuenta las clases, ya que como se define en la sección 4.3, siempre que se mapea una *object property*, se deben mapear también al menos una clase de su dominio y al menos una de su rango. Lo mismo sucede con las *data properties*, ya que es necesario mapear la clase del dominio de la misma. Por lo que es válido en la consulta SPARQL tener en cuenta únicamente las clases. La consulta es la siguiente:

```
CONSTRUCT ?mapped_object ?a ?b .
          ?b a ?f .
          ?x ?y ?mapped_object ;
          ?c ?d .
```

```
            ?d a ?e }
WHERE { VALUES ?mapped_object { list_mapped_elements} }
            ?mapped_object ?a ?b .
                OPTIONAL { ?b a ?f } .
            ?x ?y ?mapped_object ;
                ?c ?d .
                OPTIONAL { ?d a ?e } }
```

Como se observa, la consulta se puede dividir en dos partes. En la primera parte, determinada por el bloque `CONSTRUCT`, se especifica el template del grafo a construir. Como se señaló en la sección 2.5, este template busca generar triplas RDF basadas en los resultados de hacer coincidir los patrones de grafo especificados en el `WHERE` con las triplas del grafo de datos RDF.

Como primera observación, se tiene que los patrones de grafos definidos tanto en el bloque `CONSTRUCT` como en el `WHERE` son los mismos. Esto debido a que se quiere construir un nuevo grafo RDF cuyas triplas sean exactamente las mismas que se obtienen mediante los patrones de grafo definidos en el bloque `WHERE`. La idea es generar exactamente un subgrafo del grafo al cual se le está efectuando la consulta. Es necesario especificar esto, ya que podría suceder que en el bloque `WHERE` se defina el patrón de consulta `?b a ?f`, y que en el bloque definido por el `CONSTRUCT` se defina `?f a ?b`, lo que especifica que el tipo de f es `b`, siendo esto válido en SPARQL, pero no en lo que se quiere lograr en esta consulta, ya que de esta forma se estaría dando información errónea sobre el contexto.

Se puede observar que se hace uso de la sentencia `VALUES`, la cual consiste en, dado una lista de elementos de la ontología, realizar la consulta posterior para cada uno de ellos, evitando tener que definir nuevas variables. Otra de las sentencias que se usan dentro del bloque `WHERE` es la del `OPTIONAL`. Este tipo de sentencia sirve para definir triplas en un `WHERE` que se quiere que se cumplan, pero no que se rechacen los datos si no lo hacen. Esto es normalmente usado para obtener datos que pueden existir o no.

Para poder describir cada uno de los patrones de grafos especificados en el bloque comprendido por la sentencia `WHERE`, se tomará como ejemplo la ontología de la figura 6.1, en donde se puede observar los elementos mapeados, el contexto devuelto y los elementos no tenidos en cuenta en el resultado de la consulta.

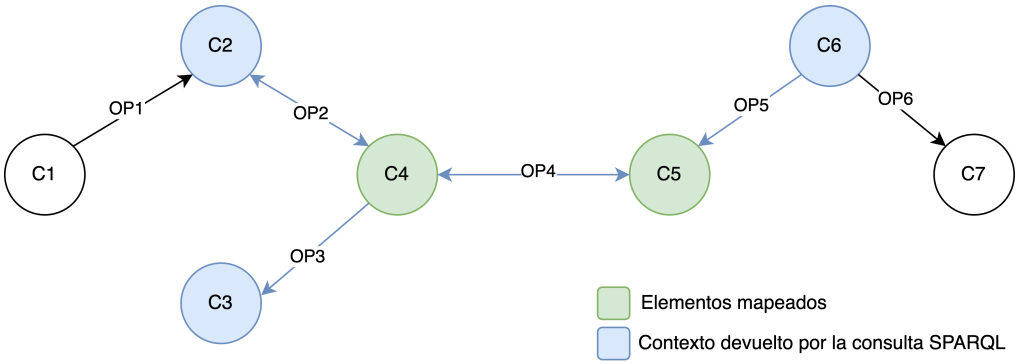

Figura 6.1: Ontología de ejemplo.

Se busca explicar para cada patrón, el conjunto de triplas que genera, como también el grafo RDF que genera.

Para el conjunto de triplas que se irán generando, se usarán los siguientes prefijos:

```
@prefix : <http://www.semanticweb.org/proyGrado#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

Suponemos que luego de un mapeo realizado por el usuario, se tiene que los elementos mapeados son las clases `C4` y `C5`, siendo estos los que forman la lista `list_mapped_elements` que se toma como entrada de la consulta. Como se dijo anteriormente, al hacer uso de la sentencia `VALUES`, la consulta ejecutará el bloque `WHERE` para cada una de estas clases.

El primer patrón, `?mapped_object ?a ?b .`, busca obtener todas las triplas en las cuales los elementos pertenecientes a la lista `list_mapped_elements`, en este caso `C4` y `C5`, aparecen como sujeto en la ontología, teniendo como resultado el grafo de la figura 6.2 y las siguientes dos triplas:

