Universidad Zaragoza
1542

Trabajo Fin de Grado

Visualización y exploración semántica de ontologías: OntView 2.0

Semantic visualization and exploration of ontologies: OntView 2.0

Autora

Carlota Quintana Elhombre

Directores

Carlos Bobed Lisbona

Eduardo Mena Nieto

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2024

AGRADECIMIENTOS

A mi director y codirector, Carlos y Eduardo, por su dedicación y apoyo constante. Sé que elegir primero a mi director y después el tema del TFG fue la mejor decisión que pude haber tomado. Gracias por haberme descubierto esta nueva área de la informática.

A mis amigos de la carrera, por las incontables visitas al laboratorio, por estar siempre presentes, por compartir los descansos y mostrar interés en mi trabajo. Han sido mi pilar durante estos años de carrera, sobretodo este último año que no ha sido fácil.

A mis amigos fuera de la carrera, que, aunque no entendían del todo lo que hacía, siempre estuvieron ahí, apoyándome y ayudándome en todo lo posible.

A mi familia, porque sin su apoyo yo no estaría aquí. A mi madre, mi mentora, porque gracias a ella, yo hoy estoy acabando de cursar Ingeniería Informática. Sin ella nunca le habría dado una oportunidad a esta carrera. A mi padre, que pese a no comprender mucho la informática, siempre me ha mostrado apoyo y se ha esforzado por entenderme.

Resumen del proyecto

En el ámbito de la gestión del conocimiento y la informática, las ontologías ofrecen un marco para modelar el conocimiento de forma organizada, definiendo conceptos y relaciones dentro de un dominio específico. Sin embargo, la falta de herramientas que ofrezcan una visualización efectiva de estas ontologías representa un obstáculo significativo. A pesar de la existencia de numerosos editores de ontologías, la mayoría carece de la capacidad para representar gráficamente el grafo de manera expresiva, lo que dificulta la comprensión de las dependencias y propiedades de los conceptos dentro de la ontología. Esta carencia limita la capacidad de los usuarios para analizar y explotar al máximo las estructuras ontológicas complejas. En este contexto, es crucial desarrollar herramientas avanzadas que ofrezcan una visualización concisa, mostrando las relaciones y propiedades de cada nodo para una comprensión más completa y profunda.

OntView 2.0 surge a partir de tres premisas clave: en primer lugar, ofrecer una herramienta visual que permita al usuario comprender de manera inmediata la definición formal de los conceptos de la ontología con solo observar la interfaz; en segundo lugar, la actualización necesaria del OntView original para aprovechar tecnologías más modernas y eficientes, lo que incluye la migración de JavaSwing a JavaFX, mejorando tanto la interfaz gráfica como el rendimiento general de la aplicación; y, por último, la capacidad de manejar tanto ontologías pequeñas como grandes, garantizando una experiencia de usuario fluida independientemente del tamaño de la ontología.

Una de las características destacadas de OntView 2.0 es su capacidad para visualizar Inclusiones de Conceptos Generales (GCI), un aspecto que no es abordado por ningún visualizador. Esta funcionalidad, junto con la manipulación avanzada de nodos, permitiendo ajustar su posición, visibilidad y conectores, mejora significativamente la visualización de ontologías y el entendimiento de las mismas. Además, OntView 2.0 mantiene el eslogan *"What you see is what you meant"*, reflejando su enfoque en mostrar de manera precisa y clara lo que el usuario realmente ha expresado al asertar la ontología.

Al optimizar estas funcionalidades y añadir nuevas capacidades, OntView 2.0 no solo mejora la experiencia del usuario, sino que también potencia la gestión y el aprovechamiento del conocimiento en diversos campos. La herramienta se convierte así en una solución más dinámica e interactiva para la exploración y análisis de ontologías.

Índice

1. Introducción

El campo de la Web Semántica ha experimentado un crecimiento considerable en los últimos años, impulsado por la necesidad de que los datos disponibles en la Web sean no solo accesibles, sino también comprensibles y procesables por máquinas. Este avance ha sido posible gracias a la implementación de ontologías, que permiten estructurar y representar el conocimiento en dominios específicos de manera que las máquinas puedan entenderlo y manipularlo. Como se menciona en "Semantic Web Technologies" [1], *"Las ontologías permiten un entendimiento común y compartido de un dominio, que puede ser comunicado tanto entre personas como entre sistemas de aplicaciones"*(traducción al español). Sin embargo, con el aumento de la complejidad y la cantidad de datos, ha surgido una serie de desafíos en la visualización y manejo de estas estructuras de conocimiento, lo que ha llevado al desarrollo de herramientas especializadas conocidas como visualizadores de ontologías.

Las ontologías son fundamentales en la Web Semántica porque ofrecen un marco para modelar el conocimiento de forma organizada, definiendo conceptos y relaciones dentro de un dominio específico. Como se menciona en *The Description Logic Handbook* [2], *"Desde hace tiempo se ha entendido que la Web podría beneficiarse de una mayor estructuración, y se reconoce ampliamente que las ontologías tendrán un papel fundamental en aportar dicha estructura."*(traducción al español). Estas estructuras permiten a los sistemas informáticos inferir nuevo conocimiento, detectar inconsistencias y clasificar datos de manera automática.

En términos simples, una ontología actúa como un diccionario especializado, que no solo define los términos, sino que también explica cómo se relacionan entre sí. Por ejemplo, en una ontología médica, "corazón" sería *un tipo de* "órgano", y a su vez, estaría relacionado con otros conceptos como "circulación". A pesar de su importancia, el manejo y la comprensión de ontologías pueden ser extremadamente complejos, especialmente cuando se trabaja con grandes volúmenes de datos o dominios muy especializados.

Uno de los retos en la Web Semántica es la visualización efectiva de ontologías. La representación gráfica de estas estructuras es esencial para que los usuarios puedan explorar, analizar y entender las relaciones y conceptos definidos en una ontología. Sin embargo, a medida que las ontologías se vuelven más grandes y complejas, la tarea de visualizarlas de manera clara y manejable se vuelve cada vez más complicado. La sobrecarga de información, la dificultad para navegar por estructuras densamente conectadas y la necesidad de herramientas interactivas que permitan manipular y explorar datos en tiempo real son algunos de los problemas que enfrentan los visualizadores de ontologías.

1.1. Alcance del proyecto

El objetivo principal del proyecto se basa en tres premisas clave. En primer lugar, ofrecer una herramienta visual que permita al usuario comprender fácilmente las definiciones formales de los conceptos con solo observar la interfaz. En segundo lugar, es la actualización del visualizador OntView original, que busca modernizar la interfaz gráfica de usuario y mejorar las prestaciones del sistema, manteniendo y extendiendo las funcionalidades existentes. El enfoque se centra en la migración de la aplicación, asegurando que las mejoras implementadas ofrezcan una experiencia más eficiente y adaptada a las necesidades actuales. Por último, garantizar que OntView 2.0 sea capaz de manejar tanto ontologías pequeñas como grandes, ofreciendo una experiencia de usuario fluida independientemente del tamaño de la ontología.

Para el modelado y uso correcto de una ontología, se requiere un conocimiento profundo de los conceptos y relaciones a representar, así como un dominio de la complejidad de los formalismos de las Lógicas Descriptivas (*Description Logics,* DL). Por lo tanto, es fundamental disponer de un visualizador que ayude tanto a los creadores a identificar posibles errores de modelado (inconsistencias) como a los desarrolladores que deseen entender y utilizar ontologías creadas por terceros. Este visualizador debe permitir observar la jerarquía de conceptos o clases, así como las clases anónimas[1], modificar la disposición de los elementos en pantalla, ocultar parcialmente el grafo, interactuar con los elementos del grafo, listar instancias de las distintas clases, visualizar la jerarquía de roles y propiedades, guardar el estado visual y la configuración, y exportar imágenes, entre otras funcionalidades.

OntView 2.0 es una herramienta desarrollada para visualizar ontologías de manera semánticamente correcta, facilitando a los usuarios la exploración y comprensión de complejas estructuras ontológicas. Dada la importancia de las ontologías en la gestión del conocimiento y el constante avance tecnológico, es esencial actualizar la base tecnológica de esta aplicación para optimizar su rendimiento. OntView 2.0 se destaca por ofrecer una visualización clara y accesible de información detallada, algo que pocas herramientas logran, lo que permite a los usuarios tomar decisiones informadas y descubrir nuevas relaciones entre datos.

Inicialmente, se consideraron dos versiones de OntView: una más antigua [3] en Java y otra que mezclaba Java + JavaScript para la capa de visualización. Tras la evaluación de ambos códigos, y después de reconciliar los lenguajes visuales de ambas versiones, se decidió partir de la versión más antigua. Se abandonaron algunas de las decisiones visuales adoptadas en la versión más reciente priorizando simplificar la pila tecnológica (aparte de la falta de documentación y del hecho de que dicha versión también estuviera afectada por el uso de librerías no mantenidas). La elección de la versión antigua también se debió a su mejor adaptabilidad para la actualización. A pesar de su antigüedad, esta versión no ha quedado obsoleta desde un punto de vista teórico, lo cual es sorprendente, ya que durante este tiempo no ha surgido ninguna solución mejor en términos conceptuales. Sin embargo, desde el punto de vista tecnológico, era necesario actualizarla debido a problemas relacionados con *bugs*, escalabilidad y visualización.

[1]Conceptos que cumplen ciertas condiciones específicas, pero que no tienen un nombre asignado en la ontología.

1.2. Enfoque del proyecto

El proyecto OntView 2.0 tiene como uno de los objetivos actualizar el visualizador OntView original, mejorando tanto su rendimiento como su usabilidad. Para llevar a cabo esta actualización, se realizará un análisis exhaustivo de las posibles tecnologías a utilizar en la migración, con especial atención a la combinación de herramientas modernas que puedan ofrecer un equilibrio óptimo entre rendimiento y flexibilidad.

La planificación del proyecto incluirá la conversión de la interfaz de usuario, desarrollada en JavaSwing, hacia una plataforma más moderna que permita mejorar la experiencia de usuario. Se considerará la posibilidad de mantener las tecnologías existentes durante la transición para garantizar la estabilidad, pero si esto genera incompatibilidades, se optará por una migración completa hacia la nueva tecnología.

El desarrollo y la migración del proyecto se llevarán a cabo utilizando Eclipse [4], una herramienta de desarrollo avanzada que facilita el proceso de refactorización y aseguramiento de la calidad del código. Para el control del proyecto, se implementarán tableros Kanban, permitiendo un seguimiento detallado del progreso y de las tareas.

Es relevante mencionar que, además de la implementación técnica, se ha llevado a cabo una planificación exhaustiva de la migración y la extensión del proyecto. El desarrollo realizado ha sido considerable y, dado el enfoque en Ingeniería del Software, la memoria se ha centrado en esta especialidad. Este proyecto podría estar enmarcado en la especialidad de Computación, y por ello, gran parte del contenido de los Anexos está enfocado en aspectos propios de Computación. Esto refleja la naturaleza dual del proyecto, que abarca tanto Ingeniería del Software como Computación. Asimismo, dada la dimensión de investigación que posee este software, se tiene planteado preparar una publicación para enviarla a la conferencia internacional sobre Web Semántica.

El trabajo de desarrollo se está llevando a cabo en colaboración con el grupo de investigación SID de Zaragoza. Esta colaboración asegura que el proyecto esté alineado con los últimos avances en el campo de las ontologías.

1.3. Contenido de la memoria

La estructura de la memoria se organiza de la siguiente manera:

- El **capítulo 2** resume los fundamentos tecnológicos del proyecto, incluyendo conceptos de ontologías, herramientas utilizadas y visualizadores comparados.
- El **capítulo 3** aborda la planificación estratégica de OntView 2.0, enfocándose en los requisitos, el diseño de la interfaz, y la estructura del sistema.
- El **capítulo 4** describe la migración de la aplicación, destacando la estrategia, la elección de tecnologías y los desafíos técnicos superados en la actualización.
- El **capítulo 5** sintetiza las conclusiones del proyecto, la planificación temporal, y propone futuras mejoras y extensiones para la evolución de la herramienta.

2. Contexto tecnológico

2.1. Conceptos relacionados con el proyecto

En esta sección se explicarán los conceptos fundamentales que se mencionarán a lo largo de la memoria.

2.1.1. Ontología

Una ontología es la especificación formal de una conceptualización [5]. En términos más sencillos, es una forma estructurada de organizar y describir el conocimiento sobre un tema específico.

Los componentes de una ontología son [6]:

- **Conceptos/Clases:** conjuntos de elementos o individuos que comparten ciertas características (e.g., en una ontología de animales, una clase podría ser "Mamíferos", que incluiría a todos los mamíferos). Caben destacar dos conceptos que suelen añadirse en todos los formalismos:
  - **Top/Thing:** concepto que subsume a todos los conceptos de la ontología.
  - **Bottom/Nothing:** subclase de todos los elementos de la ontología. Subsume todos los conceptos no satisfacibles.
- **Atributos:** aspectos, propiedades, rasgos, características, o parámetros que objetos y clases pueden tener.
- **Relaciones:** representan la interacción entre los conceptos del dominio[1].
- **Instancias/Individuos:** Representan individuos pertenecientes a un concepto (e.g., si la clase es "Perro", una instancia podría ser "Rex", un perro en particular).

Y, dependiendo del formalismo utilizado, se pueden encontrar los siguientes componentes:

- **Restricciones:** establecen descripciones formales de lo que debe ser verdad. Existen restricciones internas (las deben cumplir el concepto) y externas (son las relaciones entre conceptos).

[1] Objetos o entidades que pertenecen a un área de conocimiento específica.

– **Reglas:** declaraciones con forma de oraciones si-entonces (antecedente-consecuente) que describen inferencias lógicas que pueden ser derivadas de una aserción en una forma particular.

En la Sección 3.4 se detalla cómo se visualizan los elementos de una ontología en el visualizador OntView 2.0, incluyendo una descripción de cada uno de ellos junto con su representación gráfica. Para introducir las ontologías de manera sencilla, en la Figura 2.1 se muestra un ejemplo de una ontología llamada *"proyectos.owl"*.

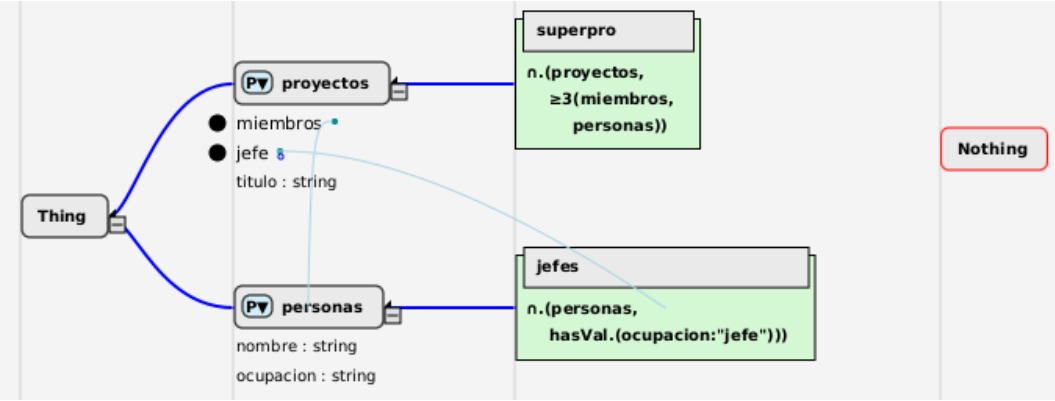

Figura 2.1: Ejemplo de una ontología

Esta ontología describe un sistema de gestión de proyectos introduciendo como vocabulario básico "personas" y "proyectos". Las personas tienen propiedades como "nombre" y "ocupación", y los proyectos están compuestos por "miembros", que son personas. Sobre esos conceptos básicos, se definen dos conceptos mediante expresiones: "jefes", toda aquella personas cuya ocupación es "jefe"; y "superpro", todo aquel proyecto que cuenta con al menos tres miembros.

2.1.2. *Description Logics* (DL)

Las Lógicas Descriptivas (*Description Logics*, DLs) son una *"familia de formalismos de representación del conocimiento que representan el conocimiento de un dominio de aplicación, definiendo primero los conceptos relevantes del dominio y luego utilizando estos conceptos para especificar las propiedades de los objetos e individuos que ocurren en el dominio"* [2]. En términos simples, las Lógicas Descriptivas (DLs) son lenguajes formales que permiten organizar y definir el conocimiento sobre un tema, describiendo conceptos, objetos y las relaciones entre ellos, de manera que se puedan hacer deducciones lógicas automáticamente. Son el formalismo adoptado por el W3C para OWL.

El siguiente ejemplo es un fragmento del que se presentó previamente en la Sección 2.1.1. Como se puede observar en la Figura 2.2, la notación formal en *Description Logics* es precisa pero bastante abstracta y difícil de comprender sin un conocimiento técnico previo.

$$\ldots$$
$$\exists miembros.\top \sqsubseteq proyectos$$
$$\top \sqsubseteq \forall miembros.personas$$
$$jefe \sqsubseteq miembros$$
$$\top \sqsubseteq \forall jefe.jefes$$
$$\ldots$$
$$jefes \equiv Personas \sqcap ocupacion(\text{“}jefe\text{”})$$
$$superPro \equiv proyectos \sqcap \geq 3\ miembros.personas$$
$$\ldots$$

Figura 2.2: Extracto de la ontología *proyecto.owl* en *Description Logics*

2.1.3. OWL y OWLAPI

OWL (*Web Ontology Language*) es un lenguaje para implementar y compartir ontologías en la WWW (World Wide Web). OWL tiene como objetivo facilitar un modelo que permita representar ontologías a partir de un vocabulario más amplio y una semántica más expresiva que la que permite RDF [7].

OWL es uno de los lenguajes más utilizados y reconocidos para representar ontologías, especialmente en el contexto de la Web Semántica, debido a que tiene mayor capacidad para expresar significado y semántica que XML, RDF, y RDF-S [6].

A continuación, se muestra el mismo fragmento del ejemplo presentado en la Sección 2.1.2, pero utilizando el lenguaje OWL. Este fragmento es solo una pequeña parte de la ontología completa descrita en la Sección 2.1.1, la cual se encuentra en el Anexo E. Aunque OWL ofrece una mayor legibilidad que la notación en *Description Logics*, sigue siendo un lenguaje técnico y complejo, lo que dificulta su comprensión.

```
1                    ...
2  <owl:ObjectProperty rdf:about="#miembros">
3          <rdfs:range rdf:resource="#personas"/>
4          <rdfs:domain rdf:resource="#proyectos"/>
5      </owl:ObjectProperty>
6      <owl:ObjectProperty rdf:about="#jefe">
7        <rdfs:range rdf:resource="#jefes"/>
8        <rdfs:subPropertyOf rdf:resource="#miembros"/>
9      </owl:ObjectProperty>
10                   ...
11     <owl:Class rdf:about="#jefes">
12       <owl:equivalentClass>
13         <owl:Class>
14           <owl:intersectionOf rdf:parseType="Collection">
15              <rdf:Description rdf:about="#personas"/>
16              <owl:Restriction>
17                <owl:onProperty rdf:resource="#ocupacion"/>
18                <owl:hasValue>jefe</owl:hasValue>
19              </owl:Restriction>
20          </owl:intersectionOf>
```

```
21          </owl:Class>
22        </owl:equivalentClass>
23      </owl:Class>
24      <owl:Class rdf:about="#superpro">
25        <owl:equivalentClass>
26          <owl:Class>
27            <owl:intersectionOf rdf:parseType="Collection">
28              <rdf:Description rdf:about="#proyectos"/>
29              <owl:Restriction>
30                <owl:onProperty rdf:resource="#miembros"/>
31                <owl:onClass rdf:resource="#personas"/>
32                <owl:minQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">3
33                </owl:minQualifiedCardinality>
34              </owl:Restriction>
35            </owl:intersectionOf>
36          </owl:Class>
37        </owl:equivalentClass>
38      </owl:Class>
39                  ...
```

Por otro lado, OWLAPI [8] es una API de Java y una implementación de referencia para crear, manipular y serializar ontologías OWL. La versión más reciente de la API está orientada hacia OWL 2. OWLAPI sirve como interfaz para trabajar con razonadores semánticos como Openllet [9], FaCT++ [10], HermiT [11], Pellet [12], Racer [13], entre otros.

2.1.4. Razonadores

Un razonador semántico [3] es un software diseñado para derivar conclusiones lógicas a partir de un conjunto de hechos o axiomas establecidos. Este tipo de software amplía el concepto de motor de inferencia, ya que incorpora un conjunto más amplio de mecanismos para procesar y manejar información.

Las reglas de inferencia suelen estar definidas en un lenguaje de ontologías, como OWL. Muchos razonadores emplean lógica de primer orden para realizar estas inferencias, utilizando técnicas de encadenamiento hacia adelante o hacia atrás para derivar nuevas conclusiones a partir de los datos existentes. El razonador empleado en el proyecto es Openllet [9].

2.1.5. Extractores de conocimiento

Un extractor de conocimiento en ontologías es una herramienta o algoritmo diseñado para identificar y extraer información relevante y significativa de las mismas. En este proyecto se han empleado tres técnicas para detectar los conceptos más relevantes: KCE, PageRank y RDFRank.

KCE El algoritmo de Extracción de Conceptos Clave (KCE) analiza la estructura de la ontología y aplica heurísticas psicológicas y cognitivas para seleccionar los conceptos más

relevantes para entender la ontología de forma global [14]. Esta técnica se basa en algoritmos que analizan y procesan la información disponible, identificando términos y relaciones significativas que pueden ser utilizados para estructurar y representar el conocimiento de manera más efectiva.

