Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Máster Universitario en Inteligencia Artificial

Trabajo Fin de Máster

Generación de Catálogo de Ontologías: *Ágora*

Autora: Silvia Cristina Franco Morán

Tutora: María Poveda Villalón

Madrid, julio 2024

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario en Inteligencia Artificial
Título: Generación de Catálogo de Ontologías *Ágora*
Julio 2024

Autora: Silvia Cristina Franco Morán

Tutora:
María Poveda Villalón
Departamento de Inteligencia Artificial
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Este Trabajo de Fin de Máster tiene como objetivo el desarrollo de una aplicación web orientada al catálogo y reutilización de ontologías. El sistema consiste en una herramienta que permite buscar, explorar y reutilizar ontologías, especialmente en la etapa de diseño ontológico. Respecto a otros catálogos actuales, que centran sus mecanismos de reutilización en el análisis del código de las ontologías (por ejemplo, en formato OWL), esta herramienta introduce como novedad principal la posibilidad de reutilizar ontologías basándose en sus diagramas de conceptualización. Este enfoque permite a los usuarios identificar y aplicar patrones, facilitando así el diseño de nuevas ontologías a partir de conceptualizaciones existentes.

La búsqueda en este catálogo utiliza distintas metodologías de Inteligencia Artificial que hacen posible que la búsqueda sea más precisa por incluir técnicas de procesamiento del lenguaje natural.

Esta herramienta es útil especialmente en la etapa de diseño de ontologías, ya que permite reutilizar patrones (*patterns)* del diagrama de conceptualización de la ontología para poder aplicarlos en nuevas conceptualizaciones.

El trabajo comienza con un análisis del estado del arte, que permite identificar las necesidades actuales en el contexto de la ingeniería ontológica, se estudian otras herramientas similares a Ágora y el contexto del desarrollo de la aplicación.

Al mismo tiempo que se analiza el contexto y entendiendo las necesidades que puede tener el nuevo sistema, se comenzó con la fase de diseño de la aplicación. Por un lado, se decidió la estructura que tendría el directorio de GitHub [1] en el que se almacenan las ontologías que componen el catálogo. Este repositorio permite también la contribución abierta de usuarios que deseen aportar sus ontologías para ser incluidas en el catálogo.

En la fase de diseño, se elaboró un árbol de navegación web y se desarrollaron dos prototipos: uno de baja fidelidad, que tuvo dos versiones debido a la necesidad de hacer algunas correcciones respecto de la versión inicial, y uno de alta fidelidad, que fue la base para la implementación de la interfaz final.

Una vez validados los prototipos, se procedió al diseño e implementación de la búsqueda semántica, así como al desarrollo completo de la aplicación web.

Finalmente, se han planteado unas líneas futuras a seguir para la mejora del sistema. El resultado final es un nuevo sistema de catálogo de ontologías que permite una búsqueda más precisa y completa de ontologías, así como la reutilización de estas, ayudando en su labor a los usuarios que deseen desarrollar sus propias ontologías.

[1] https://github.com/

Abstract

his Master's Thesis aims to develop a web application focused on the cataloguing and reuse of ontologies. The system provides a tool that enables users to search for, explore, and reuse ontologies, particularly during the ontology design phase. Unlike current catalogues, which focus their reuse mechanisms on analysing the ontology code (e.g., in OWL format), this tool introduces as a main novelty the ability to reuse ontologies based on their conceptualization diagrams. This approach allows users to identify and apply patterns, thus facilitating the design of new ontologies from existing conceptualizations. The catalogue's search functionality employs various Artificial Intelligence methodologies, making the search more accurate by incorporating Natural Language Processing techniques.

This tool is especially useful during the ontology design stage, as it allows for the reuse of patterns from the ontology conceptualization diagram, which can then be applied to new conceptualizations.

The work begins with a state-of-the-art analysis to identify current needs in the context of ontological engineering. It includes a study of tools similar to Agora and the broader context in which the application was developed.

Simultaneously with the contextual analysis and understanding of the potential system requirements, the design phase of the application was initiated. On one hand, the structure of the GitHub repository was defined. This repository stores the ontologies that make up the catalogue and also allows for open contributions from users who wish to submit their ontologies for inclusion.

During the design phase, a website navigation tree was created, and two prototypes were developed: a low-fidelity prototype—produced in two versions due to the need for corrections to the initial version—and a high-fidelity prototype, which served as the foundation for the final interface implementation.