```
:C4 rdf:type owl:Class .
:C5 rdf:type owl:Class .
```

Figura 6.2: Resultado de la aplicación del primer patrón de consulta.

El segundo patrón de grafo, `OPTIONAL ?b a ?f .` , especifica que en caso de que existan, se extraiga el tipo de los objetos de las dos triplas anteriores. Como no se tienen dichas triplas en la ontología actual, no se devuelve nada.

Siguiendo con el tercer patrón de grafo, `?x ?y ?mapped_object ;`, busca obtener todas las triplas en las que `C4` y `C5` aparecen como objeto en la ontología. Se obtiene entonces el grafo de la figura 6.3 y las siguientes triplas:

```
:OP2 rdfs:range :C4 .
:OP3 rdfs:domain :C4 .
:OP4 rdfs:domain :C4 .

:OP4 rdfs:range :C5 .
:OP5 rdfs:range :C5 .
```

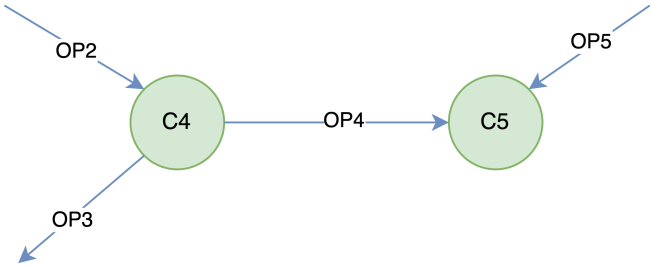

Figura 6.3: Resultado de la aplicación del tercer patrón de consulta.

El cuarto patrón de grafo, `?c ?d .`, busca obtener todas las triplas que tengan los elementos que se encuentran como sujeto en las triplas devueltas por el patrón de consulta anterior, como objeto en la ontología, obteniendo el grafo de la figura 6.4 y las siguientes triplas:

```
:OP2 rdf:type owl:ObjectProperty .
:OP2 owl:SymmetricProperty .
:OP2 rdfs:domain :C2 .

:OP3 rdf:type owl:ObjectProperty .
:OP3 rdfs:range :C3 .

:OP4 rdf:type owl:ObjectProperty .
:OP4 owl:SymmetricProperty .
```

```
:OP5 rdf:type owl:ObjectProperty .
:OP5 rdfs:domain :C6 ;
```

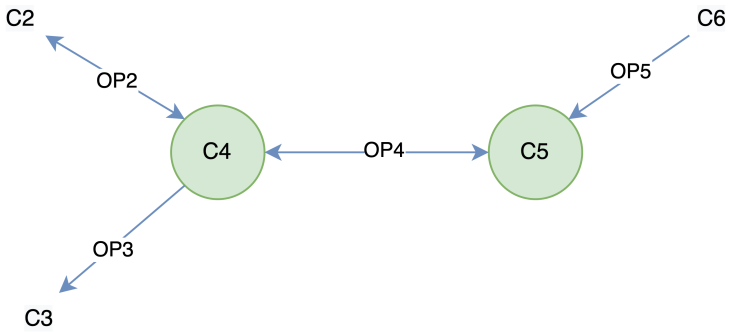

Figura 6.4: Resultado de la aplicación del cuarto patrón de consulta.

El último patrón de grafo, `OPTIONAL  ?d a ?e` , intenta obtener, en caso de que existan, los tipos de los objetos devueltos en el patrón de consulta anterior, teniendo las siguientes triplas:

```
:C2 rdf:type owl:Class .
:C3 rdf:type owl:Class .
:C6 rdf:type owl:Class .
```

Se tiene entonces como resultado final, el grafo especificado en la figura 6.5

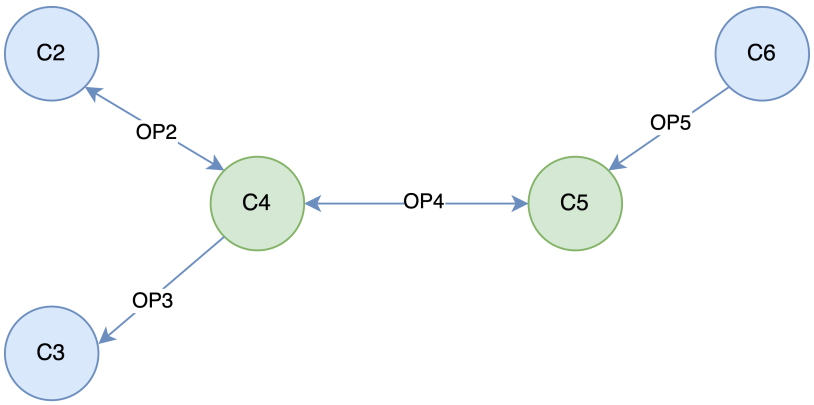

Figura 6.5: Resultado de la aplicación del quinto patrón de consulta.

Capítulo 7

Funcionamiento de la Aplicación y Resultados Obtenidos

En este capítulo, se presenta un flujo de pasos completo utilizando la aplicación, mostrando capturas del funcionamiento de la misma y analizando los resultados obtenidos. Para este flujo se utiliza la ontología y base de datos ya presentadas en el caso de estudio en la sección 3.

La aplicación cuenta con una pantalla de login que permite crear diferentes usuarios y utilizarlos a modo de perfiles. Potencialmente se puede tener distintos tipos de mapeos agrupados según nuestros usuarios.

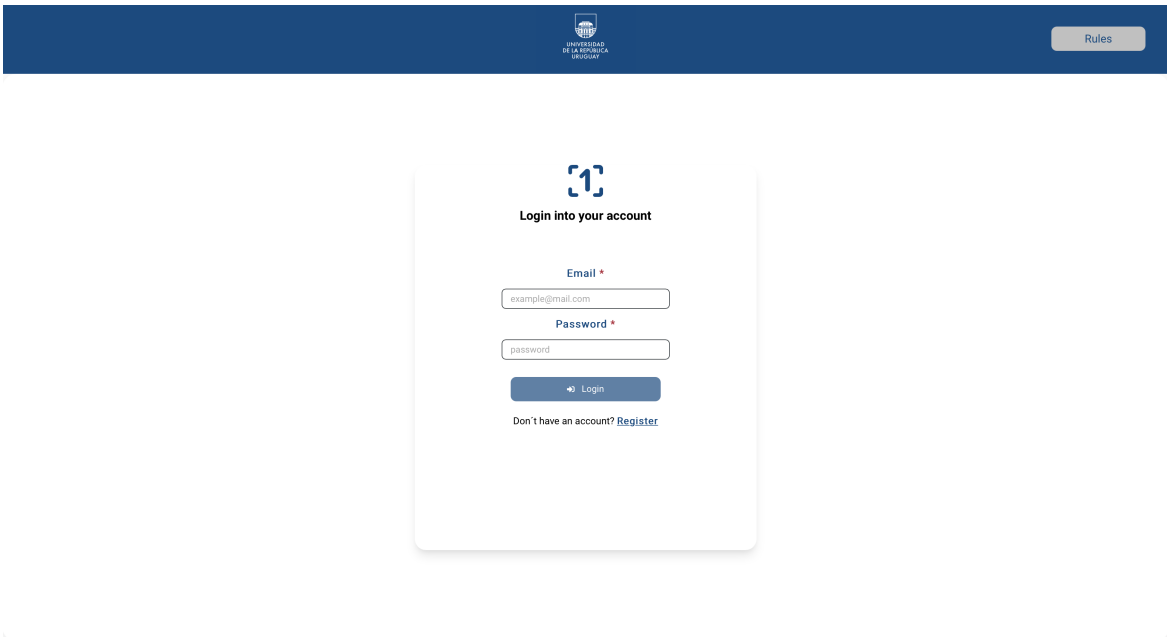

Figura 7.1: Pantalla de login

Una vez iniciada la sesión con un usuario, mediante el login de la figura 7.1, este se encuentra con un *dashboard* donde ve todos los mapeos asociados a su usuario. Aquí puede cargar un mapeo previamente creado y editarlo, o crear un mapeo nuevo, como se aprecia en la figura 7.2.

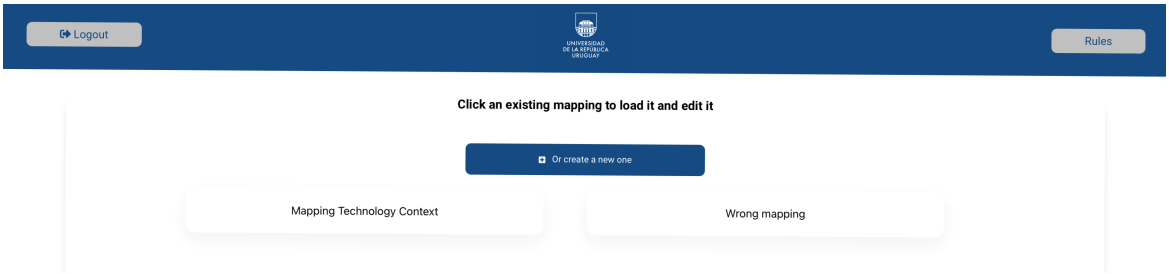

Figura 7.2: Dashboard

Al crear un mapeo nuevo, al usuario se le presenta la pantalla para cargar las ontologías para dar contexto, así como también la conexión a la base de datos PostgreSQL que se utiliza como entrada (Figura 7.3). En caso que se elija un proceso ya existente, los campos serán automáticamente completados con la información guardada. Las ontologías pueden ser subidas desde archivos *.owl* como también por su IRI. También es necesario fijar un N para fijar la cantidad de pasos a procesar, el cual especifica la distancia y un nombre para posteriormente identificar el mapeo.

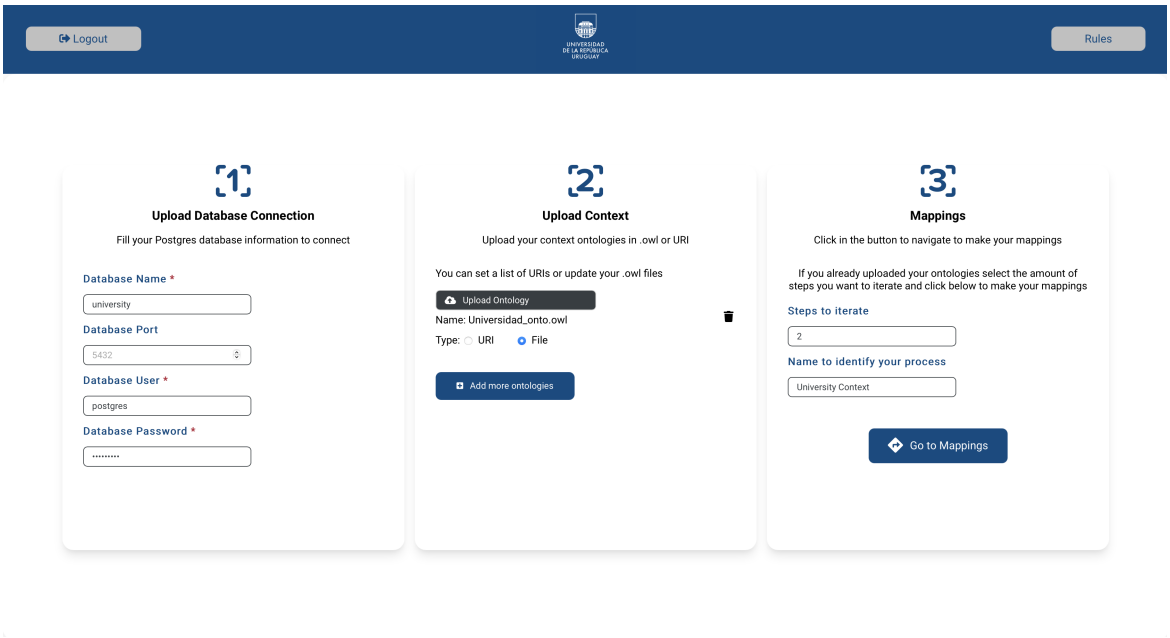

Figura 7.3: Pantalla de carga de ontologías y conexión a la base de datos.

Una vez cargadas las ontologías y la conexión a la base, la aplicación obtiene los elementos de cada una de ellas para presentárselas al usuario (Figura 7.4).

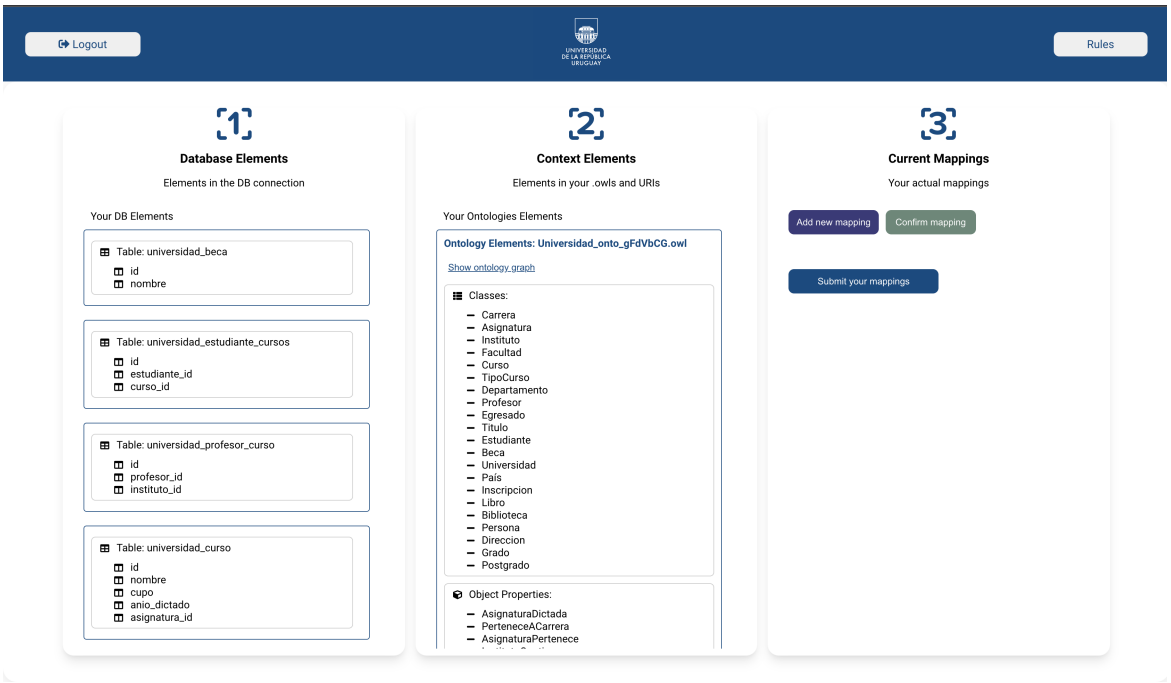

Figura 7.4: Pantalla de mapeos

En la columna de la derecha, en la Figura 7.4 se listan los mapeos actuales que el usuario va realizando donde tiene la posibilidad de eliminarlos. Cada mapeo se realiza en pasos, primero teniendo que seleccionar un elemento de la base de datos (primer columna) y luego uno o varios elementos de las ontologías cargadas (segunda columna).

En esta pantalla el usuario es capaz de visualizar las ontologías que subió en forma de grafo, de manera que el mapeo le resulte más fácil. Esta funcionalidad es muy útil cuando se trabaja con ontologías de gran tamaño y donde es difícil para el usuario recordar toda la estructura y relaciones de la ontología en cuestión.