PageRank El método PageRank fue desarrollado para evaluar la importancia de las páginas web a través de su estructura de enlaces. Sin embargo, la matemática de PageRank es completamente general y se aplica a cualquier grafo o red en cualquier dominio [15]. La importancia de un nodo se calcula iterativamente, distribuyendo el valor del PageRank de un nodo entre todos los nodos a los que enlaza, con un factor de amortiguación para ajustar la influencia de enlaces aleatorios. En resumen, un nodo es más importante si está bien conectado y si los nodos que lo enlazan también son importantes.

RDFRank El método RDFRank [16] es un algoritmo que identifica las entidades más populares en un grafo RDF examinando su interconexión. La popularidad de las entidades se puede utilizar para ordenar los resultados de las consultas de manera similar a como lo hacen los motores de búsqueda en internet, al igual que Google ordena los resultados de búsqueda utilizando PageRank. El componente RDFRank calcula un peso numérico para todos los nodos en todo el grafo RDF almacenado en el repositorio, incluyendo URIs[2], nodos en blanco[3] y literales. Los pesos son números de punto flotante con valores entre 0 y 1, que se pueden interpretar como una medida de la relevancia o popularidad de un nodo. Cabe destacar que RDFRank sólo es una adaptación bidireccional de PageRank.

2.2. Software empleado

A continuación, se presenta el software utilizado en el desarrollo del proyecto:

- **Eclipse v.4.31.0**[4]: entorno de desarrollo.
- **IntelliJ IDEA 2023.2.6 (Ultimate Edition) v.17.0.10**[19]: entorno de desarrollo.
- **JavaFX v.22.0.1**[20]: framework para crear interfaces gráficas en Java (Ver Anexo F).
- **Java v.17**[21]: lenguaje de programación del proyecto.
- **OWLAPI**[21]: API de interacción con elementos de OWL.
- **Openllet**[9]: razonador semántico.
- **Drawio**[22]: herramienta para la creación de diagramas.
- **Trello**[23] **y GitHub**[24]: herramienta para la gestión de tableros Kanban.
- **GitHub**[24]: control de versiones del código.

[2]Cadena de caracteres que identifica los recursos de una red de forma unívoca [17].
[3]Nodos sin nombre en un grafo RDF [18].

2.3. Visualizadores de ontologías

Existen varios visualizadores de ontologías. Entre los más destacados se encuentra Protégé [25] con distintos *plugins*[4] como OWLViz [26] y OntoGraf [27], WebVOWL [28] y OWLGrED [29].

A continuación, se presenta una tabla comparativa que incluye los visualizadores mencionados junto con OntView 2.0. En la Tabla 2.1 se muestran los requisitos funcionales que deberían cumplir los visualizadores de ontologías, y se indica si cada visualizador lo ofrece o no. Para un análisis más detallado sobre las características de estos visualizadores, ver el Anexo A.

	OWLViz	OntoGraf	WebVOWL	OWLGrED	OntView
Mostrar grafo asertado	✓	✓	✓	✓	✗ *
Mostrar grafo inferido	✓	✓	✗	✗	✓
Uso de razonadores	✓	✓	✗	✗	✓
Visualizar jerarquía	✓	✓	✗	✗	✓
Visualizar clases anónimas	✗	✗	✗	✗	✓
Visualizar GCIs **	✗	✗	✗	✗	✓
Modificar disposición de elementos en pantalla	✗	✓	✓	✗	✓
Ocultar parcialmente el grafo	✓	✓	✓	✗	✓
Listar instancias de las clases	✗	✗	✓	✓	✓
Visualizar jerarquía de roles y propiedades	✗	✗	✓	✓	✓
Guardar estado	✗	✓	✗	✗	✓
Exportar imágenes	✓	✓	✓	✓	✓

Tabla 2.1: Comparación entre visualizadores

* Se tiene marcado como trabajo a futuro.

** Axiomas de subsunción en los cuales los conceptos pueden ser anónimos.

[4]Un *plugin* es un componente de software que añade una característica específica a un programa de software existente, sin necesidad de modificar el programa base.

3. Planificación del visualizador

Este capítulo presenta los elementos clave para el desarrollo del visualizador, con un enfoque específico en la rama de Ingeniería del Software. A lo largo de las siguientes secciones, se abordará la identificación de los *stakeholders*, la definición del diccionario de datos, la especificación de los requisitos funcionales y no funcionales, el diseño de la interfaz gráfica de usuario (GUI) y la elaboración de diversos diagramas que representan la estructura y el comportamiento del sistema. Estas áreas de atención son fundamentales en la disciplina de Ingeniería del Software, ya que permiten garantizar que el desarrollo del sistema se realice de manera estructurada, eficiente y alineada con los objetivos del proyecto.

Se ha optado por realizar una versión simplificada del Documentación de Especificación de Requisitos (DER), debido a las limitaciones de espacio en la memoria y al hecho de que la misma también abarca la rama de Computación.

3.1. Stakeholder

Un *stakeholder*[1] es cualquier persona/grupo que se verá afectado por el sistema de forma directa o indirecta [30]. Está compuesto por dos palabras en inglés: "stake", que significa apuesta, y "holder", que puede ser traducido como "poseedor" [31].

El *stakeholder* de este proyecto es un usuario especializado en el desarrollo y uso de ontologías. Este tipo de usuario posee un conocimiento profundo sobre la estructura y gestión de ontologías, lo cual es crucial para el uso efectivo de la herramienta. Dado que OntView 2.0 está diseñado para manejar información compleja y detallar las relaciones y propiedades de los nodos dentro de una ontología, es poco probable que un usuario que no sepa al menos de esquemas relacionales encuentre la aplicación útil o accesible.

El enfoque del proyecto está centrado en personas que trabajan directamente con ontologías, como investigadores, desarrolladores de sistemas de gestión del conocimiento, y académicos que requieren herramientas avanzadas para visualizar y analizar estructuras ontológicas complejas. Estos usuarios necesitan capacidades avanzadas de visualización que les permitan explorar y manipular las ontologías de manera eficiente y precisa.

[1]Terminología empleada en la especialidad Ingeniería del Software. Hace referencia al usuario objetivo.

3.2. Diccionario de datos

1. **Ontología**: Conjunto estructurado de términos, relaciones e instancias que representan el conocimiento en un dominio específico.
2. **Razonador**: Software utilizado para inferir relaciones y propiedades adicionales a partir de una ontología.
3. **Extractor de conceptos**: Herramientas o algoritmos utilizados para identificar y extraer conceptos clave de una ontología.
4. ***Tooltip***: Elemento de la interfaz que muestra información adicional.
5. ***Canvas***: Componente que proporciona una superficie de dibujo para gráficos personalizados.
6. ***Namespaces***(Espacio de nombres): Identificador que define un espacio de nombres único para los términos utilizados en la ontología.

3.3. Requisitos funcionales y no funcionales

Debido a que el proyecto se centra en la actualización y mejora de las funcionalidades del visualizador, se han generado dos tablas que muestran las funcionalidades del proyecto original (OntView 1.0) y las nuevas funcionalidades añadidas en OntView 2.0. Para evitar la repetición de los requisitos, en la segunda tabla solo se presentarán los nuevos requisitos y los que hayan evolucionado.

Cabe destacar que la mayoría de las funcionalidades de la versión original se han mejorado, por lo que no se ha realizado una gran adición de nuevas funcionalidades. En la sección de *"Problemas y soluciones"* (Sección 4.4) se detalla de forma más concisa dichas mejoras de funcionalidades.

3.3.1. OntView 1.0

Requisitos funcionales: Sistema

La Tabla 3.1 muestra los requisitos funcionales del sistema.

Id	Requisitos
RFS-1	El sistema debe poder cargar ontologías(1).
RFS-2	El sistema debe integrar razonadores(2) como Openllet para procesar las ontologías(1).
RFS-3	El sistema debe permitir la exportación de la visualización del grafo a imágenes.

Tabla 3.1: Requisitos funcionales del sistema

Requisitos funcionales: Usuario

La Tabla 3.2 muestra los requisitos funcionales del usuario.

Id	Requisitos
RFU-1	El sistema debe permitir al usuario poder cargar ontologías(1) desde una URL o un archivo del sistema.
RFU-2	El sistema debe permitir al usuario poder inferir sobre la ontología(1) cargada con un razonador(2).
RFU-3	El sistema debe permitir al usuario poder visualizar el esquema de la ontología(1).
RFU-4	El sistema debe permitir al usuario poder aplicar zoom a la visualización de la ontología(1).
RFU-5	El sistema debe permitir al usuario visualizar la jerarquía de conceptos o clases, incluyendo las clases anónimas derivadas de sus definiciones.
RFU-6	El sistema debe permitir la búsqueda y enfoque en clases específicas de la ontología(1).
RFU-7	El sistema debe permitir al usuario poder ocultar parcialmente el grafo.
RFU-8	El sistema debe permitir al usuario cerrar y abrir una clase, ocultando o mostrando sus subniveles según sea necesario.
RFU-9	El sistema debe permitir al usuario ocultar una clase específica.
RFU-10	El sistema debe permitir al usuario mostrar/ocultar toda la información relacionada con las propiedades de todos los nodos.
RFU-11	El sistema debe permitir al usuario visualizar las relaciones disjuntas en el grafo.
RFU-12	El sistema debe permitir al usuario alternar la visibilidad de los rangos de las propiedades en el grafo.
RFU-13	El sistema debe permitir al usuario reorganizar los elementos en pantalla según sus preferencias dentro del mismo nivel.
RFU-14	El sistema debe permitir guardar la vista actual de la ontología(1) como un archivo.
RFU-15	El sistema debe restaurar una vista guardada previamente desde un archivo.
RFU-16	El sistema debe permitir al usuario exportar la vista actual como imagen.
RFU-17	El sistema debe permitir al usuario exportar la vista completa como imagen.
RFU-18	El sistema debe permitir al usuario listar las instancias de las diferentes clases.
RFU-19	El sistema debe permitir al usuario desplegar/ocultar nodos hijos.

Tabla 3.2: Requisitos funcionales del usuario

3.3.2. OntView 2.0

Requisitos funcionales: Sistema

La Tabla 3.3 muestra los requisitos funcionales del sistema.

Id	Requisitos
RFS-1	El sistema debe proporcionar un tooltip(4) con información sobre cada funcionalidad del sistema.
RFS-2	El sistema debe mostrar los nodos equivalentes como un supernodo, siendo esto un recuadro que envuelve a los nodos equivalentes.
RFS-3	El sistema debe poder extraer conceptos mediante extractores(3) como KCE, PageRank o RDFRank.
RFS-4	El sistema debe proporcionar un botón de ayuda que contenga una leyenda que ayude al usuario a navegar por la aplicación.
RFS-5	El sistema debe mostrar al usuario un resumen con la información de la ontología.
RFS-6	El sistema debe mostrar una barra situada encima de un nodo cuando este colapse sus hijos, mostrando la cantidad de nodos hijos que contiene.

Tabla 3.3: Requisitos funcionales del sistema

Requisitos funcionales: Usuario

La Tabla 3.5 muestra los requisitos funcionales del usuario.

Id	Requisitos
RFU-1	El sistema debe permitir al usuario poder habilitar o deshabilitar la visualización de conectores entre nodos.
RFU-2	El sistema debe permitir al usuario seleccionar nodos específicos para mostrar sus conectores.
RFU-3	El sistema debe permitir al usuario mostrar/ocultar las propiedades asociadas a una clase específica.
RFU-4	El sistema debe permitir al usuario eliminar visualmente todos los conectores de los nodos seleccionados al presionar un botón.
RFU-5	El sistema debe permitir al usuario alternar la visibilidad de las relaciones disjuntas en el grafo.
RFU-6	El sistema debe mostrar/ocultar etiquetas y nombres calificados de todos los nodos según el uso de *namespaces*(6) .
RFU-7	El sistema debe permitir al usuario seleccionar el idioma de las etiquetas al mostrarlas. Si la ontología dispone de múltiples idiomas, el usuario podrá elegir uno; de lo contrario, se mostrará por defecto en inglés.
RFU-8	El sistema debe permitir al usuario interactuar con los diversos elementos del grafo.

Tabla 3.4: Requisitos funcionales del usuario

Id	Requisitos
RFU-9	El sistema debe permitir al usuario mostrar tooltips(4) que contenga información de cada nodo o propiedad.
RFU-10	El sistema debe permitir al usuario desplegar/ocultar nodos padre.
RFU-11	El sistema debe mejorar la visualización del grafo de la ontología(1), mejorando la visualización de los nodos y optimizando la claridad de las relaciones y propiedades entre nodos. (Requisito RFU-3 OntView 1.0)
RFU-12	El sistema debe restaurar una vista guardada previamente sin la necesidad de realizar pasos adicionales. (Requisito RFU-15 OntView 1.0)

Tabla 3.5: Requisitos funcionales del usuario

3.3.3. Requisitos no funcionales

La Tabla 3.6 muestra los requisitos no funcionales.

Id	Requisitos
RNF-1	La aplicación no debe perder rendimiento, ni información.
RNF-2	El sistema debe garantizar que las operaciones críticas, como la carga de ontologías(1) y la sincronización con razonadores, se realicen correctamente y sin fallos.
RNF-3	El sistema debe manejar ontologías(1) de tamaño grande sin perder significativamente el rendimiento de la aplicación.
RNF-4	El sistema debe ser compatible con diferentes sistemas operativos.
RNF-5	El sistema debe asegurar que los archivos de ontología(1) cargados desde el sistema de archivos o desde una URL sean validados antes de ser procesados.
RNF-6	El sistema debe manejar eventos de usuario como clics en botones y cambios en checkboxes.
RNF-7	El sistema debe proporcionar un menú contextual con opciones de configuración específicas al hacer clic derecho en un elemento o en el *canvas*(5) del grafo.
RNF-8	El sistema debe separar los nodos cuando estos colisionen entre sí.
RNF-9	El sistema debe poder lanzarse de forma sencilla a través de mecanismos de gestión de dependencias y librerías.

Tabla 3.6: Requisitos no funcionales

3.4. GUI

En la Figura 3.1 se muestra la interfaz gráfica de usuario (GUI) del visualizador con una ontología cargada, así como todas las funcionalidades que el usuario puede seleccionar en la parte superior de la pantalla. Estas funcionalidades incluyen: (1) la carga de ontologías, (2) la selección del razonador y extractor de conceptos, la sincronización, (3) opciones de visualización de propiedades, etiquetas y nombres calificados, guardar y restaurar la vista, tomar capturas de pantalla, búsqueda y ayuda.

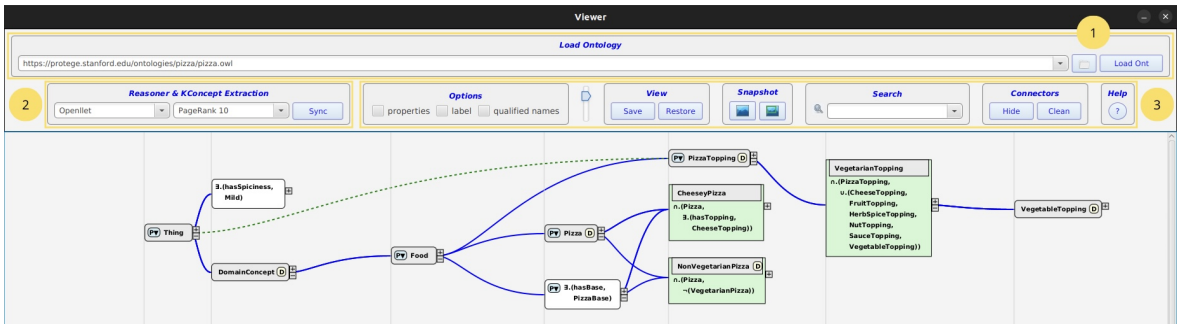

Figura 3.1: Interfaz gráfica de usuario

A continuación, se describen los elementos gráficos del visualizador OntView. Con esta información, se puede comprender cómo se representan visualmente las diferentes clases, relaciones y jerarquías en la ontología, lo que facilitará la interpretación y el análisis del grafo.

Clases primitivas: Estas clases representan conceptos básicos que forman el vocabulario de la ontología. Tienen un nombre visible que se dibuja dentro de un rectángulo de color gris claro (Figura 3.2).

Figura 3.2: Clases primitivas

Clases anónimas: Son conceptos que no tienen un nombre asignado en el esquema y se corresponden con expresiones que representan conjuntos de elementos (Figura 3.3). Estas clases son utilizadas por el razonador para definir relaciones y clasificaciones en la ontología.

Figura 3.3: Clases definidas

Clases definidas: Son conceptos que han sido especificados mediante una expresión, lo que significa que se han establecido condiciones necesarias y suficientes para determinar su pertenencia. Resaltan con un color verde claro y muestra su nombre en la parte superior del rectángulo (Figura 3.4).

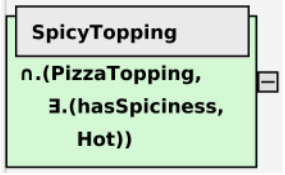

Figura 3.4: Clases definidas

Relaciones disjuntas: Los conceptos que son disjuntos con otros tienen un pequeño indicador en forma de cuadrado con la letra "D" (Figura 3.5). Pulsando en él, se muestra con cuáles lo son. De esta forma se aporta una mayor limpieza visualmente, al no representar todas las conexiones simultáneamente.

Figura 3.5: Relaciones disjuntas

Propiedades de las conceptos: El que un concepto sea el dominio de una o varias propiedades se muestra con un pequeño cuadrado "P" dentro del rectángulo que lo representa (Figura 3.6).

Figura 3.6: Propiedades

Supernodos: Los supernodos representan la combinación de nodos equivalentes y se dibujan como rectángulos grises con un borde azul (Figura 3.7).

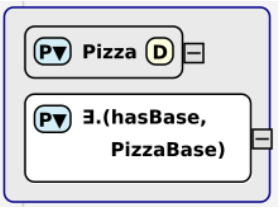

Figura 3.7: Supernodos

Conectores disjuntos Estos conectores se dibujan como un conjunto de dos líneas de color rojo (Figura 3.8) y expresan que los conceptos conectados son disjuntas, es decir, no tienen instancias en común. Se utilizan para reducir la complejidad visual manteniendo la claridad de la relación representada.

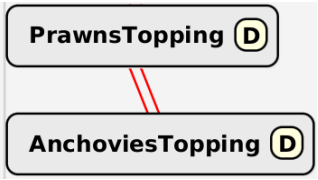

Figura 3.8: Conector disjoint

Niveles del grafo: El grafo se organiza en distintos niveles horizontales que corresponden con la distancia jerárquica máxima de un concepto respecto OWLThing (Figura 3.9). Las líneas verticales grises claras en el fondo del grafo marcan visualmente estos niveles, ayudando a distinguir y estructurar las relaciones jerárquicas entre las clases.

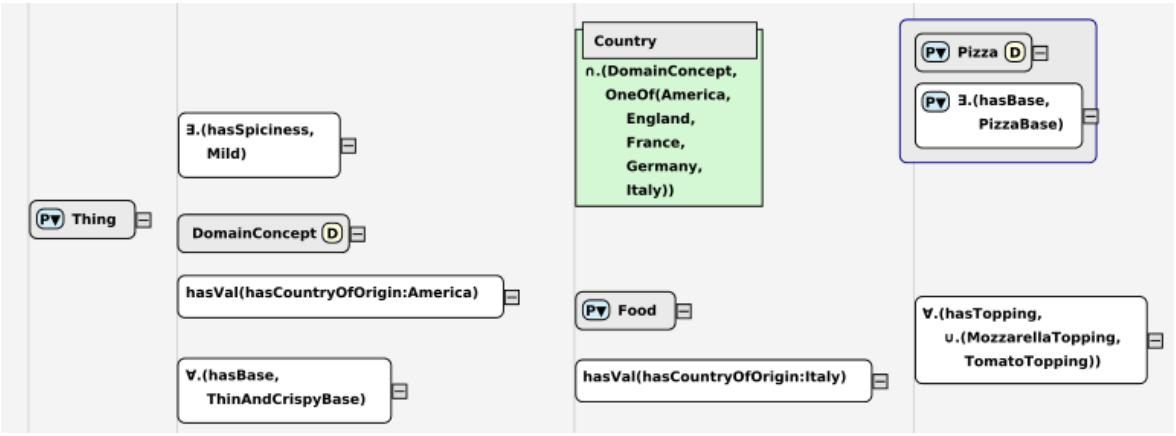

Figura 3.9: Niveles del grafo

Conectores IsA: Estos conectores representan relaciones jerárquicas directas entre conceptos en el grafo. Se dibujan como líneas sólidas, generalmente de color azul, que se vuelven naranjas cuando un nodo específico es seleccionado. Los conectores IsA indican que un concepto es una subclase de otro, estableciendo así una relación "es-un" de derecha a izquierda (Figura 3.10).

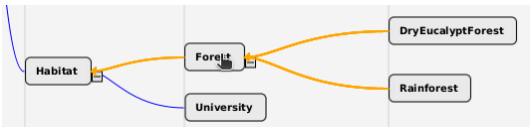

Figura 3.10: Conector Is-A

Conectores discontinuos: Un conector IsA especial, representado por líneas discontinuas, que indica una relación jerárquica indirecta (Figura 3.11). Por ejemplo, si una clase A es superclase de la clase B, y B es superclase de la clase C, un conector discontinuo representa la relación indirecta entre A y C. Estos conectores son especialmente útiles cuando se ocultan nodos intermedios, ya que aseguran que la relación visual entre los nodos principales se preserve.

Figura 3.11: Conector discontinuos

Conectores de rango: Los conectores de rango van desde una propiedad hasta un concepto, indicando el rango de la propiedad, es decir, el tipo de valor que puede tomar (Figura 3.12). E.g,. *hasBase* tiene como rango *PizzaBase*.