Once the prototypes were validated, the semantic search functionality was designed and implemented, followed by the full development of the web application.

Finally, future development directions were proposed to enhance the system. The end result is a new ontology cataloguing system that enables more precise and comprehensive ontology searches, as well as their reuse, assisting users who wish to develop their own ontologies.

Tabla de contenidos

1 Introducción 2
1.1 Motivación 2
1.2 Objetivos 3
2 Estado del arte 4
2.1 Ontologías 4
2.2 Ingeniería ontológica 6
Especificación 6
Conceptualización 7
Formalización 7
Implementación 7
Evaluación y validación 8
Publicación y reutilización 9
Mantenimiento y evolución 9
2.3 Catálogos de ontologías 10
2.3.1 Linked Open Vocabularies (LOV) 11
2.3.2 BioPortal 12
2.3.3 Agroportal 13
2.3.4 OntoHub 14
2.4 Chowlk. 15
2.4.1 Patrones Ontológicos en Chowlk 15
3 Fundamento teórico. 17
3.1 Representaciones vectoriales. 17
3.2 Modelos de generación de *embeddings* 18
3.3 Medidas de similitud en espacios vectoriales 18
3.4 Bases de Datos Vectoriales. 19
3.5 Aplicación en ingeniería ontológica 20
4 Metodología 21
4.1 Estructura del repositorio de ontologías Agora-ontos 21
4.2 Obtención de ontologías para el catálogo 23
4.3 Mecanismo de Contribución para Usuarios 24
4.4 Procesamiento del repositorio de ontologías 25
4.5 Construcción de la base de datos vectorial para la búsqueda semántica. 28
4.6 Mecanismo de búsqueda y presentación de resultados en el catálogo 30
4.6.1 Gestión de consultas en el catálogo 31
4.6.2 Gestión de resultados de las consultas 31
5 Diseño 33

5.1 Árbol web 33
5.2 Prototipado 34
5.2.1 Prototipo de baja fidelidad 34
5.2.1.1 Inicio 35
5.2.1.2 Resultados de la búsqueda 39
5.2.1.3 Ontología 42
5.2.2 Prototipo de alta fidelidad 46
5.2.2.1 Catálogo 46
5.2.2.2 Ontología 47
5.2.2.3 Resultados de la búsqueda 49
5.2.2.4 Patrones 51
6 Desarrollo 53
6.1 Entorno de desarrollo 53
6.2 Lenguajes 53
6.2.1 Python 54
6.3 Librerías y *frameworks* 54
6.3.1 Flask 55
6.3.2 FastAPI 55
6.4 Otras herramientas 56
6.4.1 Bootstrap 56
7 Evaluación y resultados 57
7.1 Pruebas de Desarrollo Internas 57
Durante todo proceso de desarrollo, se llevaron a cabo paralelamente pruebas para asegurar la calidad y el correcto funcionamiento de cada componente del sistema Ágora. Estas pruebas se centraron en: 57
7.2 Validación con usuarios expertos. 58
8 Conclusiones y líneas futuras 60
8.1 Conclusiones 60
8.2 Líneas futuras 61
9 Bibliografía 62
10 Anexos 65
10.1 Anexo I. Instrucciones de contribución al repositorio Agora-ontos (Contributing.md) 65

Tabla de Ilustraciones

Ilustración 1. Estructura del repositorio agora-ontos 22
Ilustración 2. Diagrama de flujo del procesamiento automático del repositorio agora-ontos 26
Ilustración 3. Árbol web 34
Ilustración 4. Pantalla de inicio de la Versión 1 del prototipo de baja fidelidad 36
Ilustración 5. Pantalla de inicio de la Versión 2 del prototipo de baja fidelidad 38
Ilustración 6. Pantalla de resultados de la búsqueda de la Versión 1 del prototipo de baja fidelidad 39
Ilustración 7. Pantalla de previsualización de la ontología resultado de la búsqueda de la Versión 1 del prototipo de baja fidelidad 40
Ilustración 8. Pantalla de resultados de la búsqueda de la Versión 2 del prototipo de baja fidelidad 41
Ilustración 9. Pantalla de ontología de la Versión 1 del prototipo de baja fidelidad 43
Ilustración 10. Vista expandida de un patrón de ontología de la Versión 1 del prototipo de baja fidelidad. 44
Ilustración 11. Vista expandida de un patrón de ontología de la Versión 2 del prototipo de baja fidelidad 45
Ilustración 12. Pantalla de inicio del catálogo del prototipo de alta fidelidad . 47
Ilustración 13. Pantalla de ontología del prototipo de alta fidelidad 48
Ilustración 14. Pantalla de reutilización de la ontología del prototipo de alta fidelidad 49
Ilustración 15. Pantalla de resultados de la búsqueda Versión 1 del prototipo de alta fidelidad 50
Ilustración 16. Pantalla de resultados de la búsqueda Versión 2 del prototipo de alta fidelidad 51
Ilustración 17. Vista parcial de los patrones de la ontología del prototipo de alta fidelidad 52