Una vez agregados todos los mapeos válidos (Figura 7.5) que el usuario considera necesario para obtener su contexto, este puede confirmarlos y enviarlos al *backend* para procesar la ontología que obtendrá como resultado. En caso que los mapeos cumplan las reglas definidas anteriormente en 4.3, se genera el contexto y se le da la opción al usuario de descargar un archivo con la ontología generada. Si alguno de los mapeos no es válido o no cumple con las reglas definidas, se muestra un mensaje de error, solicitándole al usuario revisar las reglas específicas que no se han cumplido y asistiéndolo en las posibles soluciones que puede tener ante dicho error. De esta forma, se agiliza la creación de mapeos ya que el usuario no necesita recorrer en secuencia sus mapeos para validar que se cumplan

las reglas, si no que la aplicación indica la regla y el elemento especifico que no la cumple.

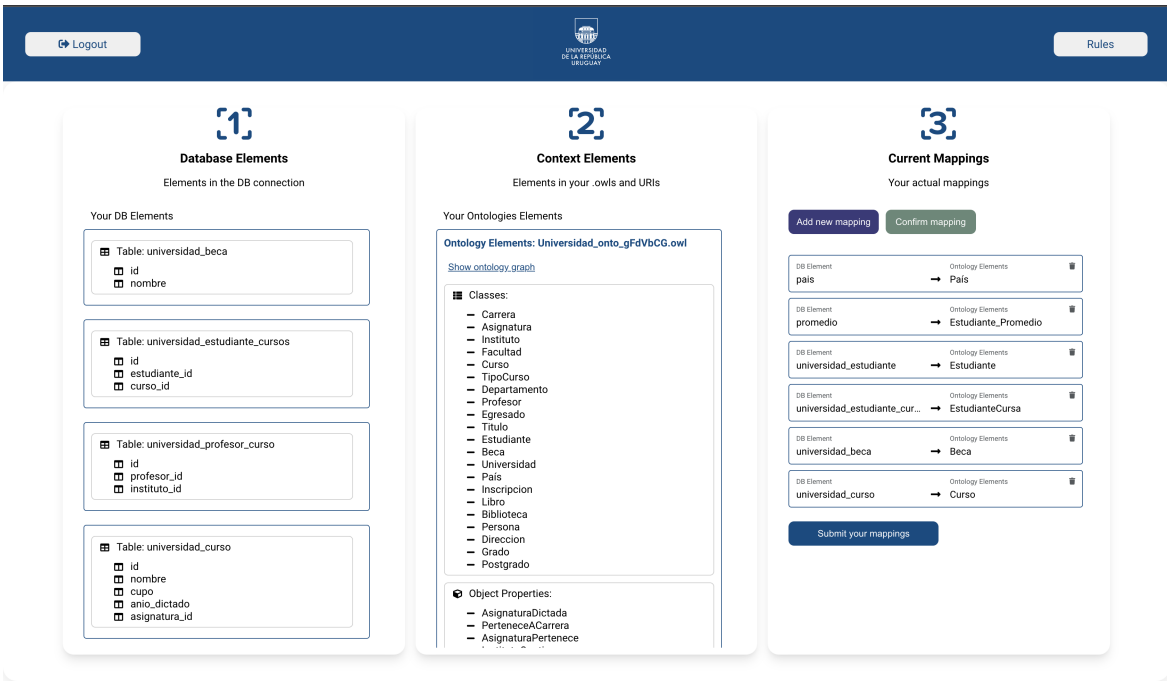

Figura 7.5: Pantalla de mapeos completa

La pantalla final del flujo, detallada en la Figura 7.6, permite al usuario visualizar el contexto generado, y le da la posibilidad de descargar la ontología como un archivo *.owl*. El grafo generado brinda la posibilidad de reorganizar los elementos, a gusto del usuario, de forma de visualizar la ontología como sea más relevante.

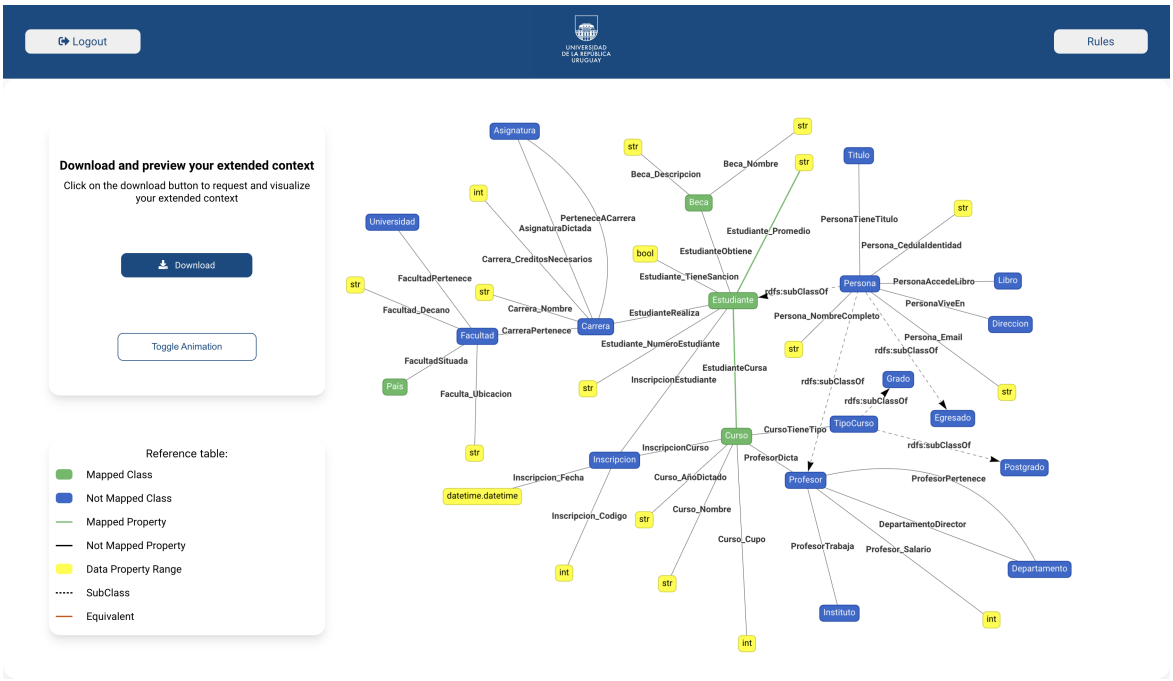

Figura 7.6: Pantalla final

El grafo resultante, mostrado en 7.6, es el resultante de generar un contexto con distancia 2. Se pueden identificar las clases mapeadas (`Estudiante`, `Curso`, `Beca`, `País`), así como también las *object properties* (`EstudianteCursa`) y las *data properties* (`Estudiante_Promedio`). Esta nueva ontología brinda contexto a los datos de entrada y permite al usuario conocer más sobre el dominio de los datos en los que trabaja.

Capítulo 8

Conclusiones y Trabajo a Futuro

Este trabajo se focaliza en dos temas principales. En primer lugar, el manejo de ontologías de dominio. La forma en que estas se definen permiten la representación de dominios, que a su vez, son un gran insumo a la hora de modelar contextos, siendo el contexto otro tema abordado en el presente proyecto.

El objetivo principal del proyecto se centró en la realización de un *framework*, con el cual se debía proveer de contexto a los datos en una base de datos relacional, a partir de ontologías de dominio.