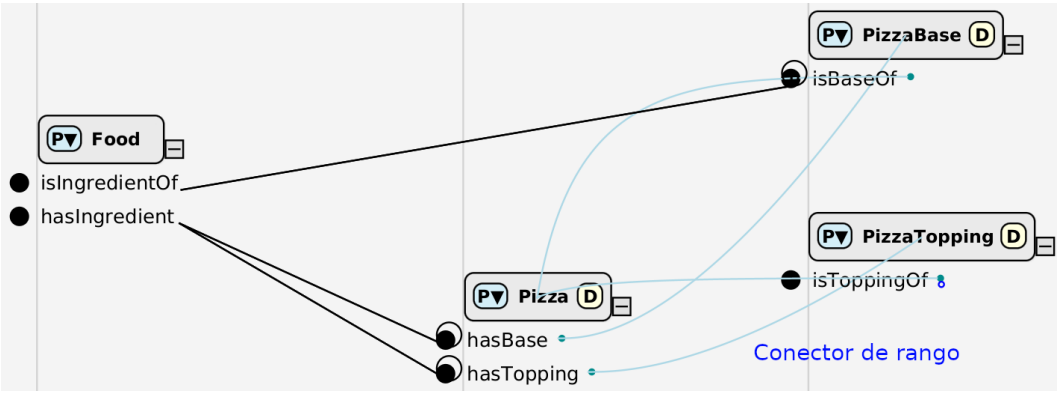

Figura 3.12: Conector de rango

Conectores de herencia Los conectores de herencia se utilizan para mostrar la relación entre propiedades, donde una propiedad hereda de otra. Se representan con una recta que conecta la subpropiedad con la propiedad padre de derecha a izquierda (Figura 3.13).

Cuando una propiedad es subpropiedad de más de una propiedad, se utilizan conectores de herencia múltiple para mostrar esta relación compleja, manteniendo la integridad y claridad del grafo ontológico (Figura 3.13).

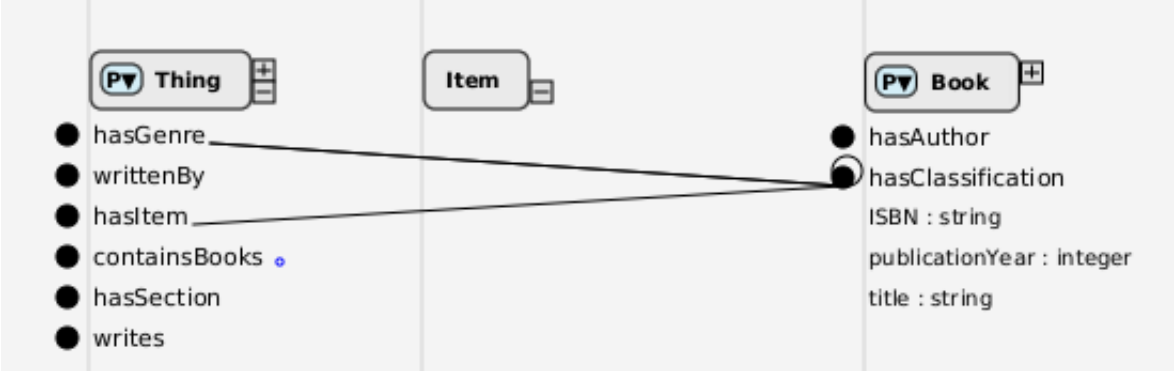

Figura 3.13: Conector de herencia

Una vez comprendidos los elementos gráficos que conforman el grafo, se pueden utilizar de manera más efectiva las funcionalidades del sistema. Para conocerlas mejor, se puede consultar el Anexo B, donde se explican en detalle cada una de las opciones disponibles en la interfaz, siendo estas las siguientes:

1. **Carga de ontologías:** Los usuarios pueden cargar ontologías desde una URL o un archivo del sistema, con la posibilidad de acceder a ontologías recientemente utilizadas a través de un desplegable (RFS-1 y RFU-1 de OntView 1.0).

2. **Selección del razonador y KCE:** Permite la selección de un razonador y la aplicación de algoritmos de extracción de conceptos clave para inferir relaciones adicionales dentro de la ontología (RFS-2 y RFU-2 de OntView 1.0) (RFS-3 de OntView 2.0).

3. **Opciones de visualización:** Los usuarios pueden mostrar u ocultar propiedades, etiquetas y nombres calificativos en el grafo, facilitando el análisis detallado o la reducción del ruido visual según sea necesario (RFU-10 de OntView 1.0) (RFU-7 de OntView 2.0).

4. **Zoom interactivo:** Proporciona control sobre el nivel de detalle visual en el grafo, permitiendo a los usuarios acercar o alejar la vista según sus necesidades (RFU-4 de OntView 1.0).

5. **Guardar y restaurar vista:** Los usuarios pueden guardar la configuración actual del grafo en un archivo XML y restaurarla posteriormente, asegurando la continuidad del trabajo y la organización visual del grafo (RFU-14 y RFU-15 de OntView 1.0) (RFU-13 de OntView 2.0).

6. **Capturas de pantalla:** Permite la captura del grafo completo o de la parte visible en pantalla, facilitando la documentación y el análisis externo (RFU-16 y RFU-17 de OntView 1.0).

7. **Búsqueda en el grafo:** Facilita la localización de elementos específicos dentro de la ontología por su nombre, centrando la vista en el nodo buscado (RFU-6 de OntView 1.0).

8. **Mostrar/ocultar y limpiar conectores:** Ofrece la posibilidad de gestionar la visibilidad de los conectores en el grafo, incluyendo la funcionalidad para mostrar u ocultar todos los conectores y limpiar los conectores seleccionados por el usuario (RFU-1, RFU-2 y RFU-5 de OntView 2.0).

9. **Botón de ayuda:** Proporciona información adicional para guiar al usuario en el uso de la aplicación (RFS-4 de OntView 2.0).

10. **Repulsión entre nodos:** Un algoritmo implementado evita la superposición de nodos al redistribuirse automáticamente cuando ocurre una colisión (RNF-8 de OntView 2.0).

11. **Mover un nodo del grafo:** Los usuarios pueden desplazar un nodo dentro de su nivel en la jerarquía del grafo, asegurando una disposición ordenada y evitando la superposición de nodos mediante un mecanismo de repulsión (RFU-13 OntView 1.0).

12. **Centrar nodo:** Permite centrar la vista en un nodo específico del grafo, resaltando los conectores asociados para una mejor comprensión de sus relaciones (RFU-6 de OntView 1.0).

13. **Mostrar/ocultar nodo:** Los usuarios pueden mostrar u ocultar nodos hijos, según la estructura del grafo (RFU-8 y RFU-19 de OntView 1.0) (RFU-10 de OntView 2.0).

14. **Tooltip con la información extra de la clase:** Permite visualizar información adicional de una clase al mantener el cursor sobre ella durante un tiempo determinado (RFU-9 de OntView 2.0).

15. **Menú contextual:** Al hacer clic derecho en un nodo o en el *canvas*[5], se despliegan opciones específicas que permiten gestionar nodos individuales o el grafo completo, incluyendo la visualización de instancias y propiedades, así como la gestión de relaciones disjuntas y rangos (RNF-7 de OntView 2.0).

Una vez analizado tanto los elementos gráficos, como las funcionalidades, se ha realizado una breve comparación entre el OntView 1.0 y el OntView 2.0 en el Anexo C.

3.5. Diagramas

En esta sección, se presentan los diagramas utilizados para describir la estructura y el comportamiento del sistema OntView.

3.5.1. Diagrama de clases

A continuación, en la Figura 3.14, se presenta el diagrama de clases que detalla las principales clases del sistema y sus relaciones. Se han omitido las clases que únicamente implementan código relacionado con la interfaz gráfica (GUI) para centrar la atención en el núcleo de la aplicación. Además, para proporcionar una vista rápida y clara del sistema, no se incluyen los nombres de todas los métodos debido a su gran cantidad.

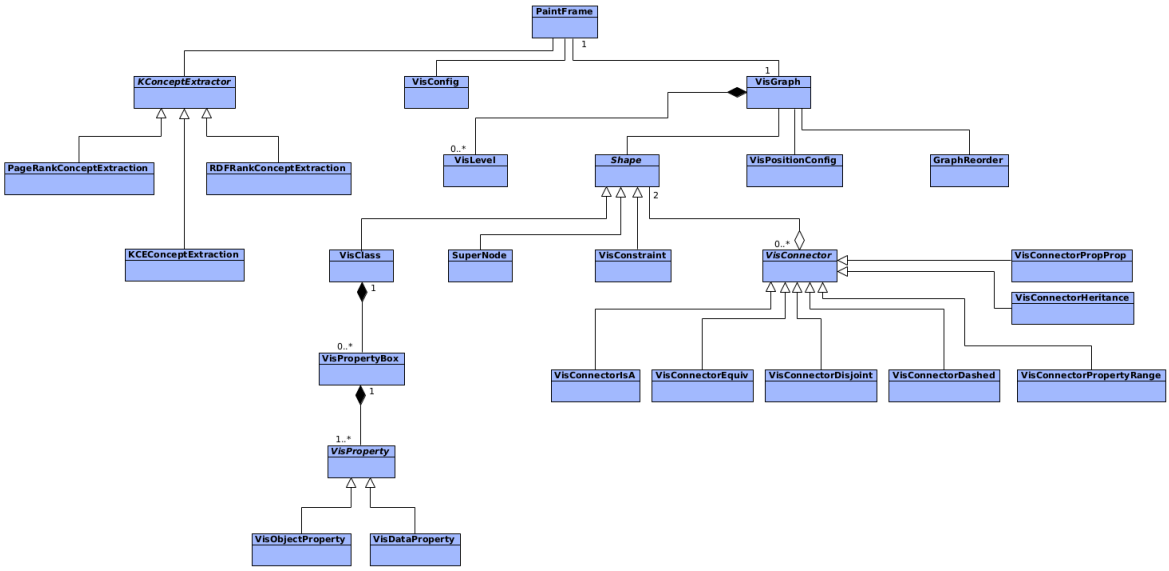

Figura 3.14: Diagrama de clases

Las clases más importantes a destacar son **VisClass** y **VisGraph**. La clase VisClass se encarga de representar visualmente una clase con una OWLClassExpression y sus relaciones. La clase VisGraph construye y gestiona el grafo visual de una ontología OWL.

3.5.2. Diagrama de paquetes

El diagrama de clases presentado anteriormente se centra en las clases y relaciones dentro del paquete "common", que es un componente clave del sistema. Como se muestra en la Figura 3.15, además de "common", el sistema se organiza en otros paquetes como "main", "expressionNaming", y "utils", los cuales agrupan diversos algoritmos y funcionalidades, tales como la obtención de resúmenes y la identificación de clases anónimas. Esta estructuración en paquetes asegura una mayor modularidad del sistema.

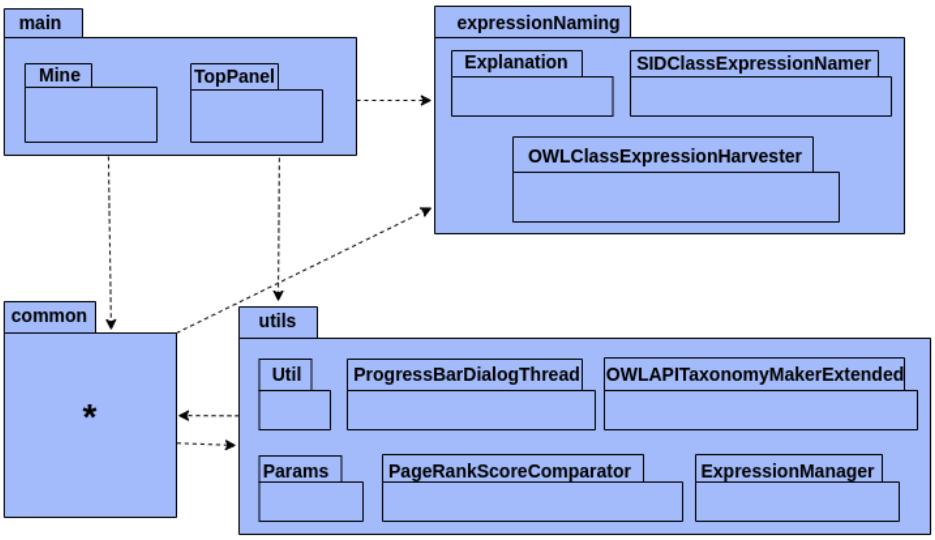

Figura 3.15: Diagrama de paquetes

* El paquete "common" corresponde con todas las clases de la información del grafo y los métodos de visualización. Estas clases no se han incluido en el diagrama de paquetes para evitar un diagrama excesivamente grande y redundante, ya que dichas clases están detalladas en el diagrama de clase (Figura 3.14).

3.5.3. Patrón MVC

OntView sigue el patrón de arquitectura Modelo-Vista-Controlador (MVC), una estructura ampliamente utilizada en el desarrollo de aplicaciones de escritorio. En este caso, la *Vista* es el gestor de escritorio, encargándose de gestionar la interfaz gráfica y la representación visual del grafo ontológico. El *Controlador* maneja todas las acciones necesarias para responder a los eventos generados por el usuario, como los clicks, la navegación o la manipulación de los nodos. Finalmente, el *Modelo* se centra en la carga y gestión de la ontología, proporcionando los datos necesarios para que la Vista pueda representar la información correctamente. Esta separación de responsabilidades permite que OntView mantenga una estructura modular y escalable, alineándose con el patrón MVC y garantizando una clara división entre la lógica de la interfaz, la gestión de eventos y la manipulación de los datos.

3.5.4. Diagrama de secuencia

El diagrama de secuencia que se presenta a continuación describe los pasos que un usuario debe seguir para interactuar con OntView y cómo el sistema procesa estas acciones. Este diagrama guía a través de las interacciones clave, desde la carga de una ontología hasta la visualización y manipulación de los datos en el grafo mostrado en la Figura 3.1.

Para estructurar y clarificar el flujo de interacciones en el visualizador OntView, el diagrama de secuencias se ha dividido en tres niveles. El **nivel 0**, siendo el de más alto nivel, ofrece una visión global del proceso, desde la carga de la ontología hasta la ejecución de las diversas opciones de visualización. El **nivel 1** profundiza en los pasos de carga de ontologías (1) y selección del razonador y extractor de conceptos, junto con la sincronización (2), especificando cómo el usuario introduce la ontología y configura las herramientas para su razonamiento. Finalmente, el **nivel 2** detalla las opciones de visualización (3), abarcando funciones como mostrar propiedades, etiquetas, nombres calificados, guardar y restaurar vistas del grafo, descargar imágenes, buscar nodos, y gestionar conectores.

Figura 3.16: Diagramas de secuencia

4. Proceso de migración

El proceso de migración de OntView a su versión 2.0 implicó la transición de JavaSwing a JavaFX. Esta actualización fue necesaria para mejorar la interfaz gráfica y el rendimiento de la aplicación. En este capítulo se describen los pasos realizados, desde la evaluación de las versiones existentes hasta la implementación de nuevas funcionalidades y la resolución de problemas encontrados durante la migración.

4.1. Estrategia de migración

El desarrollo del visualizador OntView siguió varios pasos clave para lograr una aplicación funcional y eficiente. A continuación, se detalla el proceso seguido:

1. **Inmersión en el código de las dos versiones:** El primer paso fue analizar en profundidad el código de las dos versiones existentes del visualizador OntView. Este análisis permitió comprender las bases y diferencias entre ambas versiones, identificando sus fortalezas y debilidades. Se revisó la estructura del código, las funcionalidades implementadas y la tecnología utilizada en cada versión.

2. **Decisión de la versión a migrar:** Tras esta inmersión, se tomó la decisión de cuál versión migrar. Se optó por la versión más antigua debido a su mayor facilidad para la actualización tecnológica y su compatibilidad con JavaFX. Aunque la versión más moderna presentaba algunas mejoras, los beneficios de una migración más sencilla y la estabilidad del código antiguo fueron factores determinantes.

3. **Decisión de la tecnología:** El siguiente paso fue decidir la tecnología a utilizar. Se evaluaron diversas opciones, considerando su capacidad para mejorar el rendimiento y la experiencia del usuario. Finalmente, se seleccionó JavaFX debido a sus avanzadas capacidades gráficas y su integración con Java, además de explorar la posibilidad de usar JavaScript y Canvas de HTML5 para futuras mejoras. Se explica más en detalle, junto con las pruebas de rendimiento, en el Anexo D.

4. **Estudio de la tecnología:** Con la tecnología seleccionada, se llevó a cabo un estudio detallado de JavaFX. Este estudio incluyó la comprensión de sus características, ventajas y la forma en que podría integrarse con el código existente. Se realizaron pruebas preliminares para familiarizarse con el entorno y las herramientas de desarrollo de JavaFX.

5. **Migración completa a JavaFX:** La fase de migración completa a JavaFX comenzó con la conversión de la interfaz de usuario de JavaSwing a JavaFX. Inicialmente, se intentó mantener ambas tecnologías operando simultáneamente, pero esto generó problemas de compatibilidad. Por lo tanto, se decidió realizar una migración completa a JavaFX, eliminando completamente la dependencia de JavaSwing y asegurando una base tecnológica unificada.

6. **Comprobación de cada funcionalidad:** Una vez realizada la migración, se procedió a la comprobación de cada funcionalidad del visualizador. Se verificó que todas las funcionalidades originales estuvieran correctamente implementadas en la nueva plataforma y que el sistema operara de manera estable y eficiente. Esta fase incluyó pruebas exhaustivas y la identificación de posibles fallos o áreas de mejora.

7. **Mejora de ciertas funcionalidades y adición de nuevas:** Finalmente, se mejoraron ciertas funcionalidades y se añadieron nuevas características, basadas en el *feedback* de las pruebas y las necesidades identificadas durante el desarrollo. Se llevó a cabo un *testing* visual constante para asegurar que cada funcionalidad nueva y mejorada funcionara correctamente, manteniendo la integridad del sistema y mejorando la experiencia del usuario.

4.2. Elección de tecnologías

En el proceso de actualización de OntView, fue esencial seleccionar la tecnología adecuada para asegurar una migración eficiente y mejorar el rendimiento del sistema. Se evaluaron varias opciones tecnológicas, cada una con diferentes características, con el objetivo de identificar la más adecuada para el desarrollo del visualizador.

JavaFX se consideró una de las primeras opciones debido a sus capacidades gráficas y su integración con Java, lo que facilitaba la continuidad con el código existente. También se exploraron alternativas como JavaScript en combinación con Java y HTML5 Canvas junto con JavaScript y Java, buscando mejorar la eficiencia y flexibilidad del sistema.

Para tomar una decisión basada en datos, se realizaron pruebas de rendimiento que midieron el tiempo de ejecución y la capacidad de respuesta del sistema al manejar diferentes cantidades de elementos gráficos. Estas pruebas evaluaron el desempeño de cada tecnología en pantalla completa y en resoluciones más pequeñas. Los detalles técnicos y los resultados específicos de estas pruebas se encuentran en el Anexo D.

Finalmente, se optó por realizar una migración completa a JavaFX, debido a su buen rendimiento y facilidad de integración con el código existente. Sin embargo, se consideró el uso de HTML5 Canvas con JavaScript como una posible mejora futura para la visualización del grafo, por su eficiencia en la gestión gráfica.

4.3. Gestión del Proyecto

La planificación y organización del proyecto se basó en la implementación de metodologías ágiles, con herramientas específicas como tableros Kanban y diagramas de Gantt para el seguimiento y control del progreso.

4.3.1. Tableros Kanban

La adopción de metodologías ágiles jugó un papel clave en el desarrollo del proyecto, ayudando a manejar de forma efectiva la complejidad y los cambios constantes que surgieron durante la migración tecnológica. El uso de un tablero Kanban [32] fue especialmente útil para mantener un control claro y ordenado de cada etapa del proceso.

Inicio de la migración

Para aplicar esta metodología ágil durante la migración, se estructuró el proceso a través de un sistema Kanban, que permite monitorizar de manera clara y sistemática el estado de cada fichero durante el proceso de migración. Este Kanban se ha dividido en cinco columnas: "Ficheros a migrar", "En proceso", "Bloqueados", "Finalizados" y "En dudas".

1. **"Ficheros a migrar"**: agrupa todos los ficheros que están programados para ser transferidos. Esta organización inicial es crucial, ya que proporciona una visión global del volumen de trabajo y permite priorizar las tareas de acuerdo a la urgencia y la importancia de los ficheros.

2. **"En proceso"**: está dedicada a aquellos ficheros que actualmente se encuentran en medio de su migración.

3. **"Bloqueados"**: se colocan los ficheros cuya migración depende de la transferencia previa de otros archivos. Este aspecto del sistema es especialmente útil para gestionar dependencias y asegurar que se mantenga una secuencia lógica y eficiente en el proceso de migración, evitando cuellos de botella.

4. **"Finalizados"**: representa aquellos ficheros que han sido migrados con éxito. El seguimiento de estos archivos permite confirmar que se han alcanzado los objetivos parciales y que los ficheros están ahora operativos en el nuevo entorno.

5. **"En dudas"**: incluye los ficheros que, aunque migrados, podrían presentar problemas si surgieran fallos en la aplicación una vez completada la migración total. Esta columna es esencial para la fase de pruebas y ajustes post-migración, ya que facilita la identificación rápida de los elementos que más atención requieren y permite concentrar los esfuerzos de solución en los puntos críticos detectados.

Verificación Funcional

Una vez completada la migración inicial de los ficheros, el siguiente paso en el proceso es la verificación funcional. Este proceso consiste en comprobar, funcionalidad por funcionalidad, que el sistema migrado responde correctamente. Para gestionar esta fase, se utiliza una tabla adicional que incluye descripciones detalladas de cada funcionalidad, etiquetas de importancia y si existen etiquetas de errores o *bugs*.