1 Introducción

En el contexto de la Web Semántica y la Inteligencia Artificial, las ontologías se han consolidado como herramientas fundamentales para representar el conocimiento de forma formal y explícita [1]. Sin embargo, la efectividad las ontologías como recursos no radica únicamente en su diseño, sino también en su capacidad para ser descubierta, comprendida y reutilizada por otros. En este contexto, disponer de mecanismos eficientes para descubrir y consultar ontologías es fundamental para fomentar la colaboración, duplicidades y acelerar el desarrollo de nuevas ontologías,

Este Trabajo de Fin de Máster aborda el desarrollo del sistema Ágora, una aplicación web concebida como un catálogo de ontologías, que busca facilitar su búsqueda y reutilización. Además de centralizar y organizar ontologías expresadas mediante la notación Chowlk [2], el sistema incorpora funcionalidades clave como la reutilización de patrones, la visualización interactiva de diagramas, búsqueda semántica basada en *embeddings* y un mecanismo de contribución abierto basado en GitHub.

A lo largo del documento se detalla el proceso de desarrollo de Ágora, desde el análisis del estado del arte hasta el diseño, prototipado y la implementación de sus funcionalidades clave. El resultado es una herramienta que no solo facilita el descubrimiento y la reutilización de ontologías, sino que también fomenta las buenas prácticas en la ingeniería ontológica.

1.1 Motivación

La motivación principal de este trabajo surge de la propuesta de Chowlk, desarrollado en 2022 por el grupo Ontology Engineering Group[2] (OEG) de la Universidad Politécnica de Madrid. Esta herramienta facilita la transformación de diagramas de conceptualización ontológica, expresados mediante una notación visual basada en UML, en ontologías formales codificadas en OWL

Esta herramienta supone un avance significativo en el campo de la ingeniería ontológica, ya que supone una reducción significativa de errores durante la transición entre la conceptualización y la implementación en el proceso de creación de ontologías. Sin embargo, su uso queda limitado al ámbito individual de proyectos concretos, ya que, a pesar de contar con una extensa documentación y ejemplos de otras ontologías, no dispone de una infraestructura abierta que permita publicar, compartir o consultar los modelos generados con esta notación.

También se observa que los catálogos de ontologías existentes no contemplan la indexación de recursos gráficos, como los diagramas de conceptualización que representan visualmente la estructura y lógica de una ontología. Esto limita el potencial de reutilización, ya que la mayoría de las herramientas actuales se centran exclusivamente en el análisis del código de las ontologías (por ejemplo, en OWL), sin considerar sus representaciones gráficas. En este contexto, la integración Ágora no solo abre nuevas posibilidades para realizar búsquedas más avanzadas, sino que también permite indexar y explorar estas

[2] https://oeg.fi.upm.es/

conceptualizaciones visuales, facilitando una recuperación más intuitiva y contextualizada de patrones ontológicos reutilizables.

En este contexto, se identifica una necesidad de visibilizar, documentar y facilitar la reutilización de ontologías conceptualizadas con Chowlk. La creación de un catálogo público permite, no solo centralizar estos recursos ontológicos, sino también fomentar la colaboración en el desarrollo ontológico.

Por ello, el presente trabajo se plantea como una respuesta a esta doble necesidad: por un lado, crear una herramienta accesible y funcional que facilite el uso práctico de Chowlk y la reutilización de ontologías, y por otro lado, explorar el valor añadido que aporta la inteligencia artificial en la búsqueda y gestión de conocimiento ontológico.

Las razones expuestas anteriormente han sido las principales motivaciones para realizar este trabajo, por ello en los siguientes capítulos se mostrará cómo se desarrollado la solución a dichas cuestiones.