En primer lugar, se realizó un relevamiento de la bibliografía sobre de los temas relevantes del proyecto. En particular, se relevaron las soluciones que estaban construidas al momento de la realización del proyecto, que de alguna manera u otra pudieran facilitar el cumplimiento del objetivo principal del proyecto. Si bien se encontraron varias herramientas y problemas ya resueltos, muchas de las soluciones solo abordaban un problema específico y no eran apropiadas para el proyecto. En base a esto, se propuso construir el *framework* desde cero adaptado a los requisitos que se definieron para este proyecto.

Este trabajo demuestra, mediante la aplicación de un caso de estudio, que la representación del contexto de los datos a través de ontologías de dominio, es factible. Las ontologías, y su representación en forma de grafo, permiten fácilmente su navegación con el propósito de extraer información relevante. Esta tarea se realiza haciendo uso del lenguaje OWL, como también del lenguaje de consulta SPARQL, el cual provee una serie de herramientas que posibilitan la manipulación de grafos RDF.

Por lo tanto, se logró concretar el objetivo principal, es decir, la realización de un *framework* que provee de contexto a los datos en una base de datos relacional, a partir de ontologías de dominio. Además, este *framework* brinda la posibilidad de visualizar el contexto obtenido. Esto último es una

funcionalidad de utilidad para los usuarios, ya que les permite analizar el contexto gráficamente y por lo tanto mejorar su experiencia.

8.1. Trabajo a Futuro

Si bien se contemplaron muchos casos y funcionalidades, en el marco del proyecto hay mucho campo para seguir explorando y mejorando. A nivel general, y en base al objetivo principal, se hizo foco en obtener una aplicación lo más completa posible. Sin embargo, a continuación se detallarán algunos puntos de trabajo a futuro que pueden ser explotados para generar una solución más rica y compleja.

En primer lugar se considera necesario investigar cómo dar soporte al mapeo entre instancias. Como ya se mencionó, el marco de este proyecto busca generar contextos a nivel de esquemas y no se centra en instancias. El incluir un mapeo de instancias, también involucraría cambiar la arquitectura del *frontend*, ya que se debería soportar, de una manera amigable al usuario, que se puedan ingresar los diferentes mapeos de las diferentes instancias, evitando que el usuario tenga que ingresarlas una por una manualmente.

En adición, por temas de tiempo y alcance, solo se soporta una base de datos relacional PostgreSQL como entrada. Se podría extender el módulo de carga de la base de datos para que soporte otros tipos de bases de datos (MySQL [11], Microsoft SQL Server [10], Oracle [12], etc). En el marco de este proyecto, este cambio no se consideró totalmente relevante ya que en el caso que el usuario cuente con su base de datos de entrada en alguno de los manejadores diferentes a PostgreSQL, fácilmente puede generar un respaldo y levantarlo en dicho manejador. Extendiendo lo dicho anteriormente, también podría estudiarse la posibilidad de utilizar un *Data Warehouse* [3] como entrada y llevar la aplicación a un próximo nivel, ya que se podría contextualizar una gran cantidad de datos de forma fácil y rápida.