La tabla de verificación funcional se organiza de tal manera que cada funcionalidad se evalúa individualmente, asegurando que todas las partes del sistema operan como se espera. Las descripciones proporcionan un resumen claro de cada funcionalidad, mientras que las etiquetas de importancia ayudan a priorizar las pruebas y correcciones. Las etiquetas de errores identifican cualquier problema detectado, permitiendo que se aborden de manera rápida y eficiente.

Más tarde, se decidió trasladar todas las "tarjetas"del nuevo tablero Kanban a *issues* en GitHub[24], donde también se aloja todo el código de la aplicación. Este enfoque centraliza la gestión del proyecto en una única plataforma, mejorando la organización y facilitando la comunicación con el grupo SID. Además, dentro de GitHub, los *issues* se asociaron a un tablero Kanban, lo que permite una gestión más eficiente de las tareas y un seguimiento claro del progreso del proyecto.

4.4. Problemas de la migración

El primer paso del proyecto consistió en decidir cuál de las dos versiones se iba a actualizar. Esto llevó al primer problema encontrado: cómo lanzar ambas versiones. Debido a la falta de documentación de una de las versiones, se enfrentaron diversos desafíos. Para la versión más moderna, se recuperó el archivo .war para poder ejecutarlo, mientras que para la versión más antigua, se logró lanzarla mediante Eclipse. Cabe destacar que, debido a la antigüedad de la primera versión, muchas librerías estaban obsoletas y algunas incluso se habían perdido. Fue necesario actualizar dichas librerías, asegurándose de que las nuevas versiones no presentaran incompatibilidades entre sí. Esto implicó que, en ese momento del proyecto, las librerías no fueran las más actuales debido a las incompatibilidades entre ellas. Más adelante, se actualizaron todas las librerías. Tras un análisis exhaustivo de ambas versiones, se decidió continuar con la versión original del visualizador.

Al comenzar la migración del programa, se tenía en mente realizar una migración parcial, pudiendo lanzar parte en JavaFX y otra en JavaSwing. Sin embargo, esto no fue posible debido a la incompatibilidad entre ambas tecnologías, ya que cada una se lanza en hilos diferentes y para el correcto funcionamiento del programa, todo debe lanzarse en el mismo hilo principal, es decir, en JavaFX o JavaSwing. Dado este impedimento técnico, se optó por un enfoque alternativo, decidiendo llevar a cabo una migración completa hacia JavaFX en la primera instancia.

Una vez en funcionamiento la aplicación, los siguientes pasos fueron comprobar funcionalidad por funcionalidad su correcto funcionamiento, lo que llevó a la corrección de los siguientes problemas provenientes de la versión inicial:

1. **Curva de Aprendizaje de JavaFX:** La transición de JavaSwing a JavaFX implicó un período de adaptación debido a las diferencias en la arquitectura y la gestión de interfaces gráficas. Aunque JavaFX ofrece características modernas y flexibles, requirió familiarización con nuevas técnicas para el manejo de eventos, animaciones y vinculación de datos. El proceso de aprendizaje se llevó a cabo mediante la formación y pruebas con JavaFX, lo que facilitó la integración de las nuevas herramientas en el proyecto.

2. **Optimización de rendimiento:** Se detectaron ineficiencias en la frecuencia de redibujado del grafo y en el manejo de colisiones entre nodos, lo que afectaba la *performance* general de la aplicación. Para solucionar esto, se implementaron mejoras en los algoritmos de redibujado y gestión de colisiones, optimizando el rendimiento y reduciendo la carga innecesaria en el sistema.

3. **Mejora de la detección de colisiones:** El algoritmo de detección de colisiones inicial era ineficiente y a veces inexacto, resultando en superposiciones de nodos en el grafo que dificultaban la claridad visual y la usabilidad. Se mejoró tanto el algoritmo de detección como el proceso de separación de nodos al producirse una colisión, ajustando la lógica de cálculo de repulsión y asegurando que se respetara la distancia mínima entre nodos, teniendo en cuenta también los niveles de jerarquía para evitar repulsiones innecesarias.

4. **Mejora de la usabilidad:** Varias funcionalidades relacionadas con la interacción del usuario, como la visualización de *tooltips* y la gestión de menús, presentaban problemas que afectaban la experiencia del usuario. Para mejorar la usabilidad, se corrigieron estos comportamientos, asegurando que el usuario tuviera una experiencia más coherente y fluida al interactuar con la aplicación.

5. **Corrección de funcionalidades existentes:** La migración reveló varios errores heredados de la versión original que afectaban la correcta visualización y manipulación de elementos en el grafo. Se realizaron ajustes y correcciones en la lógica de visualización y manipulación de nodos y propiedades, asegurando que las funcionalidades trabajen como se espera en el entorno migrado.

5. Conclusiones y trabajo a futuro

La migración de OntView de JavaSwing a JavaFX ha mejorado tanto la interfaz de usuario como la eficiencia del sistema. Estos cambios refuerzan la utilidad de OntView en la visualización y exploración de ontologías. A partir de estos avances, se identifican áreas para seguir desarrollando el visualizador y adaptarlo a las futuras necesidades del campo.

5.1. Conclusiones técnicas

El desarrollo de OntView 2.0 ha estado guiado por tres premisas fundamentales: mejorar la visualización de las ontologías para facilitar su comprensión, actualizar y modernizar la herramienta aprovechando nuevas tecnologías, y garantizar la escalabilidad para manejar tanto ontologías pequeñas como grandes. En cuanto a la visualización, se ha conseguido una representación gráfica mucho más clara y precisa que en versiones anteriores, permitiendo al usuario comprender de manera más intuitiva la estructura formal de las ontologías con solo observar la interfaz. La migración de JavaSwing a JavaFX ha sido un aspecto clave en la modernización de la herramienta, proporcionando una mejora significativa en el rendimiento y en la experiencia de usuario. Sin embargo, en lo que respecta al manejo de ontologías grandes, aunque OntView 2.0 ha logrado avances importantes frente a la versión 1.0, especialmente en términos de optimización de la visualización y manipulación de grandes volúmenes de datos, aún es necesario seguir trabajando en este aspecto para asegurar una experiencia completamente fluida en todos los escenarios.

Durante el proceso de desarrollo de OntView 2.0, se logró mantener las funcionalidades de la versión original, lo que permitió asegurar la continuidad del sistema sin perder las características clave de la primera versión. Al mismo tiempo, se trabajó en la mejora de aspectos importantes, como la manipulación de nodos, que ahora ofrece una interacción más dinámica y flexible, permitiendo al usuario ajustar y reorganizar los nodos según sus necesidades. También se mejoró la visualización de las relaciones y propiedades entre conceptos, lo que facilita una comprensión más clara de las ontologías, especialmente en aquellas con una estructura más compleja. La experiencia del usuario también evolucionó, con una interfaz gráfica más moderna e intuitiva que responde mejor a las expectativas de quienes trabajan con grandes volúmenes de datos.

A lo largo del proceso, surgieron desafíos, particularmente en la gestión de la compatibilidad entre tecnologías. La transición de JavaSwing a JavaFX supuso un reto, ya que implicaba adaptar el código a una nueva plataforma y asegurar que las funcionalidades heredadas funcionaran correctamente en el nuevo entorno. Esto requirió un esfuerzo

considerable para identificar y corregir *bugs* del código heredado. La detección y solución de estos problemas fue clave para garantizar la estabilidad y el rendimiento del nuevo sistema.

Aunque surgieron obstáculos, el proyecto ha logrado alcanzar sus objetivos principales, modernizando la herramienta, ampliando sus capacidades y mejorando su rendimiento. El uso de nuevas tecnologías ha proporcionado una base sólida para futuras actualizaciones. Sin embargo, el manejo de ontologías de gran tamaño aún presenta oportunidades de mejora, marcándose como un trabajo a futuro para optimizar aún más la experiencia del usuario y la eficiencia del sistema.

Cabe destacar que el proyecto se ha dejado en un estado que permite a cualquier persona continuarlo sin dificultad, gracias al uso del tablero Kanban mencionado anteriormente. Este tablero ha facilitado la documentación detallada de las mejoras pendientes y los bugs por resolver, asegurando que el trabajo pueda retomarse de manera clara y organizada.

5.2. Diagrama de Gantt

El diagrama de Gantt refleja el tiempo dedicado a diversas fases del proyecto, comenzando por la documentación. Esta fase incluyó el estudio del documento original de OntView 1.0, un análisis independiente de las tecnologías valoradas para la migración y una profundización en la tecnología finalmente seleccionada. Además, se realizó un estudio detallado sobre la Web Semántica, las ontologías y todos los temas relacionados, que proporcionaron el marco teórico necesario para el desarrollo. Durante la inmersión en el código, se fue comprendiendo cómo funcionaba el sistema original y la versión más actual de OntView, para poder decidir cuál de las versiones se iba a migrar. En cuanto a las pruebas de rendimiento, el tiempo invertido incluye los test realizados para evaluar las tecnologías a migrar y la selección de los argumentos que optimizarán el rendimiento del sistema. La fase de migración, por su parte, abarca el tiempo dedicado a actualizar la aplicación a la nueva tecnología, así como la implementación de nuevas funcionalidades. Las reuniones con los directores reflejan el tiempo utilizado para resolver dudas surgidas durante el proceso, establecer los límites del proyecto y revisar los avances de la memoria. Por último se encuentra el tiempo invertido en la redacción de la memoria.

EL diagrama de Gantt se encuentra en la Figura 5.1.

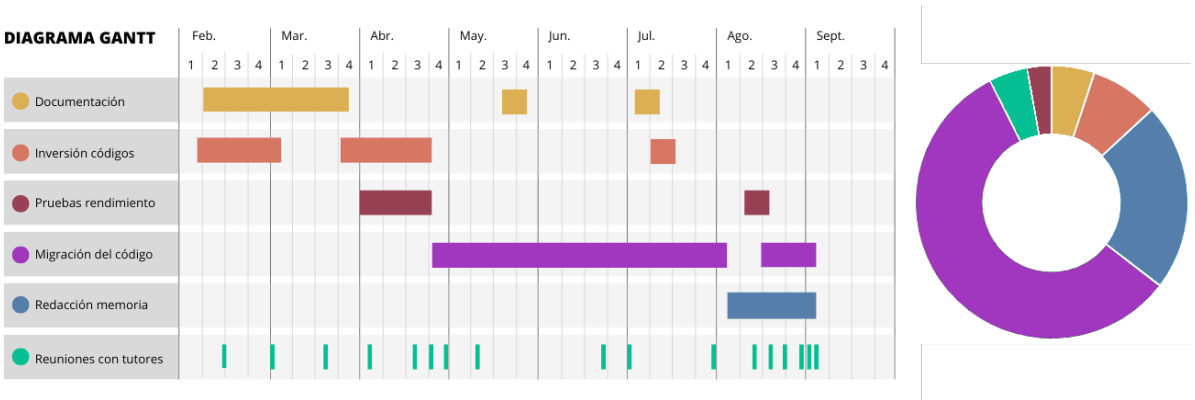

Figura 5.1: Diagrama de gantt y distribución de horas

La Tabla 5.1 muestra el tiempo invertido en horas para cada una de las tareas. La disposición de la tareas se encuentran en el mismo orden que en el diagrama de Gantt.

Tarea	Horas invertidas
Documentación	28h
Inversión de ambos códigos	35h
Pruebas de rendimiento	9h
Migración del código	183h
Redacción de la memoria	83h
Reuniones con los directores	18h
	Horas totales: 356h

Tabla 5.1: Dedicación en horas al proyecto

5.3. Posibles ampliaciones

Al ser un proyecto vivo, hemos detectado y anotado una serie de posibles ampliaciones que se plantean como trabajo futuro:

- **Mejora en la visualización de grandes ontologías:** El campo de las ontologías está en constante evolución, con ontologías cada vez más grandes y complejas que requieren herramientas capaces de manejarlas de manera eficiente. Aunque OntView gestiona adecuadamente ontologías de tamaño mediano, manejar ontologías a gran escala puede resultar complicado debido a la complejidad y el volumen de datos. Una posible mejora sería optimizar la visualización para ontologías grandes, mediante técnicas de renderizado progresivo o la implementación de un sistema de niveles que permita visualizar distintas partes de la ontología sin necesidad de cargar todo el grafo a la vez. Esto también podría incluir la capacidad de realizar zoom semántico, donde el nivel de detalle mostrado en el grafo varíe según el nivel de zoom aplicado.

- **Mostrar el grafo asertado:** OntView actualmente muestra el grafo inferido, que incluye tanto las relaciones asertadas como las deducidas automáticamente por un razonador semántico. Sin embargo, aún falta la opción de visualizar exclusivamente el grafo asertado, es decir, las relaciones y propiedades definidas directamente por el usuario. Implementar esta funcionalidad permitiría a los usuarios comparar fácilmente el conocimiento asertado con el inferido, mejorando así el análisis y la comprensión de la ontología.

- **Estudio con usuarios:** Una de las próximas etapas importantes para OntView es la realización de un estudio con usuarios que permita evaluar de manera objetiva los beneficios y la usabilidad de la herramienta. Este estudio tiene como objetivo involucrar a expertos en el manejo de ontologías, así como a usuarios con diferentes niveles de experiencia, para identificar las fortalezas del visualizador y áreas de mejora.

- **Publicación en foro internacional:** La evolución de OntView 2.0 marca una mejora significativa en la visualización y manejo de ontologías. A pesar del tiempo pasado desde la versión original, y a pesar del avance en herramientas tecnológicas, sigue sin existir

una solución que supere conceptualmente a OntView 1.0 en cuanto a visualización semántica de estructuras ontológicas complejas. Con este contexto, se está preparando una publicación que será presentada en una conferencia internacional especializada en tecnologías de la Web Semántica.

- **Distribución como software abierto y libre:** El proyecto se hará público bajo una licencia abierta tras su publicación en la conferencia. Al optar por un modelo de código abierto, buscamos promover la colaboración y facilitar el acceso a OntView 2.0 dentro de la comunidad académica y profesional. Esto permitirá que otros investigadores y desarrolladores contribuyan activamente a su evolución.
- **Plugin de Protégé:** El desarrollo de un *plugin* para Protégé, una de las herramientas más utilizadas para la edición de ontologías, permitiría a los usuarios visualizar las ontologías con OntView mientras siguen utilizando Protégé para editarlas. Esta integración combinaría la capacidad de visualización avanzada de OntView con las funcionalidades de edición de Protégé.

5.4. Opinión personal

Durante mi paso por Ingeniería Informática, pocas veces había tenido la oportunidad de enfrentarme a proyectos desarrollados por terceros. Sin embargo, trabajar en este proyecto me ha permitido aprender cómo abordar el código creado por otras personas. Esta experiencia me ha ayudado a ampliar mi conocimiento sobre ontologías en un entorno controlado, permitiéndome profundizar en el estado del arte y adquirir una comprensión más sólida del mismo

A lo largo de la migración de la aplicación, he comprendido la complejidad de trabajar con un proyecto desarrollado por otra persona. Fue un desafío entender la lógica del autor original y mejorar su trabajo sin introducir nuevos errores. Este proceso generó dudas sobre la procedencia de los errores, lo que complicó la tarea de distinguir entre problemas heredados y aquellos causados por las nuevas implementaciones.

Además, estimar el tiempo necesario para completar ciertas tareas resultó ser más difícil de lo esperado. Lo que inicialmente parecía sencillo a menudo requería muchas más horas de trabajo, lo que me enseñó a ser más realista y flexible en la planificación.

Finalmente, trabajar en un proyecto en constante evolución me ha hecho comprender la importancia de no obsesionarse con el perfeccionismo, ya que a veces es mejor algo bueno y a tiempo que algo perfecto que nunca llega. Siempre hay formas de mejorar, pero el tiempo es limitado. Este proyecto me ha enseñado a buscar siempre la mejor solución posible, dentro de los límites razonables, debido al tiempo limitado que abarca un Trabajo de Fin de Grado.

6. Bibliografía

[1] Ltd John Wiley Sons. *Semantic Web Technologies: Trends and Research in Ontology-based Systems*. John Wiley & Sons, 2006.

[2] Daniele Nardi Peter F. Patel-Schneider Franz Baader, Deborah L. McGuinness. *THE DESCRIPTION LOGIC HANDBOOK: Theory, implementation, and applications*. Cambridge University Press, 2007.

[3] Jorge Bobed Lisbona. Visualizador de ontologías basado en un razonador de Lógica Descriptiva. Trabajo fin de máster, Universidad de Zaragoza, 08-2012.

[4] Eclipse Fundation. Eclipse. `https://www.eclipse.org/`. 06-09-2024.

[5] Thomas R. Gruber. *A Translation Approach to Portable Ontology Specifications*. Knowledge Acquisition, 1993.

[6] W3C. Lenguaje de Ontologías Web (OWL). Vista General. `https://www.w3.org/TR/2004/REC-owl-features-20040210/`. 06-09-2024.

[7] W3C. OWL. `https://www.w3.org/2001/sw/Europe/events/200406-esp/trabajo-final-extratesauros/node5.html`. 06-09-2024.

[8] Grupo de investigación en la Universidad de Manchester. The OWL API. `https://owlapi.sourceforge.net/`. 06-09-2024.

[9] Openllet. `https://github.com/Galigator/openllet`. 06-09-2024.

[10] FaCT++. `http://owl.man.ac.uk/factplusplus/`. 06-09-2024.

[11] HermiT OWL Reasoner. `http://www.hermit-reasoner.com/`. 06-09-2024.

[12] Pellet: An Open Source OWL DL reasoner for Java. `https://github.com/stardog-union/pellet`. 06-09-2024.

[13] Racer. `https://www.ifis.uni-luebeck.de/~moeller/racer/`. 06-09-2024.

[14] Silvio Peroni, Enrico Motta, and Mathieu d'Aquin. Identifying Key Concepts in an Ontology, through the Integration of Cognitive Principles with Statistical and Topological Measures. In *The Semantic Web (ASWC 2008)*, pages 242–256. Springer, 2008.

[15] David F. Gleich. PageRank Beyond the Web. `https://epubs.siam.org/doi/10.1137/140976649`. 06-09-2024.

[16] GraphDB. RDF Rank. https://graphdb.ontotext.com/documentation/10.0/rdf-rank.html. 06-09-2024.

[17] Wikipedia. Identificador de recursos uniforme. https://es.wikipedia.org/wiki/Identificador_de_recursos_uniforme. 06-09-2024.

[18] Nodos en blanco y valores literales. https://semantizandolaweb.wordpress.com/2013/03/16/nodos-en-blanco-y-valores-literales/. 06-09-2024.

[19] JetBrains IDEs. IntelliJ ADEA. https://www.jetbrains.com/idea/. 06-09-2024.

[20] JavaFX. https://openjfx.io/. 06-09-2024.

[21] Oracle. Java. https://www.java.com/es/. 06-09-2024.

[22] draw.io. https://app.diagrams.net/. 06-09-2024.

[23] Atlassian. Trello. https://trello.com/es. 06-09-2024.

[24] GitHub. https://github.com/. 06-09-2024.

[25] Stanford Center for Biomedical Informatics Research. Protégé. https://protege.stanford.edu/. 06-09-2024.

[26] Matthew Horridge. OWLViz. https://protegewiki.stanford.edu/wiki/OWLViz. 06-09-2024.

[27] Sean Falconer. OntoGraf. https://protegewiki.stanford.edu/wiki/OntoGraf. 06-09-2024.

[28] Sören Auer Andreas N. Rector Evgeny Pavlov y Daniel Viglianti. Steffen Lohmann, Sahar Jabeen. WebVOWL. https://github.com/VisualDataWeb/WebVOWL, 06-09-2024.

[29] Grupo de investigacion de la Universidad de Letonia. OWLGrEd. http://owlgred.lumii.lv/. 06-09-2024.

[30] Gregorio de Miguel Casado Francisco Javier Fabra Caro. *Ingeniería de Requisitos, Tema 3 - Captura y obtención de requisitos*. Universidad de Zaragoza.

[31] Nancy Rodrigues. Stakeholders: qué son y cuál es su impacto en las empresas. https://blog.hubspot.es/sales/que-es-stakeholder. 06-09-2024.

[32] Max Rehkopf. ¿Qué es un tablero de kanban? https://www.atlassian.com/es/agile/kanban/boards. 06-09-2024.

[33] Universidad de Stanford. Protégé. https://protege.stanford.edu/software.php. 06-09-2024.

[34] Wikipedia. Protégé (software). https://en.wikipedia.org/wiki/Prot%C3%A9g%C3%A9_(software). 06-09-2024.

[35] Miriam M. Escobar-Vega. WebVOWL: Visualization of individuals into ontologies. Trabajo fin de grado, maestría en sistemas computacionales, Tlaquepaque, Jalisco: ITESO, 2017.

[36] Ferdinando Villa Sergey Krivov, Richard Williams. GrOWL: A tool for visualization and editing of OWL ontologies. https://www.sciencedirect.com/science/article/pii/S1570826807000157, 06-09-2024.

[37] Grupo de investigación e-Lite del Politécnico de Turín. OntoSpere3D. https://ontosphere3d.sourceforge.net/. 06-09-2024.

[38] Silvia Esther Sánchez López. Modelo de indexación de formas en sistemas VIR basado en ontologías. Master's thesis, Universidad de las Américas Puebla, 14-05-2007.

[39] Oracle Java Documentation. JavaFX Overview. https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784. 06-09-2024.

[40] Oracle Java Documentation. Understanding the JavaFX Architecture. https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-architecture.htm#A1106328. 06-09-2024.

[41] Oracle Java Documentation. Working with the Canvas API. https://docs.oracle.com/javase/8/javafx/graphics-tutorial/canvas.htm#JFXGR214. 06-09-2024.

[42] Stefan Negru David Bold, Steffen Lohmann. VOWL. https://protegewiki.stanford.edu/wiki/VOWL. 06-09-2024.