1.2 Objetivos

El objetivo principal de este Trabajo de Fin de Máster es la creación de una aplicación web, Ágora, concebida como un catálogo de ontologías. Esta herramienta permitirá visualizar y consultar ontologías que incorporen diagramas de conceptualización basados en la notación Chowlk, facilitando así su interpretación, reutilización y documentación.

Para alcanzar este objetivo general, se han definido los siguientes objetivos específicos:

- Analizar el estado del arte y las soluciones existentes relacionadas con catálogos de ontologías, identificando sus aportaciones y limitaciones.

- Diseñar e implementar un repositorio público que permita a los usuarios aportar sus ontologías para ser incluidas en el catálogo.

- Diseñar e implementar Ágora como aplicación web, integrando tanto el componente de visualización de ontologías como sus diagramas de conceptualización.

- Incorporar un sistema de búsqueda asistida por Inteligencia Artificial en el catálogo de ontologías, basado en técnicas de *embeddings* y recuperación semántica mediante bases de datos vectoriales, que permita realizar consultas más precisas y relevantes sobre las ontologías registradas.

Estos objetivos buscan como resultado, no solo la creación de una herramienta funcional, sino también, promover buenas prácticas y la reutilización en la ingeniería ontológica y explorar el uso de técnicas de IA en la recuperación de información ontológica.

2 Estado del arte

Antes de ahondar en el desarrollo del sistema, es importante revisar el estado del arte de la cuestión. Tratándose de un campo tan dinámico la inteligencia artificial es crucial situar el proyecto en su contexto de forma que nos permita y facilite el análisis y la identificación de las lagunas existentes en el campo que nos motivan a la realización de este proyecto. Así como, para la correcta comprensión del sistema es importante tener claras algunas nociones básicas de la ingeniería ontológica.

2.1 Ontologías

Dado que el objetivo principal del proyecto es la generación de un catálogo de ontologías, es crucial entender qué es una ontología. Aunque definir con exactitud qué es una ontología no es posible ya que no existe un consenso.

¿Qué es una ontología?

El término ontología procede del griego y está formado por “ontos” (ὄντος) que significa 'ente, ser'; y “lógos” (λόγος) cuyo significado es 'ciencia, estudio, teoría, razonamiento', por lo que se podría traducir como ‘estudio del ser’. En el ámbito de la filosofía, se trata de una rama de la metafísica que estudia la naturaleza del ser, el ente, la existencia y la realidad. Pretende dar una explicación semántica de la existencia mediante una explicación formal y explícita de una conceptualización compartida.

“El análisis ontológico aclara la estructura del conocimiento. Dado un dominio, su ontología forma el corazón de cualquier sistema de representación del conocimiento para ese dominio. Sin ontologías, o las conceptualizaciones que subyacen al conocimiento, no puede haber un vocabulario para representar el conocimiento … Las ontologías permiten compartir conocimientos". [3]

Una ontología en el ámbito de la informática también es una representación del conocimiento. En el contexto de la Web Semántica y la inteligencia artificial:

“Una ontología es una especificación formal y explícita de una conceptualización compartida” [3].

Esta definición establece que una ontología es una conceptualización es decir, una abstracción de una entidad del mundo real. Es explícita, formal y compartida ya que se trata de una conceptualización clara y exacta, creada por un conjunto de expertos en el dominio y con una estructura y componentes determinados que forman una formalización lógica que permita que las máquinas puedan interpretarla y extraer información de ella. Para ello no solo establecen términos y relaciones, sino también restricciones, equivalencias semánticas, axiomas y reglas.

Las ontologías incluyen información sobre el significado y restricciones sobre cómo aplicarlas de forma lógica y coherente. Ayudan a compartir el conocimiento, codificándolo y definiendo un conjunto de normas representacionales que son relaciones existentes entre los conceptos de conocimiento. Establecen un marco común o un “vocabulario” que reduce el número de ambigüedades a la hora de representar el conocimiento.

Hoy en día las ontologías no son sólo teóricas, si no que en el ámbito de la informática han obtenido un rol específico para la inteligencia artificial, la lingüística computacional y para la teoría de bases de datos. Computacionalmente, una ontología puede verse como un grafo orientado etiquetado en el que los nodos corresponden a conceptos o entidades, y las aristas, a las relaciones semánticas entre ellas. Esta visión se alinea con la implementación de ontologías en formato RDF (Resource Description Framework) o el lenguaje OWL (Web Ontology Language) definido por el W3C, que permite expresar no solo estructuras de datos sino también reglas complejas de inferencia a través de lógica descriptiva.