En lo que respecta al *frontend*, un posible trabajo futuro sería extender el proyecto para que sea *responsive*. Esto significa que se adapte a todas las resoluciones donde el usuario la utilice. En el marco del proyecto únicamente se tuvo en cuenta pantallas de escritorio con dimensiones medianas, ya que se entendió que la solución iba a ser utilizada casi exclusivamente en estos dispositivos y que no era justificable la relación tiempo-beneficio de realizar toda esta mejora.

Teniendo en cuenta las reglas definidas en la sección 4.3, se podría estudiar e investigar si existen otros mapeos posibles que la aplicación no soporte al día de hoy. Un caso de uso que puede ser de interés, es el hecho de mapear una tabla o columna a una ontología en sí misma. Este caso puede ser

útil para datos como la fecha o la ubicación geográfica.

Por último, si bien la aplicación cuenta con distintas optimizaciones y siempre se buscó reducir los tiempos de generación y carga, en algunas ocasiones puede ser lenta. Esto se debe a que muchas de las ontologías son de un gran tamaño y realizar una consulta iterativa en SPARQL sobre ellas, es muy costoso. Se considera que un punto muy valioso a investigar, es el hecho de intentar reducir todavía más los tiempos de carga y obtener como resultado una aplicación más rápida y con tiempos de respuesta menores de los que se tiene actualmente.

Bibliografía

[1] asyncpg. https://magicstack.github.io/asyncpg/. Accessed: October 2021.

[2] Axios. https://axios-http.com/docs/intro. Accessed: October 2021.

[3] Data Warehouse. https://www.ibm.com/mx-es/analytics/data-warehouse. Accessed: March 2022.

[4] Django. https://www.djangoproject.com/. Accessed: March 2022.

[5] Docker. https://www.docker.com/. Accessed: March 2022.

[6] Docker Compose. https://docs.docker.com/compose/. Accessed: March 2022.

[7] Github. https://github.com/. Accessed: March 2022.

[8] JavaScript. https://www.javascript.com/. Accessed: March 2022.

[9] Manual de usuario del proyecto en GitHub. https://github.com/ssantiacevedo/proyecto-grado/tree/develop#user-guide. Accessed: December 2021.

[10] MicrosoftSQL. https://www.microsoft.com/es-es/sql-server/sql-server-2019. Accessed: March 2022.

[11] MySQL. https://www.mysql.com. Accessed: March 2022.

[12] Oracle SQL. https://www.oracle.com/database/technologies/appdev/sqldeveloper-landing.html. Accessed: March 2022.