Lista de Figuras

Lista de Tablas

Anexos

A. Visualizadores de ontología

Este anexo tiene como objetivo realizar una comparación más detallada entre diferentes herramientas de visualización de ontologías.

Para realizar una comparación entre los visualizadores mencionados, se ha utilizado la misma ontología, en este caso, la ontología Pizza[1] de Protégé. En situaciones donde un visualizador esté obsoleto y no se pueda ejecutar, se incluirá una visualización de la ontología obtenida de fuentes confiables en internet para asegurar una comparación justa.

El análisis abarca los siguientes visualizadores:

1. Protégé [25] con los *plugins* OWLViz [26] y OntoGraf [27]
2. WebVOWL [28]
3. OWLGrED [29]
4. GrOWL [36]
5. OntoSphere [37]

El estudio se centra en resaltar tanto las ventajas como las limitaciones de cada herramienta de visualización de ontologías. Es importante mencionar que se ha decidido incluir en este anexo dos visualizadores que no fueron mencionados en la sección 2.3, debido a que actualmente están obsoletos. Estos visualizadores son GrOWL y OntoSphere. Al encontrarse en desuso, no ha sido posible ejecutar sus aplicaciones, lo que ha impedido comprobar directamente su capacidad para realizar las funciones que se comparan en dicha sección. Sin embargo, se han incluido referencias visuales obtenidas de fuentes confiables para proporcionar una representación aproximada de su funcionalidad y permitir una comparación más completa.

[1] `https://protege.stanford.edu/ontologies/pizza/pizza.owl`, 06-09-2024

A.1. Protégé

Protégé es una plataforma gratuita y de código abierto que ofrece una serie de herramientas para construir modelos de dominio y aplicaciones basadas en conocimiento mediante ontologías [33].

Protégé proporciona una interfaz gráfica de usuario para definir ontologías. También incluye clasificadores deductivos que permiten validar que los modelos sean consistentes e inferir nueva información basada en el análisis de una ontología [34]. Integra razonadores como FaCT++[10], HermiT[11], entre otros, para permitir la inferencia sobre las ontologías y proporcionar a los usuarios una visión más completa y detallada del conocimiento representado.

Por defecto, Protégé solo permite visualizar la jerarquía de la ontología como un árbol, lo que limita la comprensión visual de las relaciones complejas entre los conceptos. Para conseguir que Protégé permita visualizar la ontología cargada, se debe añadir un plugin de visualización. En esta sección abordamos dos: OWLViz [26] y OntoGraph [27]. Se puede adquirir en la aplicación si se realiza una instalación completa de Protégé.

OWLViz permite visualizar y navegar de forma incremental por las jerarquías de clases en una ontología OWL, facilitando la comparación entre la jerarquía de clases definida y la jerarquía de clases inferida.

En cuanto a las ventajas de OWLViz:

- Muestra los niveles jerárquicos.
- Permite diferenciar los asertado de lo inferido mediante el uso de razonadores.
- Permite mostrar y ocultar nodos.

Como limitaciones que ofrece OWLViz:

- No visualiza las propiedades, instancias o clases anónimas.
- La visualización es estática, lo que no permite manipular los nodos.
- No muestra las propiedades.
- No ofrece resúmenes de la ontología.
- No permite la posibilidad de desplegar la ontología entera.

Debido a que no se puede desplegar la ontología entera, en la Figura A.1 representa parte del grafo de la ontología Pizza:

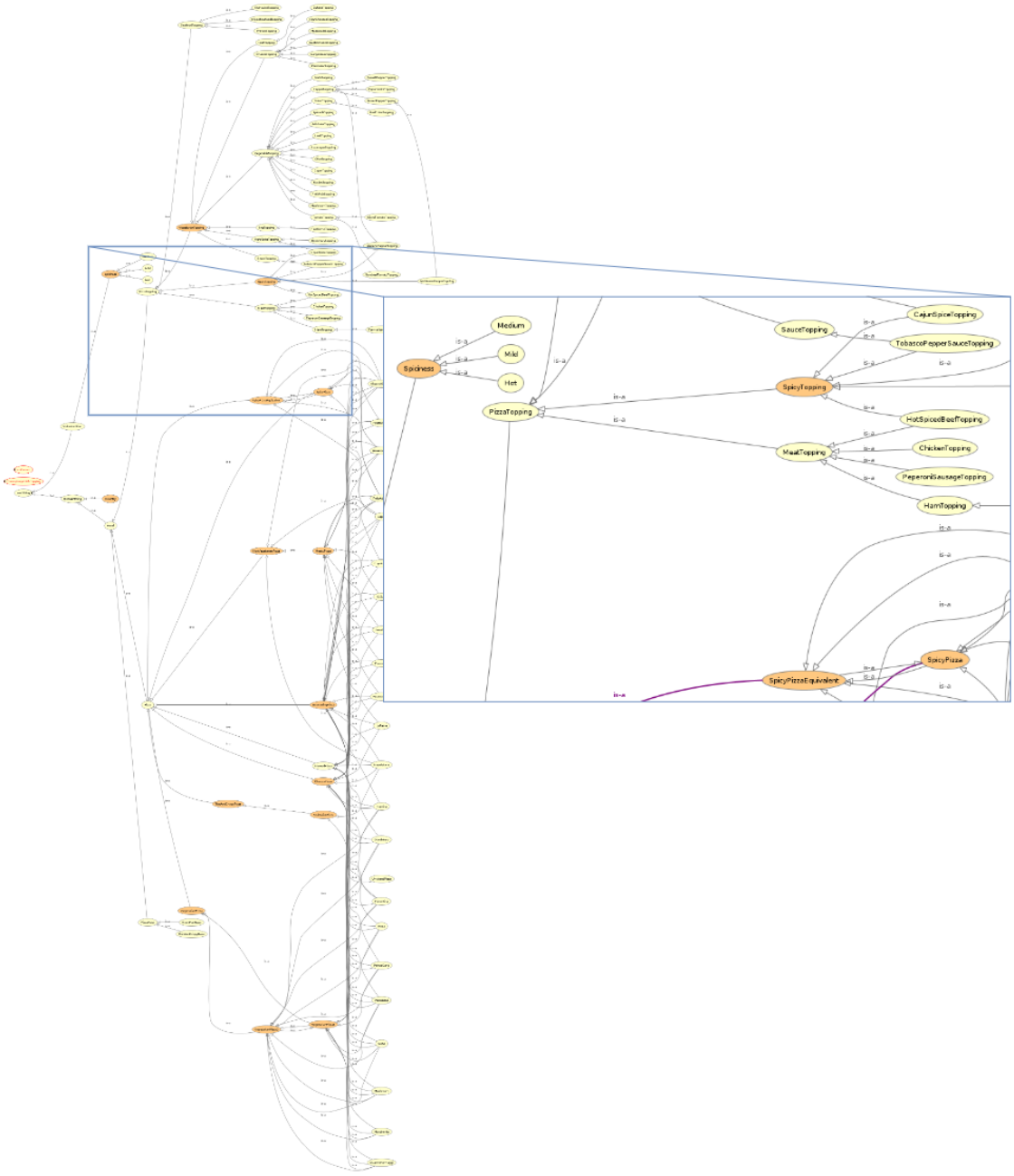

Figura A.1: Ejemplo de visualización con OWLViz y zoom aplicado a mano

OntoGraf está diseñado para la visualización interactiva de la ontología completa en forma de grafo, mostrando no solo las jerarquías de clases, sino también las relaciones más complejas entre diferentes elementos, como propiedades y conexiones entre instancias.

Las ventajas que tiene son:

- Permite mostrar/ocultar nodos.
- Muestra los niveles de las ontologías.
- Ofrece la posibilidad de cambiar la disposición del grafo, ya sea esta en cubo, en árbol mostrado de derecha a izquierda o árbol mostrado de arriba a abajo.
- Permite guardar el grafo y exportarlo.
- Permite diferenciar los asertado de lo inferido mediante el uso de razonadores.

Pero contiene las siguientes desventajas:

- Pese a disponer de una leyenda, los conectores pueden resultar difíciles de comprender.
- Se debe expandir nodo a nodo para conseguir visualizar la ontología completa o existe una funcionalidad de búsqueda que realizando un búsqueda del caracter "*" se despliega el grafo entero (no es intuitivo para el usuario).

Se visualizan las ontologías tal y como muestra la Figura A.2

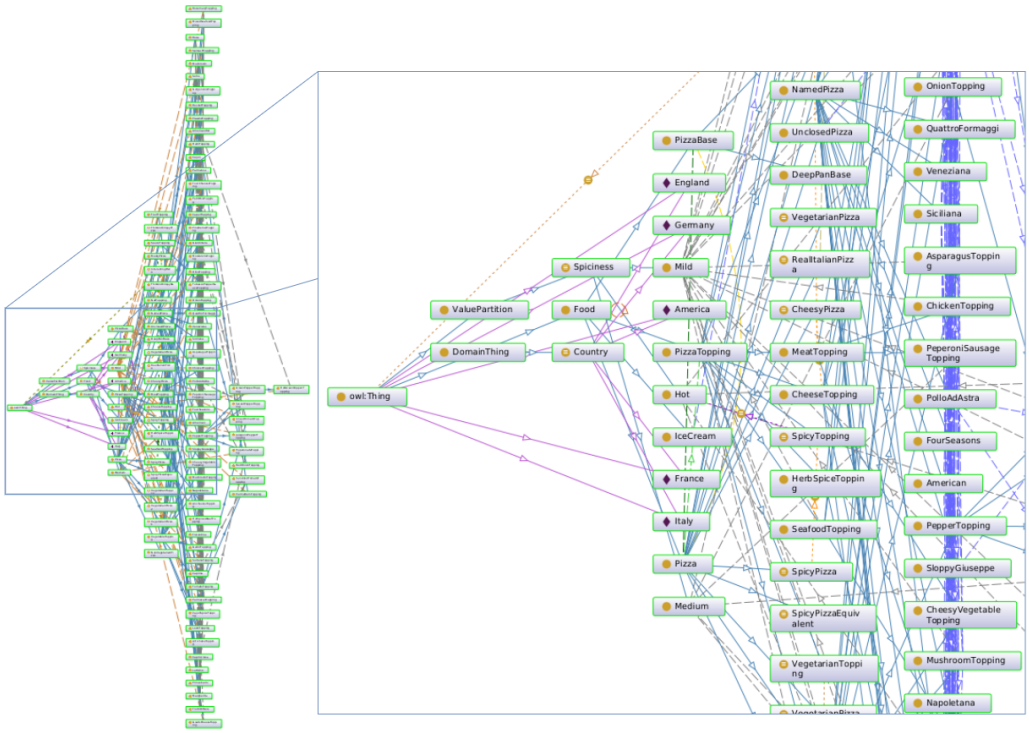

Figura A.2: Ejemplo de visualización con OntoGraph y zoom aplicado a mano

A.2. WebVOWL

WebVOWL (*Web-Based Visualization of Ontologies*) es una aplicación web para la visualización orientada al usuario de ontologías que implementa la notación visual VOWL[42] para ontologías OWL y está basado en estándares Web abiertos [35].

Las ventajas que aportan son:

- Ofrece un despegable al seleccionar un nodo con información relevante del mismo (nombre, tipo, dominio, rango, etc).
- Contiene un buscador dentro de la ontología.
- Permite editar, aunque está en etapa experimental.
- Ofrece la posibilidad de exportar la ontología como URL, JSON, SVG, TeX y TTL.

Como desventajas tiene:

- No representa de forma clara los niveles de jerarquía.
- La disposición de los nodos es confusa, incluso aparecen nodos repetidos.
- Las etiquetas con las propiedades se superponen entre sí, dificultando su visualización.
- Con ontologías grandes, se superponen los nodos, lo que dificulta la comprensión de la ontología.

En la Figura A.3 se muestra un ejemplo de visualización.

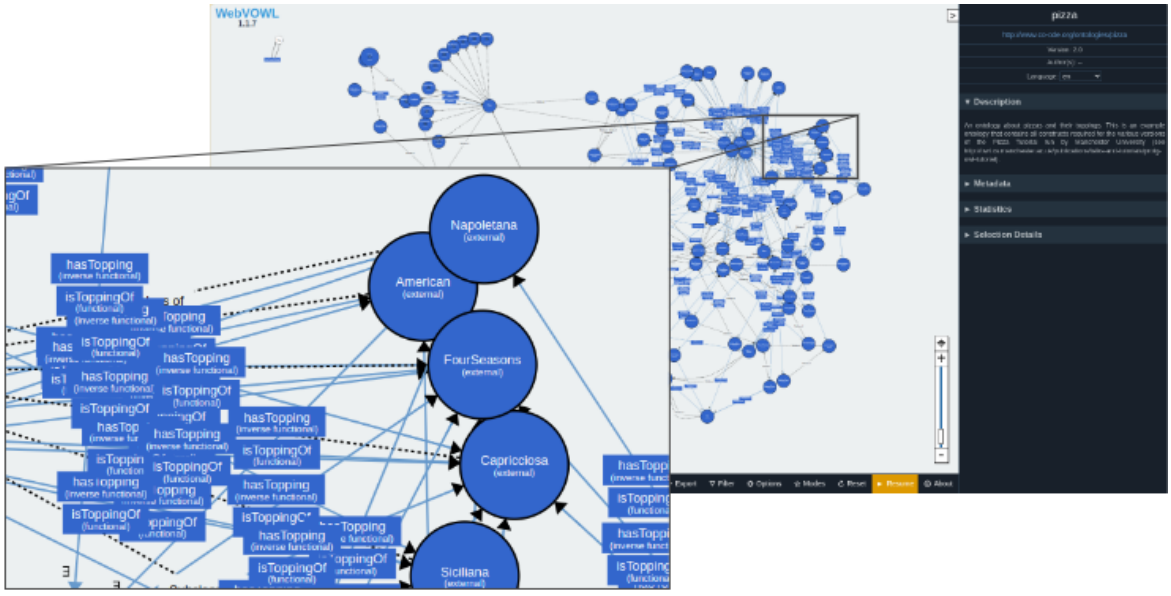

Figura A.3: Ejemplo de visualización con WebVOWL y zoom aplicado a mano

A.3. OWLGrEd

OWLGrEd es una herramienta de visualización y edición de ontologías OWL que utiliza una notación gráfica similar a los diagramas UML (Unified Modeling Language). Además de visualizar, OWLGrEd permite editar directamente las ontologías dentro de estos diagramas [29].

Las ventajas que ofrecen son:

- Permite editar ontologías.
- Ofrece la posibilidad de exportar la ontología en formato OWL, SVG y como imagen.
- Permite configurar la visualización de la ontología, permitiendo al usuario decidir que estilo va a tener cada nodo.

En cuanto a las desventajas:

- La visualización puede resultar confusa.
- No ofrece la posibilidad de usar un razonador.
- No permite la visualización parcial de una ontología.
- No visualiza la jerarquía de niveles.

Un ejemplo de visualización se presenta en la Figura A.4:

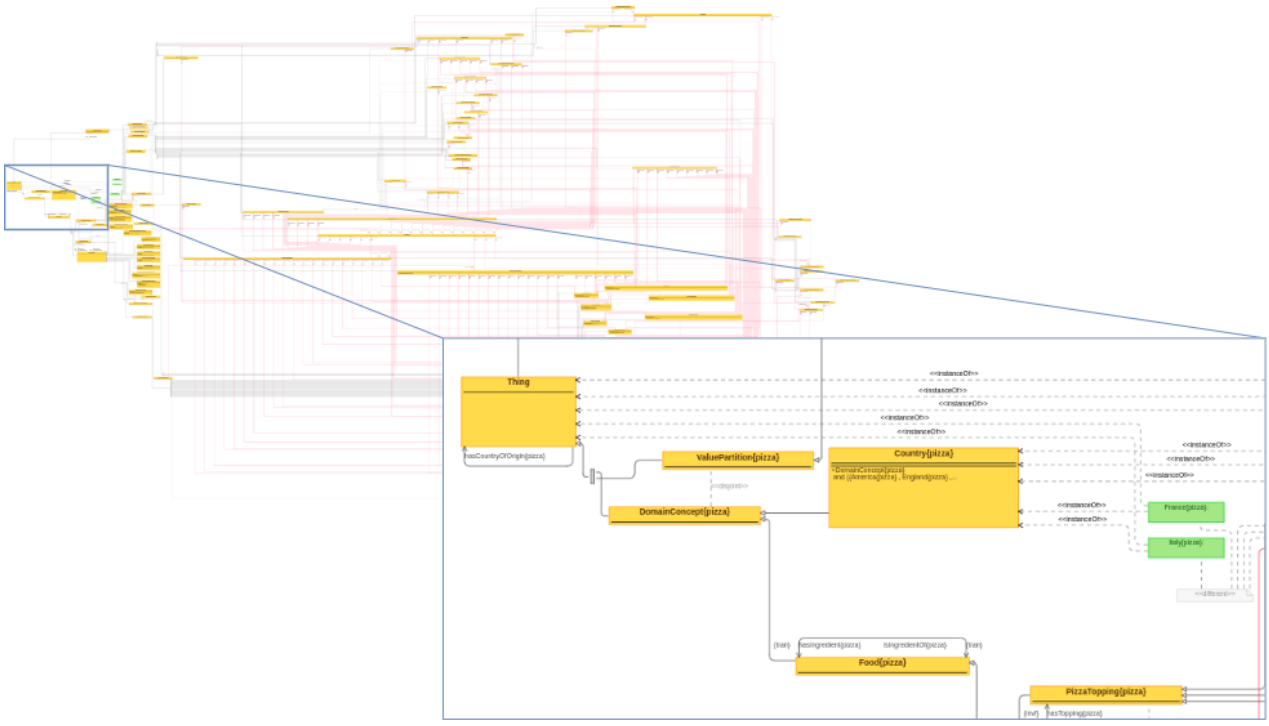

Figura A.4: Ejemplo de visualización con OWLGrEd y zoom aplicado a mano

A.4. GrOwl

GrOwl es un visualizador implementado como un applet de Java, un *plugin* para Protégé o una aplicación independiente de Java [36].

Como ventajas se puede destacar:

- Permite editar ontologías.
- Incluye un mecanismo de filtrado que permite restringir la vista parcialmente.
- Emplea colores, sombreados y formas específicas de los nodos para codificar propiedades de los constructores del lenguaje OWL.

Como desventajas:

- Actualmente se encuentra obsoleta.
- No representan la jerarquía de roles.
- No utiliza el razonador para completar el esquema.
- No existe nigun tipo de orden visual.

A pesar de los esfuerzos realizados para poner en funcionamiento el visualizador GrOWL, no fue posible hacerlo operativo, lo que impidió visualizar la ontología Pizza utilizando esta herramienta. Por ende, se incluye en la Figura A.5 la visualización de una ontología diferente a la de Pizza.

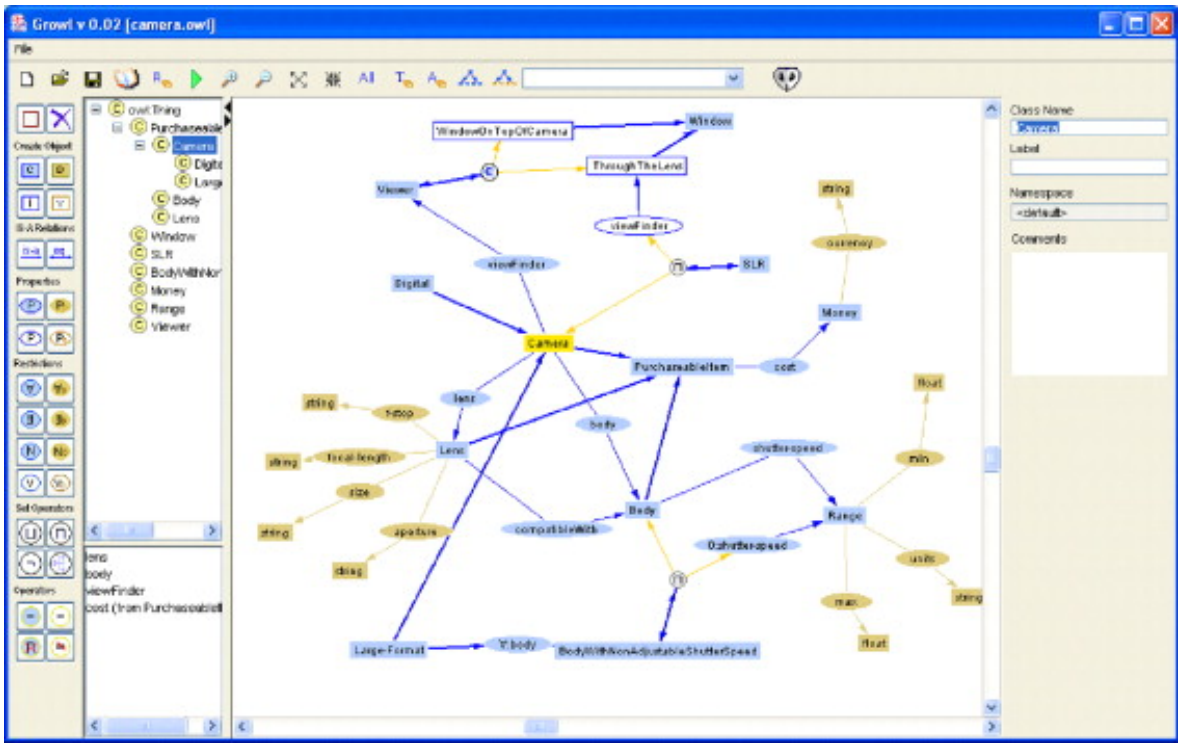

Figura A.5: Ejemplo de visualización con GrOwl

A.5. OntoSphere

OntoSphere es una herramienta de visualización de ontologías que utiliza un espacio tridimensional, donde la información se complementa mediante señales visuales (como el color o el tamaño de las entidades visualizadas). La herramienta tiene como objetivo abordar los problemas de visualización de modelos visuales de ontologías mediante la adopción de un mecanismo de colapso dinámico y diferentes vistas, en diferentes niveles de granularidad, para garantizar una navegabilidad constante del modelo renderizado [37].