Elementos de una ontología

Las ontologías se construyen a partir de un conjunto de elementos ontológicos fundamentales que permiten representar estructuradamente y modelar el conocimiento de un dominio concreto. Estos elementos forman la base semántica sobre la que se construye la ontología y permiten que su análisis y reutilización, así como que el conocimiento sea interpretable por sistemas computacionales.

Los elementos primarios de una ontología incluyen:

1. **Clases** o conceptos. Son los elementos mediante los cuales de organiza el conocimiento, representan categorías o conjuntos de individuos que comparten características comunes en un dominio.

2. **Propiedades** (Properties) representan las relaciones entre clases o entre clases e instancias, pueden tener atributos propios. Pueden ser “de objeto” o “de datos”. Las ***Object Properties*** describen las relaciones entre clases y tienen un dominio, clase de la que parte la relación; y un rango, clase a la que llega la relación. Por otro lado, las ***Data Properties*** representan atributos de instancias de clases que tienen como valor un dato literal, y su dominio y rango se especifican por el tipo de dato.

3. **Instancias** o individuos. Son elementos específicos que pertenecen a una o más clases. Son las "cosas" concretas (o hipotéticos) del mundo real que la ontología describe.

4. **Axiomas.** Declaraciones fundamentales y restricciones propias del dominio modelado que definen la semántica formal de la ontología. Establecen restricciones estructurales y semánticas que regulan la interacción entre las entidades del modelo y constituyen la base para el razonamiento automático. Los tipos más comunes son:

   - **De Dominio y Rango.** Definen las clases de sujetos (dominio) y objetos (rango) válidos para una propiedad.

   - **De equivalencia** (*owl:equivalentClass, owl:equivalentProperty*) Indican que dos clases o propiedades tienen el mismo significado, es decir, que son semántica y lógicamente iguales.

   - **De Disyunción** o axiomas de inconsistencia *(owl:disjointWith).* Clases que no pueden tener ningún individuo común.

- **Restricciones lógicas de propiedades**. Permiten establecer condiciones más específicas sobre las propiedades. Existen las propiedades **transitivas** (si A se relaciona con B, y B con C, entonces A se relaciona con C), **simétricas** (si A se relaciona con B, entonces B también se relaciona con A), **funcionales** (aseguran que una propiedad tiene un único valor para un individuo dado), e **inversas** (que definen la relación recíproca de una propiedad).

5. **Otros elementos.** Además de los elementos fundamentales, mencionados anteriormente, para cumplir con las buenas prácticas de publicación (FAIR *principles*) [4] las ontologías deben incluir además los siguientes elementos:

- **Propiedades de anotación** (*Notation Properties*). Incorporan metadatos o información descriptiva de la ontología, no tienen impacto en la lógica de la ontología pero aportan información del autor, fechas, comentarios, descripciones que son muy importantes a la hora de facilitar la comprensión humana del modelo, la documentación y el procesamiento con herramientas como; en este caso para que la herramienta, Ágora, que extrae automáticamente información como el título, la descripción o el autor de la ontología a partir de estos metadatos. Entre las más utilizadas encontramos*: rdfs:label* (nombres legibles), *rdfs:comment* (descripciones), *dc:creator* (autores), *dc:date* (fecha), *dc:license* (licencia), *owl:versionInfo*, etc.

- ***Namespaces.*** Consisten en espacios de nombres en forma de URIs que identifican un conjunto de nombres (clases, propiedades) y evitan colisiones de nombres a la hora de combinar ontologías. Para su gestión, se suelen utilizar las anotaciones*: vann:preferredNamespacePrefix* y *vann:preferredNamespaceUri*.

2.2 Ingeniería ontológica

La ingeniería ontológica es el área dentro de la inteligencia artificial simbólica y la Web Semántica que estudia y define proceso de diseño, construcción, mantenimiento y reutilización de ontologías. [29] Esta disciplina abarca todo el proceso de modelado de conocimiento no sólo estableciendo vocabularios de conceptos de un dominio, sino estableciendo también las entidades, relaciones, restricciones, dominio, etc.