[13] OWL W3 Org. https://www.w3.org/OWL/. Accessed: March 2022.

[14] owlready2. https://owlready2.readthedocs.io/en/v0.35/. Accessed: October 2021.

[15] Plotly. https://plotly.com/. Accessed: October 2021.

[16] PostgreSQL. https://www.postgresql.org/. Accessed: October 2021.

[17] psycopg2. https://pypi.org/project/psycopg2/. Accessed: October 2021.

[18] PyMySQL. https://pypi.org/project/PyMySQL/. Accessed: October 2021.

[19] Python. https://www.python.org/. Accessed: March 2022.

[20] RDF 1.1 Concepts and Abstract Syntax. https://www.w3.org/TR/rdf11-concepts/. Accessed: March 2022.

[21] RDF Schema 1.1. https://www.w3.org/TR/rdf-schema/. Accessed: March 2022.

[22] rdflib. https://rdflib.readthedocs.io/en/stable/. Accessed: October 2021.

[23] ReactJs. https://es.reactjs.org/. Accessed: March 2022.

[24] Repositorio del proyecto en GitHub. https://github.com/ssantiacevedo/proyecto-grado. Accessed: December 2021.

[25] SPARQL Query Language 1.1. https://www.w3.org/TR/sparql11-query/. Accessed: March 2022.

[26] Styled components. https://styled-components.com/. Accessed: October 2021.

[27] Vis Js. https://visjs.org/. Accessed: October 2021.

[28] VNVD3. https://nvd3.org/. Accessed: October 2021.

[29] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *International symposium on handheld and ubiquitous computing*, pages 304–307. Springer, 1999.

[30] Gabriela N Aranda and Francisco Ruiz. Clasificación y ejemplos del uso de ontologías en ingeniería del software. In *XI Congreso Argentino de Ciencias de la Computación*, 2005.

[31] Okba Barkat and Ladjel Bellatreche. Linking context to ontologies. In *2015 11th International Conference on Semantics, Knowledge and Grids (SKG)*, pages 57–64. IEEE, 2015.

[32] G. Chen and D. Kotz. A survey of context-aware mobile computing research. 2000.

[33] Joëlle Coutaz, James L Crowley, Simon Dobson, and David Garlan. Context is key. *Communications of the ACM*, 48(3):49–53, 2005.

[34] Real Academia Española. Definición de ontología. https://dle.rae.es/ontolog%C3%ADa. Accessed: November 2021.

[35] J. J. Carroll G. Klyne and B. McBride. Resource Description Framework (RDF): Concepts and Abstract Syntax. http://www.w3.org/TR/rdf-concepts/.

[36] D Galico, K Natanzon, C Vega, S Matalonga, and M Solari. Software sensible al contexto: definiciones y desarrollo de un estudio de caso en google glass. *Universidad ORT Uruguay*, 2015.

[37] Thomas R Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[38] Nicola Guarino. Formal ontologies and information systems. 06 1998.

[39] Nicola Guarino, Daniel Oberle, and Steffen Staab. *What Is an Ontology?*, pages 1–17. 05 2009.

[40] Peter Haase, Jeen Broekstra, Andreas Eberhart, and Raphael Volz. A comparison of rdf query languages. In *International Semantic Web Conference*, pages 502–517. Springer, 2004.