Una interfaz de navegación intuitiva, que facilita la manipulación directa de la escena (rotación, desplazamiento, zoom, selección de objetos, etc.), permite a los expertos en el dominio, incluso aquellos con pocas habilidades técnicas en el campo de la Web Semántica, inspeccionar, modificar y revisar gráficamente los componentes de la ontología. La herramienta se ha implementado como un *plugin* para dos marcos diferentes: Protégé y Eclipse.

Aunque se descargó una versión antigua de OntoSphere que, según la información proporcionada en la página oficial[2], debería ser compatible con Protégé, no se logró que dicho software reconociera el *plugin*. A pesar de seguir los pasos indicados, Protégé no cargó correctamente OntoSphere, lo que impidió utilizarlo para la visualización de la ontología acordada. Sin embargo, se encontró en internet una imagen que muestra la visualización de la ontología en OntoSphere antes de que quedara obsoleto. Esta visualización se presenta en la Figura A.6.

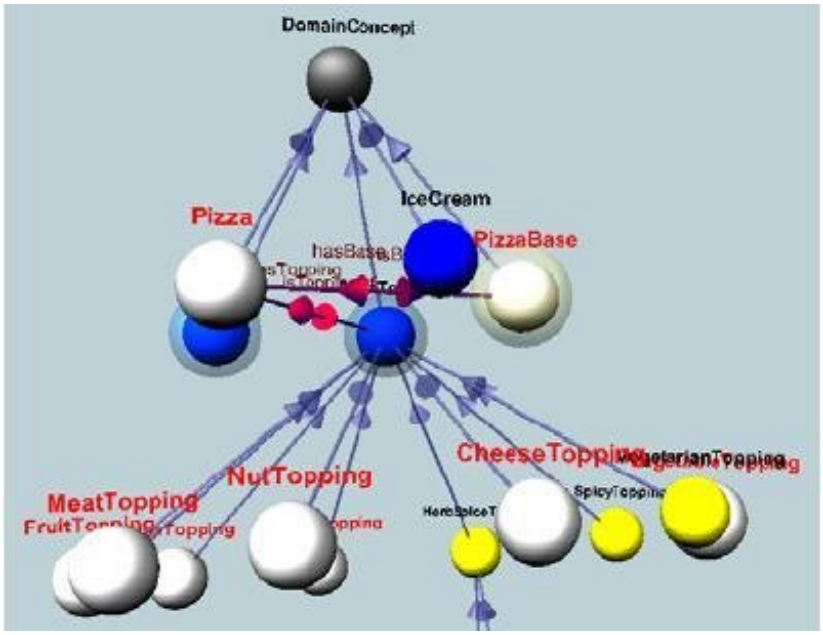

Figura A.6: Ejemplo de visualización con OntoSphere

[2] `https://ontosphere3d.sourceforge.net/install.html`, 06-09-2024

Adjunto nuevamente la tabla comparativa (Tabla A.1) de los visualizadores que actualmente están en funcionamiento. En ella, se presentan los requisitos funcionales que deberían cumplir los visualizadores de ontologías, indicando si cada uno de ellos cumple con dichos requisitos.

	OWLViz	OntoGraf	WebVOWL	OWLGrED	OntView
Mostrar grafo asertado	✓	✓	✓	✓	✗ *
Mostrar grafo inferido	✓	✓	✗	✗	✓
Uso de razonadores	✓	✓	✗	✗	✓
Visualizar jerarquía	✓	✓	✗	✗	✓
Visualizar clases anónimas	✗	✗	✗	✗	✓
Visualizar GCIs **	✗	✗	✗	✗	✓
Modificar disposición de elementos en pantalla	✗	✓	✓	✗	✓
Ocultar parcialmente el grafo	✓	✓	✓	✗	✓
Listar instancias de las clases	✗	✗	✓	✓	✓
Visualizar jerarquía de roles y propiedades	✗	✗	✓	✓	✓
Guardar estado	✗	✓	✗	✗	✓
Exportar imágenes	✓	✓	✓	✓	✓

Tabla A.1: Comparación entre visualizadores

* Se tiene marcado como trabajo a futuro.

** Axiomas de subsunción en los cuales los conceptos pueden ser anónimos.

B. Funcionalidades del sistema

Este anexo tiene como objetivo proporcionar una descripción detallada de las funcionalidades disponibles en el visualizador OntView.

B.1. Funcionalidades de la barra de herramientas

La interfaz gráfica de usuario incluye una barra de herramientas en la parte superior de la pantalla (Figura B.1), donde se encuentran todas las funcionalidades que el usuario puede seleccionar. A continuación, se presenta una descripción detallada de cada funcionalidad.

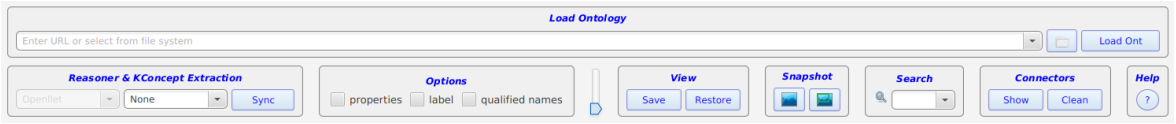

Figura B.1: Funcionalidades de la barra de herramientas

B.1.1. Carga de ontologías

Permite al usuario cargar ontologías desde una URL(1) o un archivo del sistema(2). Además, la barra de búsqueda funciona como un desplegable que almacena las URL de las ontologías utilizadas recientemente, como se muestra en la Figura B.2.

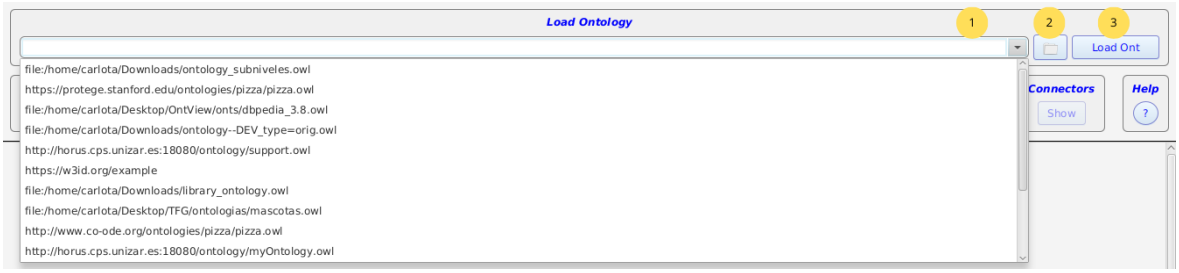

Figura B.2: Cargar una ontología

Para ayudar al usuario a conocer la funcionalidad(2), al dejar el ratón sobre el botón, aparece un *tooltip* con una explicación del botón, como se muestra en la Figura B.3.

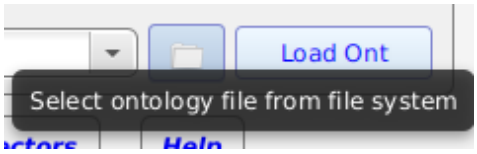

Figura B.3: *Tooltip* sobre seleccionar una ontología desde el sistema

Una vez seleccionada la ontología a cargar, se debe hacer clic en el botón `Load Ont`(3). Dependiendo de la situación, pueden ocurrir tres escenarios:

1. La ontología se carga correctamente, lo que desbloqueará la opción de seleccionar un razonador.
2. Si el sistema no puede cargar la ontología, se mostrará un mensaje de error, como se ilustra en la Figura B.4.
3. Si el usuario intenta cargar sin haber seleccionado una ontología, aparecerá un mensaje de error, como se muestra en la Figura B.5.

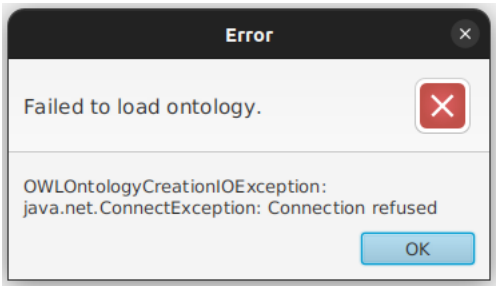

Figura B.4: Error: no se puede cargar la ontología seleccionada

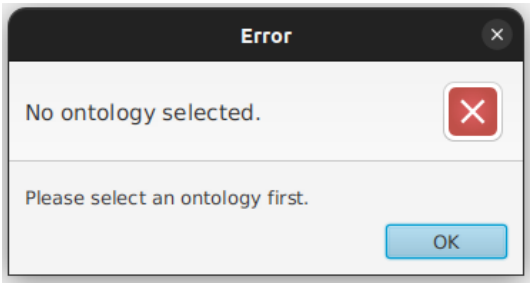

Figura B.5: Error: ontología no seleccionada

B.1.2. Selección del razonador y KCE

Permite al usuario seleccionar el razonador que se utilizará para inferir sobre la ontología cargada y extraer conceptos clave mediante algoritmos de KConceptExtraction (Figura B.7). Es importante destacar que la opción de seleccionar el razonador solo estará disponible después de que la ontología haya sido cargada correctamente. Actualmente, el único razonador(1) disponible es Openllet.

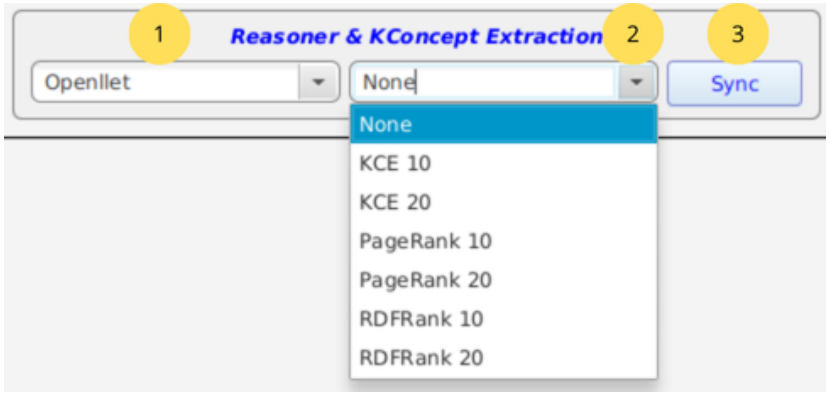

Figura B.6: Seleccionar razonador y KConceptExtraction

Para KConceptExtraction(2), el usuario puede elegir entre las siguientes opciones:

1. **None:** No se realizará ninguna extracción de conceptos adicionales.
2. **KCE10/KCE20:** Utiliza el algoritmo KCE con diferentes parámetros (10 o 20) para extraer conceptos clave.
3. **PageRank10/PageRank20:** Aplica el algoritmo PageRank para identificar conceptos importantes dentro de la ontología.
4. **RDFRank10/RDFRank20:** Emplea el algoritmo RDFRank para extraer conceptos relevantes de la ontología.

Si el usuario decide seleccionar el botón `Sync`(3), sin antes haber cargado una ontología, el programa mostrará una ventana de error (Figura B.7).

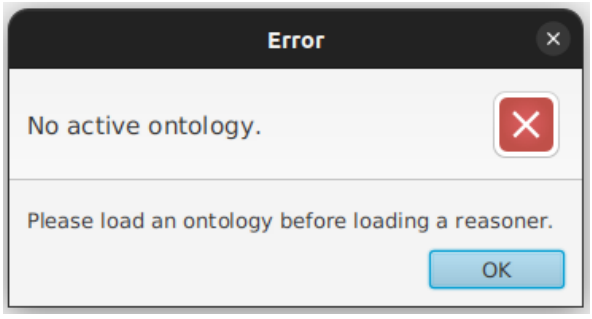

Figura B.7: Error: se necesita tener cargada una ontología

Teniendo seleccionado un razonador y una de las opciones de KConceptExtraction, se selecciona el botón `Sync`(3) para poder inferir sobre la ontología, que se mostrará visualmente en la parte inferior del programa.

En la Figura B.8 se muestra la ontología desplegada al completo y en la Figura B.9 se ha empleado RDFRank20.

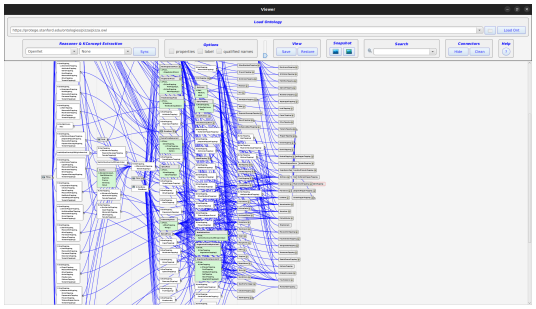

Figura B.8: Sin extractor de conceptos

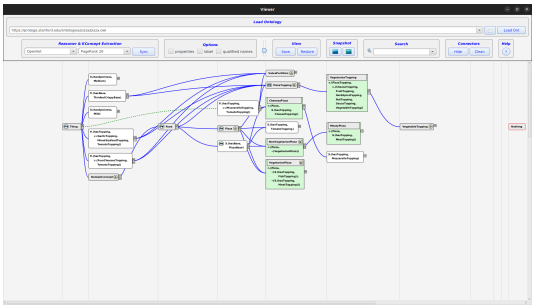

Figura B.9: Con extractor de conceptos

B.1.3. Opciones de visualización

Permite al usuario seleccionar tres opciones diferentes, siendo ninguna excluyente entre sí. Se pueden observar la Figura B.10.

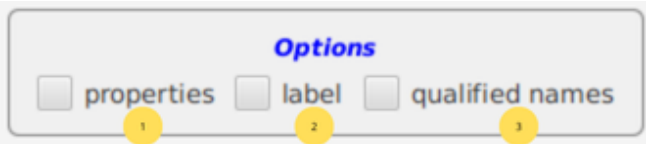

Figura B.10: Mostrar/ocultar properties, labels, qualified names

- Al seleccionar `properties`(1), se le muestra u oculta al usuario todas las propiedades existentes en la ontología.
- Al seleccionar `label`(2), se le muestra u oculta al usuario todas las etiquetas existentes en la ontología.
- Al seleccionar `qualified names`(3), se le muestra u oculta al usuario todas los nombres calificativos existentes en la ontología.

B.1.4. Zoom interactivo

Permite al usuario acercar o alejar la vista del grafo utilizando un deslizador (Figura B.11). El usuario puede ajustar el nivel de zoom arrastrando el control deslizante con el ratón para obtener una vista más detallada o una visión general del grafo.

Figura B.11: Aumentar/disminuir zoom

B.1.5. Guardar y restaurar vista

Permite al usuario guardar (`save`(1)) la vista actual del grafo como un archivo XML, manteniendo la posición y disposición de los nodos. De esta manera, el usuario puede guardar el estado del grafo y no perder su disposición al cerrar la aplicación. Posteriormente, puede restaurar el grafo usando la función `restore`(2) (Figura B.12). Es importante destacar que, para restaurar la vista, se debe cargar y razonar sobre la misma ontología de la que se desea restaurar.

Figura B.12: Guardar/restaurar grafo en XML

Nuevamente, el usuario dispone de *tooltips* para conocer su funcionalidad (Figura B.13 y Figura B.14).

Figura B.13: *Tooltip* sobre capturar el grafo en formato XML

Figura B.14: *Tooltip* sobre restaurar el grafo

B.1.6. Capturas de pantalla

Permite tomar dos tipos de capturas (Figura B.15) guardadas en formato PNG:

- **Completa**(1)**:** Captura la ontología completa. Esto significa que la imagen mostrará el grafo en su totalidad, sin importar la parte visible en la pantalla.
- **Parcial**(2)**:** Captura únicamente la parte del grafo que el usuario ve en la pantalla.

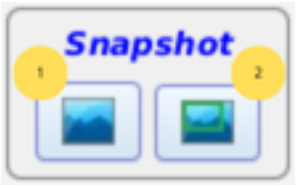

Figura B.15: Capturas del grafo en PNG

Para ayudar al usuario, ambas opciones contiene un *tooltip* al reposar encima de cada funcionalidad el ratón (Figura B.16 y Figura B.17).

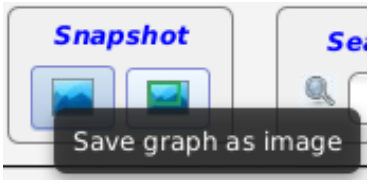

Figura B.16: *Tooltip* sobre capturar el grafo completo

Figura B.17: *Tooltip* sobre capturar el grafo parcialmente

B.1.7. Búsqueda en el grafo

Permite al usuario realizar búsquedas dentro del grafo para localizar y enfocar elementos específicos de la ontología (Figura B.18). Al encontrar un elemento, el grafo se ajusta automáticamente para centrar la vista en el nodo buscado.

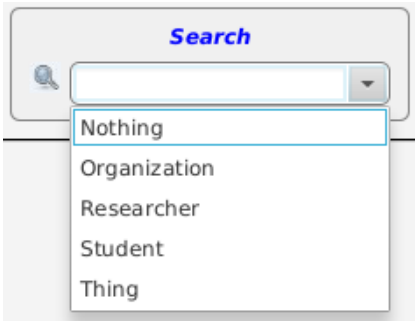

Figura B.18: Búsqueda de nodos en el grafo

B.1.8. Mostrar/ocultar y limpiar conectores

En cuanto a los conectores, el programa ofrece dos funcionalidades clave. La primera permite al usuario mostrar u ocultar todos los conectores de la ontología(1). Al mostrar los conectores, se revelan las relaciones directas entre los nodos, facilitando la comprensión de la estructura ontológica (Figura B.19). Por otro lado, al ocultarlos, se reduce el ruido visual, lo que es útil para enfocarse en nodos específicos o en otras propiedades del grafo. La segunda funcionalidad incluye la posibilidad de limpiar los conectores seleccionados por el usuario(2) (Figura B.19).Esta funcionalidad surge a partir de la opción que permite al usuario seleccionar un nodo específico en el grafo para visualizar sus conectores (Para más detalle ver Sección B.2.3).

Figura B.19: Mostrar/ocultar y limpiar conectores

B.1.9. Botón de ayuda

Proporciona dos leyendas informativas que ayudan al usuario a navegar por la aplicación (Figura B.20).

Figura B.20: Ayuda al usuario

La primera leyenda ofrece una guía rápida sobre los pasos que se deben seguir para comenzar a utilizar la aplicación (Figura B.21). La segunda leyenda explica el significado de los diferentes elementos visuales dentro de la interfaz. Específicamente, detalla cómo interpretar los símbolos y colores que representan cada clase (Figura B.22).

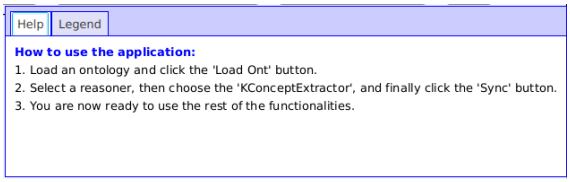

Figura B.21: Leyenda sobre como emplear la aplicación.

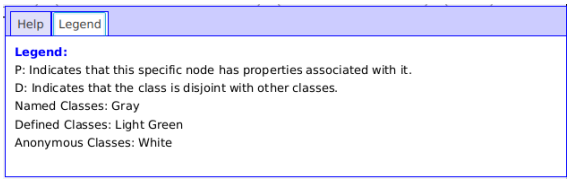

Figura B.22: Leyenda sobre los elementos del grafo.

B.2. Funcionalidades interactivas en el grafo

Además de las funcionalidades disponibles en la barra de herramientas superior, OntView permite al usuario manipular y analizar el grafo directamente. A continuación, se presentan estas funcionalidades detalladas:

B.2.1. Mover un nodo del grafo

Permite al usuario desplazar un nodo dentro del *canvas*, manteniendo su posición fija en el eje X, que corresponde a su nivel en la jerarquía del grafo. Los nodos están limitados a moverse solo dentro de su nivel, marcado por una línea gris clara. Además, se ha implementado un mecanismo de repulsión para prevenir la superposición de nodos, asegurando una disposición visualmente ordenada. Cuando se selecciona el nodo, los conectores del mismo se muestran en color naranja (Figura B.23).

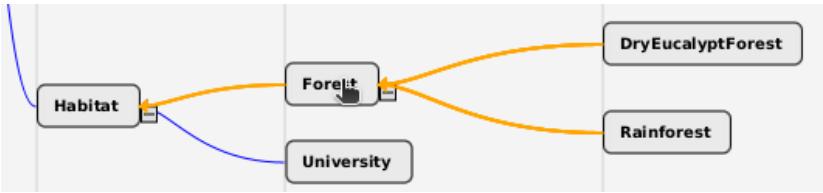

Figura B.23: Mover un nodo del grafo

B.2.2. Centrar nodo

Permite al usuario centrar el grafo en la posición de un nodo específico con un solo clic. Al hacer esto, los conectores asociados a ese nodo se resaltan en color azul, facilitando la identificación de sus relaciones dentro del grafo.

B.2.3. Mostrar conectores específicos

Permite al usuario que al seleccionar un nodo específico en el grafo muestre los conectores de dicho grafo (Figura B.24). Si se selecciona el mismo nodo por segunda vez, desaparecerán visualmente los conectores de dicho nodo.

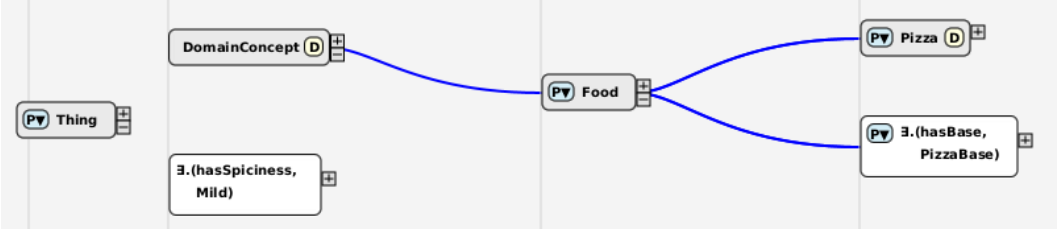

Figura B.24: Mostrar conectores específicos

B.2.4. Mostrar propiedades

Permite al usuario seleccionar un nodo específico en el grafo para visualizar sus propiedades. Un nodo con propiedades se identifica por la presencia de una "P" en su interior. Esta "P" actúa como un botón, que, al ser seleccionado, despliega las propiedades del nodo. Si las propiedades del nodo están vinculadas a las de otro nodo, estas también se mostrarán en el despliegue (Figura B.25).

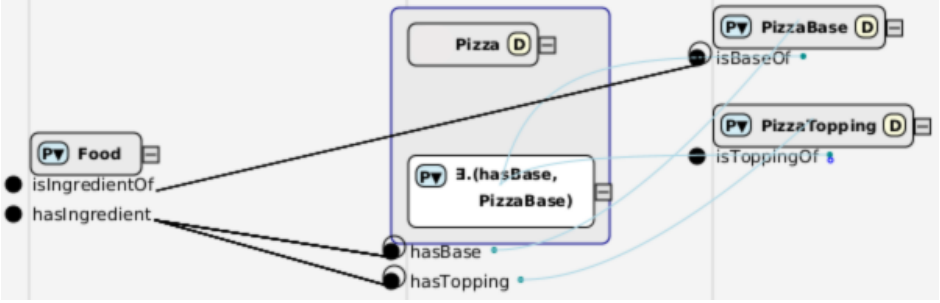

Figura B.25: Mostrar propiedades

Además, en ciertos tipos de propiedades, se encuentran disponibles *tooltips* adicionales. Estos *tooltips* están indicados por un círculo negro ubicado a la izquierda de las propiedades correspondientes (Figura B.26).

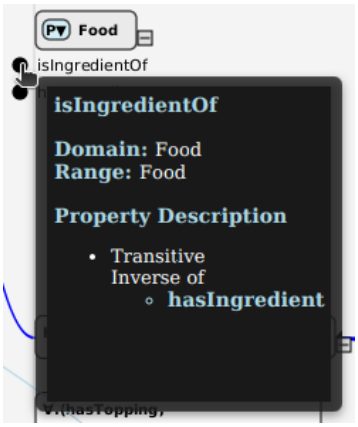

Figura B.26: *Tooltip* de propiedades

B.2.5. Repulsión entre nodos

El programa cuenta con un algoritmo implementado que maneja la separación automática de nodos cuando ocurre una colisión, es decir, cuando un nodo se superpone a otro. Este mecanismo de repulsión entre nodos garantiza que, en situaciones de colisión, los nodos sean redistribuidos de manera que no se superpongan, permitiendo al usuario visualizar claramente todos los elementos en la interfaz.

B.2.6. Mostrar etiquetas

Permite al usuario visualizar las etiquetas asociadas a los conceptos en el grafo (Figura B.27). Como se ha explicado anteriormente, las etiquetas son identificadores breves que proporcionan información adicional o descripciones alternativas para los conceptos, ayudando a clarificar su significado y uso en la ontología.

Figura B.27: Mostrar etiquetas

En este caso específico, al activar la funcionalidad, el concepto previamente visualizado como `Food` se detalla con su etiqueta completa, pasando a mostrarse como `‘‘Food’’@en`, lo que resalta la información lingüística asociada al concepto. `@en` significa que el literal está en inglés.

B.2.7. Mostrar nombres calificativos

Permite al usuario visualizar los nombres completos de los conceptos en el grafo, incluyendo cualquier calificación adicional que pueda diferenciar un concepto de otros similares (Figura B.28). Esto es útil para conocer el uso de *namespaces*, ontologías modulares y reusables que importamos y así se puede saber el origen de los términos.

Figura B.28: Mostrar nombres calificativos

En este ejemplo, al habilitar la opción de mostrar nombres calificativos, el concepto representado inicialmente como `Thing` se actualiza para reflejar su denominación completa, apareciendo ahora como `owl:Thing`.

B.2.8. Mostrar/ocultar nodo

Para mostrar un nodo que tiene hijos ocultos, el usuario puede utilizar el botón [+] que aparece a la derecha del nodo.

En cambio, para ocultar se permite hacer de dos forma diferentes:

1. Hacer doble clic en el nodo, siempre y cuando este no tenga hijos visibles.
2. Hacer clic en el símbolo [-], también ubicado a la derecha del nodo.

Ambas funcionalidades aparecen en la Figura B.29. Cabe destacar que si no existen nodos para mostrar u ocultar, los símbolos [+] y [-] no estarán visibles.

Figura B.29: Mostrar/ocultar nodos

B.2.9. *Tooltip* con la información extra de la clase

Permite al usuario visualizar la información extra de la clase manteniendo el cursor del ratón sobre él (Figura B.30).

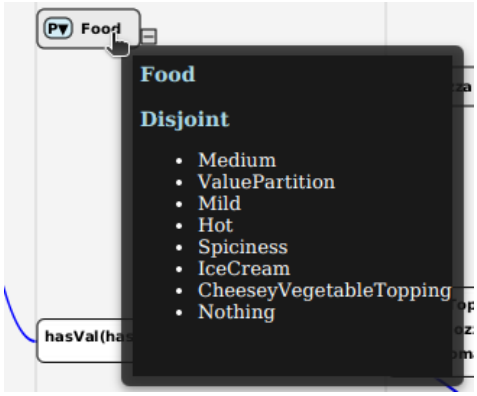

Figura B.30: *Tooltip* con la información extra de la clase

B.2.10. Menú contextual

Al hacer clic derecho en el visualizador, se despliega un menú contextual que ofrece opciones específicas según el área en la que se haya realizado el clic.

- **Sobre un elemento del grafo:** Si se realiza el clic derecho sobre un nodo del grafo, el menú presenta varias opciones específicas para la gestión de ese nodo (Figura B.31):
  1. Mostrar las instancias (Figura B.33).
  2. Mostrar/ocultar las propiedades del nodo.
  3. Ocultar el nodo seleccionado, siempre y cuando no tenga hijos visibles. En caso contrario, la opción permanecerá deshabilitada.
- **Sobre el *canvas*:** Si el clic derecho se realiza en un área vacía del *canvas*, el menú contextual ofrece opciones para gestionar el grafo en su totalidad (Figura B.32):
  1. Mostrar/ocultar las propiedades del grafo.
  2. Mostrar/ocultar las relaciones disjuntas del grafo.
  3. Mostrar/ocultar los rangos del grafo.

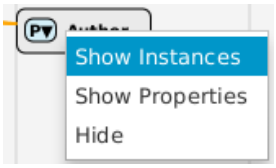

Figura B.31: Menú contextual de un nodo

Figura B.32: Menú contextual sobre el *canvas*

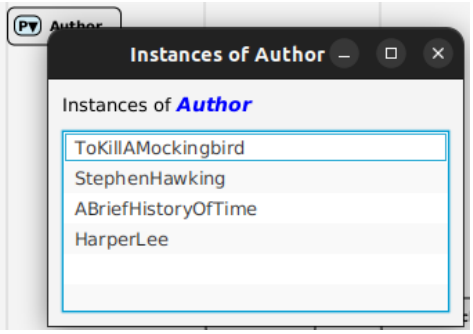

Figura B.33: Mostrar instancias de un nodo

C. OntView 1.0 vs OntView 2.0

En este anexo se presenta una comparación visual entre OntView 1.0 y OntView 2.0, enfocándose en las mejoras implementadas en la representación gráfica de las ontologías. A través de esta comparación, se destacan los cambios en los elementos visuales que reflejan las optimizaciones realizadas en la nueva versión. Dado que la eficiencia de las funcionalidades no se puede capturar en una imagen, este anexo se centrará exclusivamente en los aspectos visuales.

C.1. Interfaz Gráfica de Usuario (GUI)

En OntView 2.0, la barra de herramientas ha sido rediseñada para ofrecer una experiencia más intuitiva y accesible, adaptándose además a diferentes tamaños de pantalla, algo que la versión 1.0 no permitía (Figura C.1). Mientras que en la versión anterior las opciones estaban dispuestas de manera compacta y fija, la nueva versión (Figura C.2) reorganiza estas funcionalidades, distribuyéndolas de forma más amigable.

OntView 2.0 también amplía sus capacidades con nuevas funcionalidades, además de permitir las ya existentes. Estas mejoras complementan las opciones ya existentes, como guardar y restaurar vistas, o capturar pantallas directamente desde la interfaz. Una característica destacada de esta versión es la inclusión de *tooltips* informativos en todas las funcionalidades que podrían requerir una explicación adicional. Estos *tooltips* están diseñados para guiar al usuario, proporcionando aclaraciones útiles que facilitan la comprensión y el uso efectivo de la herramienta, tanto para las funciones nuevas como para las ya conocidas.

Figura C.1: Barra de herramienta OntView 1.0

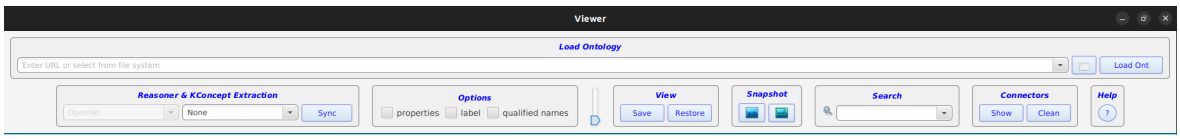

Figura C.2: Barra de herramienta OntView 2.0

C.2. Visualización del grafo

A continuación, se presentan cuatro ontologías de distintos tamaños: pequeña, mediana y grande, para evaluar cómo OntView 1.0 y OntView 2.0 gestionan la visualización de sus estructuras.

C.2.1. Proyectos

La ontología de proyectos es una ontología pequeña, compuesta por 4 clases primitivas, de las que 2 son clases definidas, y 5 propiedades. Es utilizada principalmente para probar la representación gráfica y las funcionalidades del visualizador.

La visualización en OntView 1.0 (Figura C.3) muestra las clases definidas se encuentran distribuidas en varios niveles. En esta versión, las expresiones que definen las clases, como en los casos de "jefes"' o "superpro", se representan como nodos separados que derivan de sus expresiones. Esta disposición, aunque funcional, introduce niveles adicionales que pueden generar ruido visual, dificultando la comprensión rápida de la estructura ontológica.

Por otro lado, OntView 2.0 (Figura C.4) simplifica esta visualización al compactar las clases definidas en un único nodo. Este nodo agrupa tanto el nombre de la clase como la expresión que la define, eliminando la necesidad de múltiples niveles y reduciendo significativamente el ruido visual. Al unificar esta información en un solo punto, la estructura se vuelve más clara y fácil de entender, permitiendo a los usuarios captar de inmediato las relaciones clave dentro de la ontología.

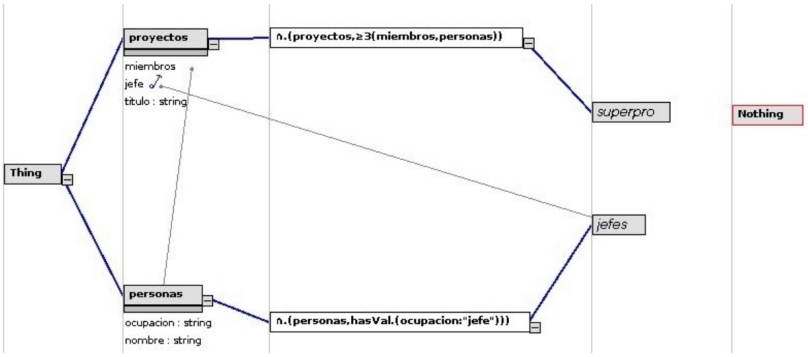

Figura C.3: Visualización de *proyectos.owl* en OntView 1.0

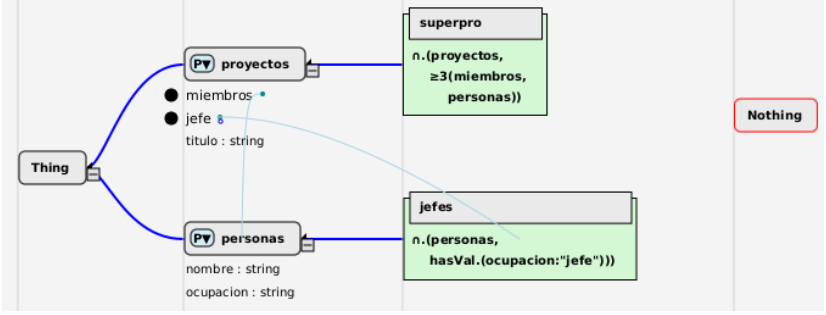

Figura C.4: Visualización de *proyectos.owl* en OntView 2.0

C.2.2. Koala

La ontología de Koala[1] es de tamaño moderado. Su complejidad intermedia la hace adecuada para evaluar cómo los visualizadores manejan relaciones y clases más complejas dentro de una estructura ontológica más extensa que la ontología de proyectos.

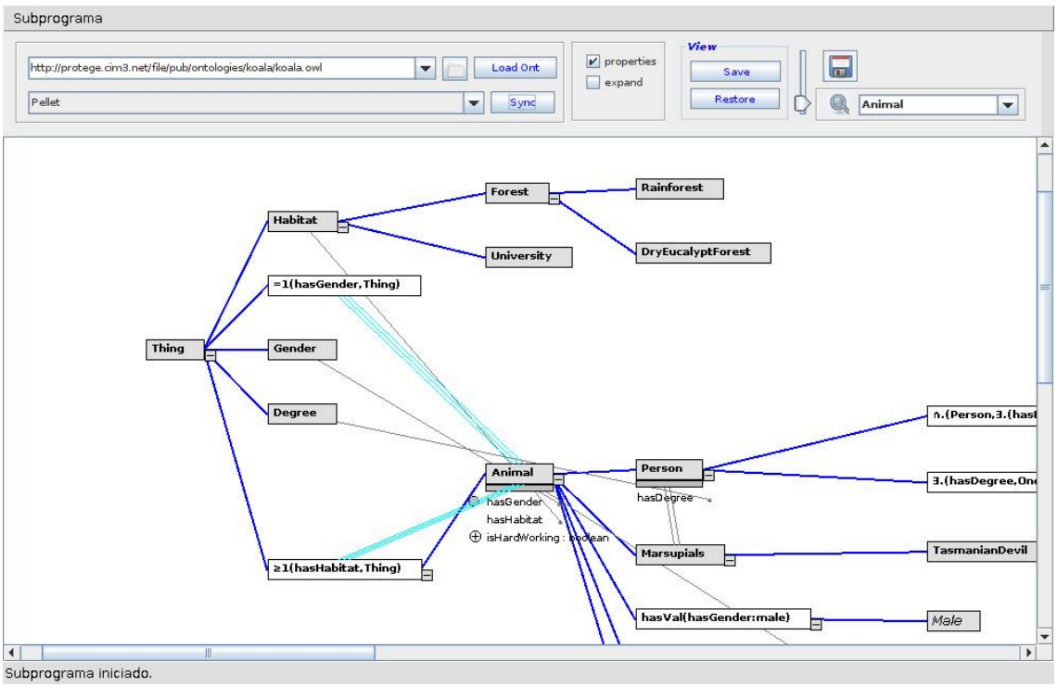

Figura C.5: Visualización de *koala.owl* en OntView 1.0

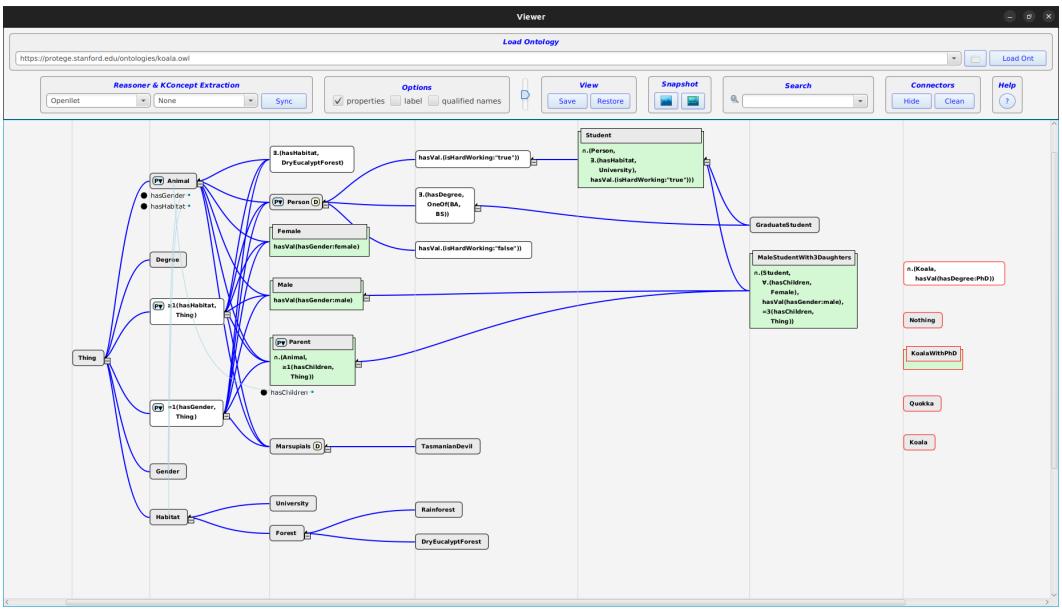

Figura C.6: Visualización de *koala.owl* en OntView 2.0

[1] `https://protege.stanford.edu/ontologies/koala.owl`, 06-09-2024

Al comparar OntView 1.0 (Figura C.5) y OntView 2.0 (Figura C.6), es importante señalar que OntView 2.0 no solo presenta una visualización más organizada, sino que también muestra un mayor nivel de detalle. Aunque esto podría hacer la comparación menos equitativa, ya que 1.0 visualiza menos información, la experiencia de usuario en OntView 2.0 sigue siendo más fluida. A pesar del incremento en la cantidad de datos presentados, la disposición más clara y la capacidad de distinguir fácilmente entre diferentes tipos de clases hacen que la navegación y la comprensión de la estructura de la ontología sean más eficientes y accesibles en la versión 2.0.

C.2.3. Animales

La ontología de animales es de tamaño mediano, con 53 clases o conceptos y un total de 122 axiomas. Su tamaño y complejidad hacen que sea una buena prueba para evaluar cómo un visualizador maneja una ontología más grande y con múltiples relaciones.

En OntView 1.0 (Figura C.7), la visualización de la ontología de animales presenta dificultades en cuanto a la organización y la claridad. Las clases no se distinguen claramente entre sí, lo que complica la identificación de las relaciones y la interpretación general del grafo. Además, cierta información relevante no se representa, lo que puede limitar la comprensión completa de la ontología. En contraste, OntView 2.0 (Figura C.8) ha realizado mejoras significativas en la visualización. A pesar del mayor tamaño y complejidad de la ontología, el visualizador permite identificar rápidamente el tipo de cada clase. Es fácil distinguir entre clases primitivas, clases anónimas y clases definidas, lo que facilita la comprensión de la estructura ontológica. Esta clara diferenciación entre los tipos de clases mejora la navegación y permite una interpretación más precisa y eficiente del grafo.