[41] Su-Cheng Haw, Jiawei Wilson May, and Samini Subramaniam. Mapping relational databases to ontology representation: A review. In *Proceedings of the International Conference on Digital Technology in Education*, pages 54–58, 2017.

[42] Gergely Héja, György Surján, and Péter Varga. Ontological analysis of snomed ct. In *BMC medical informatics and decision making*, volume 8, pages 1–5. BioMed Central, 2008.

[43] Afraz Jaffri, Hugh Glaser, and Ian Millard. Uri disambiguation in the context of linked data. 2008.

[44] Vipul Kashyap and Amit Sheth. Semantic heterogeneity in global information systems: The role of metadata, context and ontologies. *Cooperative information systems: Current trends and directions*, 139:178, 1998.

[45] Jesper Kjeldskov and Mikael B Skov. Exploring context-awareness for ubiquitous computing in the healthcare domain. *Personal and Ubiquitous Computing*, 11(7):549–562, 2007.

[46] Michael Martin and Jörg Unbehauen. Improving the performance of semantic web applications with sparql query caching.

[47] OMG Mof. 2.0 query/views/transformations rfp. *OMG Document ad/02/04/10*, 2002.

[48] Fabian Neuhaus. On the definition of 'ontology'. 09 2017.

[49] López Sabater Ontiveros Baeza. *La Economía de los Datos: Riqueza 4.0.* Grupo Planeta, 2018.

[50] Oxford. Oxford English Dictionary. `https://www.oed.com/`. Accessed: July 2021.

[51] Stefan Poslad. *Ubiquitous computing: smart devices, environments and interactions.* John Wiley & Sons, 2011.

[52] Rinton Press. Ontology and database mapping: a survey of current implementations and future directions. *Journal of Web Engineering*, 7(1):001–024, 2008.

[53] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Candidate Rec. 6 April 2006. `https://www.w3.org/TR/rdf-sparql-query/`. Accessed: November 2021.

[54] Camila Sanz. *Context based Data Quality Rules for Multidimensional Data*. PhD thesis, Pedeciba Informática. Instituto de Computación. Facultad de Ingeniería. UdelaR., 2022. work in progress.

[55] Bill N Schilit and Marvin M Theimer. Disseminating active map information to mobile hosts. *IEEE network*, 8(5):22–32, 1994.

[56] Flavia Serra. Handling context in data quality management. In *ADBIS, TPDL and EDA 2020 Common Workshops and Doctoral Consortium*, pages 362–367. Springer, 2020.

[57] Flavia Serra. *Context-based data quality management for heterogeneous Information Systems*. PhD thesis, Pedeciba Informática. Instituto de Computación. Facultad de Ingeniería. UdelaR., 2022. work in progress.

[58] Diane M Strong, Yang W Lee, and Richard Y Wang. Data quality in context. *Communications of the ACM*, 40(5):103–110, 1997.