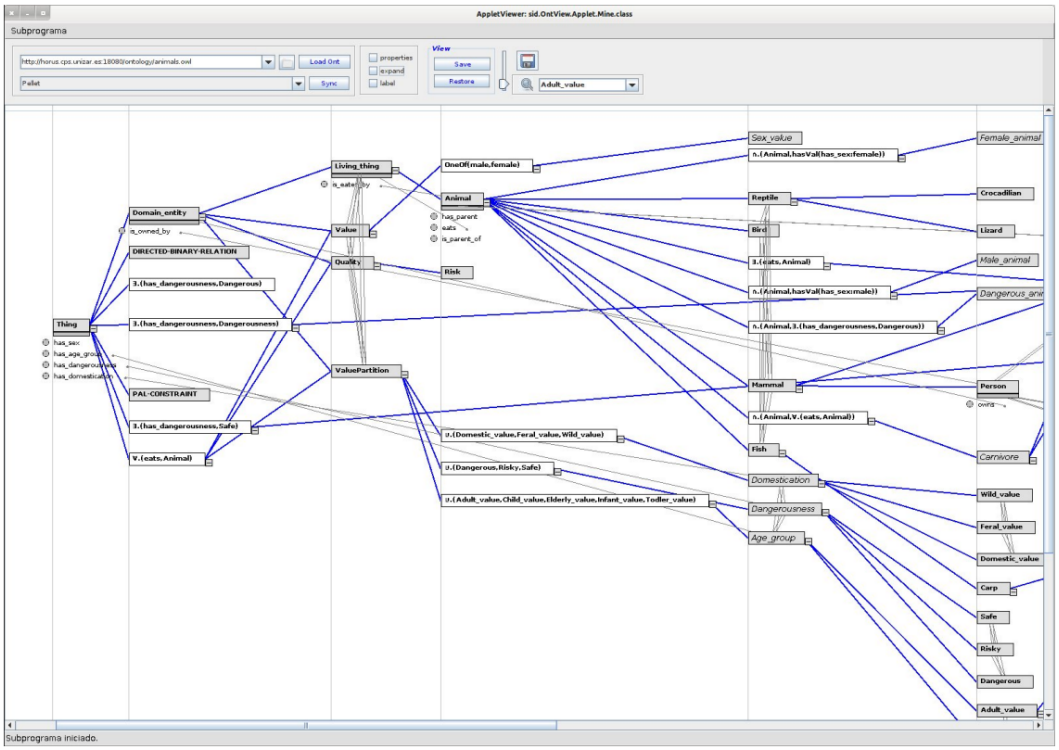

Figura C.7: Visualización de *animals.owl* en OntView 1.0

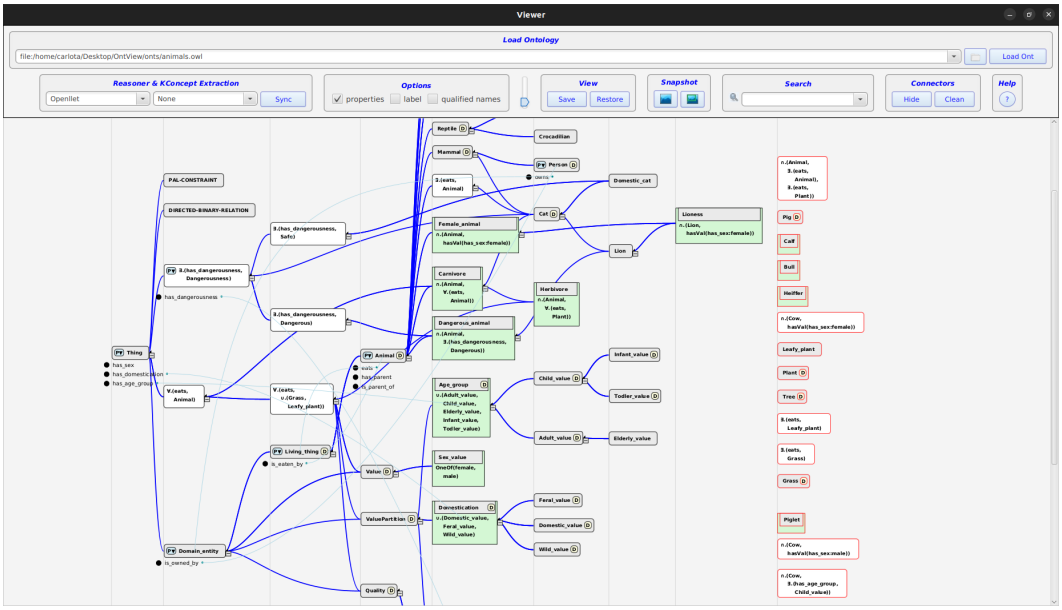

Figura C.8: Visualización de *animals.owl* en OntView 2.0

C.2.4. DBpedia

La ontología de DBpedia es considerablemente grande, con 768 clases y 3.000 propiedades. En OntView 2.0 (Figura C.9), la visualización de esta ontología muestra una jerarquía más clara y definida en comparación con la versión 1.0 (Figura C.10), lo que facilita la comprensión de las relaciones entre clases. Además, la interfaz en OntView 2.0 es más limpia, lo que mejora la legibilidad y permite una exploración más eficiente de la ontología compleja de DBpedia (ambas figuras en la página siguiente por maquetación).

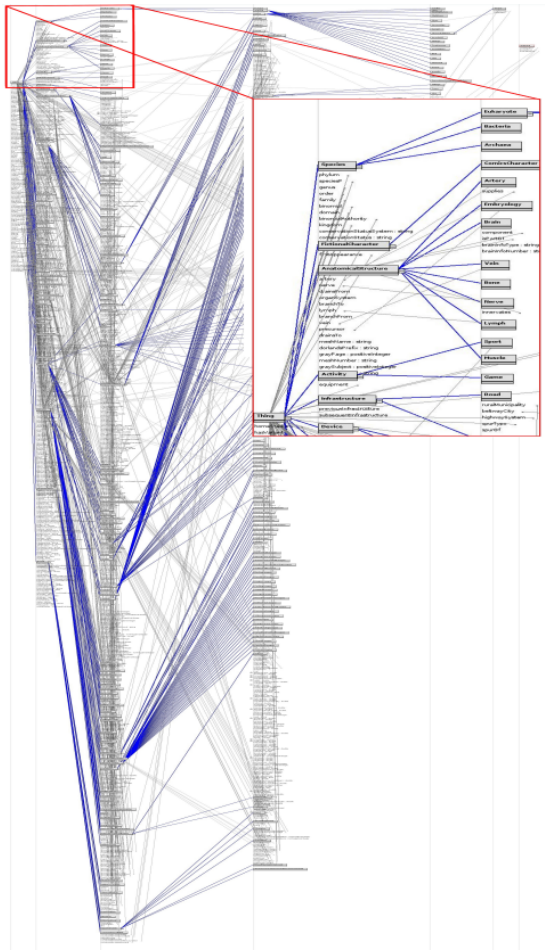

Figura C.9: Visualización de *DBpedia.owl* en OntView 1.0

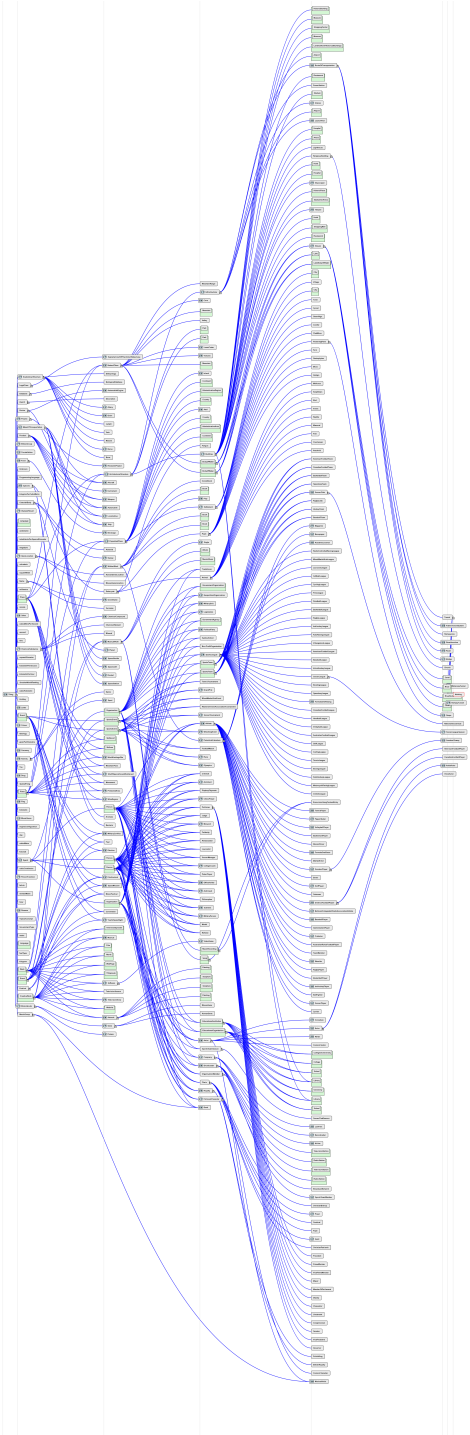

Figura C.10: Visualización de *DBpedia.owl* en OntView 2.0

D. Elección de tecnologías

En este anexo se detalla el proceso de selección de tecnologías para la actualización.

Para esta nueva versión del proyecto, se exploraron varios enfoques para encontrar la mejor manera de actualizar el sistema. Para tomar una decisión bien informada, se realizó una nueva evaluación de unas pruebas previamente diseñadas en el contexto de la segunda versión de OntView. La ejecución de estas pruebas tenía como objetivo evaluar el tiempo de ejecución y el rendimiento de diferentes combinaciones de tecnologías, para identificar la opción que ofreciera el mejor rendimiento en términos de velocidad y eficiencia.

Las pruebas consistieron en dibujar una cantidad variable de rectángulos amarillos, cada uno de tamaño 100x20 píxeles, con el texto `Hola` centrado en cada rectángulo. La disposición de los rectángulos se determinó de forma aleatoria. Además, se evaluaron los tiempos de ejecución tanto en un canvas de 800x600 píxeles como a pantalla completa.

Las pruebas se dividieron en tres tipos:

- **Solo JavaFX:** Esta prueba se centró exclusivamente en el uso de JavaFX para dibujar y gestionar los rectángulos, evaluando su rendimiento y capacidad de respuesta.
- **JavaScript + Java:** En esta prueba se combinó JavaScript para la parte de la interfaz gráfica con Java para el backend, con el fin de evaluar cómo estas dos tecnologías pueden trabajar juntas para manejar la carga de trabajo.
- **HTML5 Canvas + JavaScript + Java:** Esta prueba utilizó el elemento Canvas de HTML5 junto con JavaScript y Java, para explorar un enfoque híbrido que aproveche las fortalezas de cada tecnología en la gestión de gráficos y procesamiento de datos.

Las pruebas se llevaron a cabo en un portátil equipado con un procesador Intel Core™ i7-8550U de 8 cores a 1.80GHz y 8 GiB de memoria RAM. La versión de JavaFX utilizada fue la 22.0.1. Es importante destacar que, inicialmente, JavaFX estaba configurado en modo software, lo que significa que no se estaba aprovechando el potencial de la aceleración gráfica por hardware durante las primeras ejecuciones. Esta configuración inicial fue ajustada posteriormente para evaluar el impacto de diferentes opciones de aceleración gráfica en el rendimiento de las pruebas, probando diversas configuraciones para optimizar el uso de los recursos gráficos disponibles.

D.1. Pruebas de rendimiento

Las medidas obtenidas son en segundos.

D.1.1. Solo JavaFX

La Tabla D.1 muestra los resultados de las pruebas de rendimiento usando solo JavaFX.

Cantidad / Canvas	100	500	1.000	5.000	10.000	50.000	100.000	500.000	1.000.000
800x600	0.056	0.138	0.182	0.311	0.623	1.067	2.294	—	—
Completo	0.042	0.152	0.214	0.39	0.46	1.195	1.535	—	—

Tabla D.1: Resultados de pruebas de rendimiento con JavaFX

D.1.2. JavaScript + JavaFX

La Tabla D.2 muestra los resultados de las pruebas de rendimiento utilizando JavaScript junto con JavaFX.

Cantidad / Canvas	100	500	1.000	5.000	10.000	50.000	100.000	500.000	1.000.000
800x600	0.444	0.607	0.755	1.135	1.629	3.792	7.526	—	—
Completo	0.571	0.734	0.759	1.415	1.844	4.463	8.841	—	—

Tabla D.2: Resultados de pruebas de rendimiento con JavaScript + JavaFX

D.1.3. Canvas + JavaScript + JavaFX

La Tabla D.3 muestra los resultados de las pruebas de rendimiento utilizando Canvas junto con JavaScript y JavaFX.

Cantidad / Canvas	100	500	1.000	5.000	10.000	50.000	100.000	500.000	1.000.000
800x600	0.01	0.023	0.029	0.287	0.396	0.97	1.704	5.985	11.132
Completo	0.001	0.007	0.015	0.064	0.11	1.071	1.608	5.593	9.637

Tabla D.3: Resultados de pruebas de rendimiento con Canvas + JavaScript + JavaFX

Tras realizar las pruebas, la decisión tomada fue comenzar por una migración solo a JavaFX y plantear como una mejora a futuro, la migración de la visualización del grafo a canvas de HTML5 + JavaScript.

D.2. Pruebas de rendimiento avanzadas

Después de realizar las pruebas iniciales para determinar las tecnologías más adecuadas, y con un mejor entendimiento de estas tecnologías, se ha explorado la posibilidad de mejorar el rendimiento del sistema mediante el ajuste de parámetros y argumentos específicos al momento de lanzar el programa. Estas pruebas avanzadas tienen como objetivo optimizar el tiempo de ejecución y la eficiencia del sistema.

A continuación, se presentan los resultados de las pruebas de la sección D.1.1 (*"Solo JavaFX")* realizadas con estos nuevos argumentos, manteniendo las mismas condiciones y variaciones de tamaño para asegurar la correspondencia con las pruebas iniciales.

Dichas configuraciones son:

1. **-Dprism.forceGPU:** fuerza a que utilice la gráfica
2. **-Dprism.order= :** elige el pipeline gráfico que usa prism
    a) **d3d:** Direct3d (Windows)
    b) **es2:** openGL (Linux)
    c) **sw:** software1
    d) **j2d:** Java 2D API (software)
    e) **null3D:** parece que es un dummy

En estas pruebas realizadas, no se incluyen tablas de resultados detallados debido a que no se observaron mejoras significativas en el rendimiento en términos de tiempo de ejecución. Sin embargo, es importante destacar que sí se notaron mejoras en la estabilidad del sistema en OntView 2.0 al ajustar los parámetros y argumentos específicos.

Un aspecto clave observado fue que al forzar el uso de la tarjeta gráfica mediante el argumento -Dprism.forceGPU, el rendimiento general de OntView 2.0 mejoró. En particular, si se utiliza un sistema operativo Windows, el uso del pipeline gráfico d3d (Direct3D) demostró ser significativamente más eficiente, proporcionando un mejor rendimiento en comparación con otras configuraciones.

Por lo tanto, aunque las pruebas avanzadas no muestran mejoras destacables en términos de tiempo de ejecución, la optimización a nivel de estabilidad y la configuración gráfica adecuada sí han contribuido a un mejor funcionamiento general del sistema en OntView 2.0.

E. Ontología en OWL

Este anexo contiene la ontología completa de *proyectos* escrita en lenguaje OWL.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY proyectos "http://sid.cps.unizar.es/ontology/proyectos.owl#" >
    ]>
<rdf:RDF xmlns="http://sid.cps.unizar.es/ontology/proyectos.owl#"
     xml:base="http://sid.cps.unizar.es/ontology/proyectos.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:proyectos="http://sid.cps.unizar.es/ontology/proyectos.owl#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <owl:Ontology rdf:about=""/>



    <!-- ///////////////////////////////////////////////////////////////////////
    // Object Properties
    ///////////////////////////////////////////////////////////////////////-->
    <!-- http://sid.cps.unizar.es/ontology/proyectos.owl#jefe -->
    <owl:ObjectProperty rdf:about="#jefe">
        <rdfs:range rdf:resource="#jefes"/>
        <rdfs:subPropertyOf rdf:resource="#miembros"/>
    </owl:ObjectProperty>
    <!-- http://sid.cps.unizar.es/ontology/proyectos.owl#miembros -->
    <owl:ObjectProperty rdf:about="#miembros">
        <rdfs:range rdf:resource="#personas"/>
        <rdfs:domain rdf:resource="#proyectos"/>
    </owl:ObjectProperty>

```

```
<!-- ///////////////////////////////////////////////////////////////////////
// Data properties
///////////////////////////////////////////////////////////////////////////////-->
<!-- http://sid.cps.unizar.es/ontology/proyectos.owl#nombre -->
<owl:DatatypeProperty rdf:about="#nombre">
    <rdfs:domain rdf:resource="#personas"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<!-- http://sid.cps.unizar.es/ontology/proyectos.owl#ocupacion -->
<owl:DatatypeProperty rdf:about="#ocupacion">
    <rdfs:domain rdf:resource="#personas"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<!-- http://sid.cps.unizar.es/ontology/proyectos.owl#titulo -->
<owl:DatatypeProperty rdf:about="#titulo">
    <rdfs:domain rdf:resource="#proyectos"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<!-- ///////////////////////////////////////////////////////////////////////
// Classes
///////////////////////////////////////////////////////////////////////////////-->
<!-- http://sid.cps.unizar.es/ontology/proyectos.owl#jefes -->
<owl:Class rdf:about="#jefes">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="#personas"/>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#ocupacion"/>
                    <owl:hasValue>jefe</owl:hasValue>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
</owl:Class>
<!-- http://sid.cps.unizar.es/ontology/proyectos.owl#personas -->
<owl:Class rdf:about="#personas"/>
<!-- http://sid.cps.unizar.es/ontology/proyectos.owl#proyectos -->
<owl:Class rdf:about="#proyectos"/>
<!-- http://sid.cps.unizar.es/ontology/proyectos.owl#superpro -->
<owl:Class rdf:about="#superpro">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="#proyectos"/>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#miembros"/>
```

```
84                          <owl:onClass rdf:resource="#personas"/>
85                          <owl:minQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">3
86                          </owl:minQualifiedCardinality>
87                      </owl:Restriction>
88                  </owl:intersectionOf>
89              </owl:Class>
90          </owl:equivalentClass>
91      </owl:Class>
92
93      <!-- ///////////////////////////////////////////////////////////////////////
94      // Individuals
95      ///////////////////////////////////////////////////////////////////////-->
96      <!-- http://sid.cps.unizar.es/ontology/proyectos.owl#aims -->
97      <owl:Thing rdf:about="#aims">
98          <rdf:type rdf:resource="#proyectos"/>
99          <titulo rdf:datatype="&xsd;string">aims</titulo>
100         <miembros rdf:resource="#edu"/>
101         <miembros rdf:resource="#josemi"/>
102         <miembros rdf:resource="#josito"/>
103     </owl:Thing>
104     <!-- http://sid.cps.unizar.es/ontology/proyectos.owl#edu -->
105     <owl:Thing rdf:about="#edu">
106         <rdf:type rdf:resource="#personas"/>
107         <nombre rdf:datatype="&xsd;string">edu</nombre>
108         <ocupacion rdf:datatype="&xsd;string">jefe</ocupacion>
109     </owl:Thing>
110     <!-- http://sid.cps.unizar.es/ontology/proyectos.owl#jonni -->
111     <owl:Thing rdf:about="#jonni">
112         <rdf:type rdf:resource="#personas"/>
113         <ocupacion rdf:datatype="&xsd;string">currito</ocupacion>
114         <nombre rdf:datatype="&xsd;string">jonni</nombre>
115     </owl:Thing>
116     <!-- http://sid.cps.unizar.es/ontology/proyectos.owl#josemi -->
117     <personas rdf:about="#josemi">
118         <rdf:type rdf:resource="&owl;Thing"/>
119         <nombre rdf:datatype="&xsd;string">josemi</nombre>
120         <ocupacion rdf:datatype="&xsd;string">tirano</ocupacion>
121     </personas>
122     <!-- http://sid.cps.unizar.es/ontology/proyectos.owl#josito -->
123     <owl:Thing rdf:about="#josito">
124         <rdf:type rdf:resource="#personas"/>
125         <ocupacion rdf:datatype="&xsd;string">currito</ocupacion>
126         <nombre rdf:datatype="&xsd;string">josito</nombre>
127     </owl:Thing>
128     <!-- http://sid.cps.unizar.es/ontology/proyectos.owl#nestor -->
129     <owl:Thing rdf:about="#nestor">
130         <rdf:type rdf:resource="#personas"/>
131         <ocupacion rdf:datatype="&xsd;string">especialista</ocupacion>
```

```
132         <nombre rdf:datatype="&xsd;string">nestor</nombre>
133     </owl:Thing>
134 
135     <!-- ///////////////////////////////////////////////////////////////////////
136     // General axioms
137     ///////////////////////////////////////////////////////////////////////-->
138     <rdf:Description>
139         <rdf:type rdf:resource="&owl;AllDifferent"/>
140         <owl:distinctMembers rdf:parseType="Collection">
141             <rdf:Description rdf:about="#josemi"/>
142             <rdf:Description rdf:about="#josito"/>
143             <rdf:Description rdf:about="#jonni"/>
144             <rdf:Description rdf:about="#nestor"/>
145             <rdf:Description rdf:about="#aims"/>
146             <rdf:Description rdf:about="#edu"/>
147         </owl:distinctMembers>
148     </rdf:Description>
149 </rdf:RDF>
```

F. Conceptos relacionados con JavaFX

JavaFX [20] es un conjunto de paquetes de gráficos y medios que permite a los desarrolladores diseñar, crear, probar, depurar y desplegar aplicaciones cliente que funcionan de manera consistente en diferentes plataformas [39].

JavaFX está escrito como una API de Java, lo que permite que las aplicaciones JavaFX puedan utilizar cualquier API de Java para acceder a capacidades del sistema y conectarse a aplicaciones middleware.

La arquitectura de JavaFX puede verse en la Figura F.1 y los principales elementos que se van a emplear son:

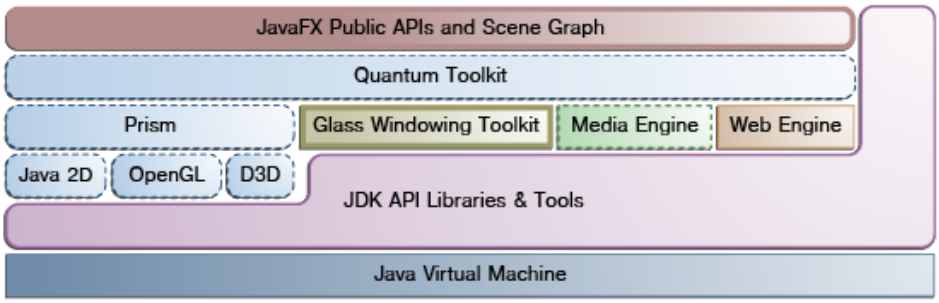

Figura F.1: Arquitectura de JavaFX

Scene Graph El grafo de escenas de JavaFX, mostrado como parte de la capa superior en la Figura F.1, es el punto de partida para construir una aplicación JavaFX. Es un árbol jerárquico de nodos que representa todos los elementos visuales de la interfaz de usuario de la aplicación. Puede manejar entradas y ser renderizado [40].

Canvas La API de Canvas de JavaFX proporciona una superficie de dibujo personalizada que permite dibujar gráficos directamente en un área de la escena. Está definida por las clases Canvas y GraphicsContext en el paquete *javafx.scene.canvas*. El uso de esta API implica crear un objeto Canvas, obtener su GraphicsContext y realizar operaciones de dibujo para renderizar formas personalizadas en la pantalla. Dado que Canvas es una subclase de Node, puede ser utilizado en el grafo de escenas de JavaFX [41].