El desarrollo de una ontología sigue una serie de fases que han de seguirse para que la ontología sea útil, reutilizable y mantenible. Estas fases pueden variar según la metodología que se decida seguir para su desarrollo, pero comúnmente son las que se explican a continuación:

Especificación

Se trata de la fase inicial de desarrollo de la ontología. En esta fase se busca establecer los objetivos de ésta, qué se espera de la ontología y para qué se va a utilizar, quiénes la van a utilizar (usuarios potenciales), así como el alcance del dominio y los requisitos funcionales y no funcionales.

En esta fase también se establecen las fuentes de conocimiento que van a utilizarse, es decir, los recursos de los que se extraerá el conocimiento que será modelado en la ontología. Estos recursos pueden ser documentación técnica

como normativas o artículos científicos, repositorios como bases de datos o incluso, ontologías ya existentes que se quieran reutilizar o extender.

Conceptualización

En esta fase, se modelan el conocimiento del dominio y sus relaciones de forma abstracta pero estructurada, sin seguir un lenguaje formal; se suelen utilizar diagramas conceptuales o tablas. En esta fase se recomienda utilizar herramientas existentes como *Chowlk* o diagramas UML. El objetivo de esta fase es definir las clases principales, las relaciones existentes entre ellas, así como sus propiedades y restricciones.

Formalización

La formalización de la ontología es el proceso de transformación de la conceptualización en una representación lógica formal y computacionalmente interpretable por sistemas inteligentes. Los lenguajes más comunes para la formalización del conocimiento son RDF (Resource Description Framework) y OWL (Web Ontology Language), ambos estandarizados por el W3C.

- **RDF** (Resource Description Framework). [5] Es un estándar que define un modelo de datos para representar información de forma estructurada. Permite describir entidades y sus relaciones mediante tripletas con la forma: sujeto, predicado, objeto; lo que concuerda con la visión de las ontologías como grafos dirigidos etiquetados, donde los nodos representan entidades y las aristas las propiedades. RDF se caracteriza por el uso de URIs (Identificadores Uniformes de Recursos) para identificar las entidades y propiedades, evitando ambigüedades semánticas y permitiendo enlazar datos de distintas fuentes. La semántica de RDF es sencilla, lo cual facilita su adopción, pero a su vez limita su capacidad expresiva, no permitiendo por ejemplo, definir restricciones complejas o realizar inferencias de alto nivel.

- **OWL** (Web Ontology Lenguage). [6] Es un lenguaje formal para la creación y definición de ontologías. Es una extensión de RDF, ya que cuenta con una mayor expresividad lógica, lo cual posibilita entre otras cosas, definir restricciones. Su semántica formal declarativa basada en lógica descriptiva, además de permitir definir clases, propiedades e instancias, permite también deducir conocimiento implícito al ofrecer mecanismos para modelar equivalencias entre clases, disyunciones, relaciones jerárquicas, restricciones cardinales, dominios y rangos de propiedades, etc. Además, permite aplicar motores de inferencia como HermiT [7] o Pellet [8], para verificar la consistencia lógica de la ontología y generar conocimiento derivado.

Implementación

La implementación consiste en, una vez la ontología ya ha sido formalizada a nivel conceptual y lógico, codificarla utilizando lenguajes y formatos específicos.

La herramienta más utilizada para la implementación de ontologías es Protégé[3], un entorno de desarrollo de código abierto que permite crear, editar, visualizar y documentar ontologías, y ofrece soporte completo para el lenguaje OWL.

En esta fase, además de la codificación de clases, propiedades y axiomas, se incorporan otros elementos como: anotaciones (descripciones, etiquetas, comentarios en diferentes idiomas), enlaces a otras ontologías (rdfs:seeAlso, owl:equivalentClass) o metadatos de autoría, licencias, versiones, etc. El resultado de esta fase es una ontología funcional, interpretable por los sistemas.

Evaluación y validación

Una vez formalizada e implementada la ontología, resulta crucial someterla a una fase de evaluación y validación semántica y lógica de la ontología para garantizar su calidad, utilidad y sostenibilidad. El objetivo de etapa es asegurarse de que la ontología satisface los requisitos para los que fue diseñada, garantizando su coherencia lógica, completitud y facilidad de uso y reutilización. [9]

Se analiza la calidad ontológica desde distintas perspectivas:

- **Sintáctica**. Se trata de una evaluación técnica que se centra en la búsqueda de errores de codificación, asegurándose de que la ontología respete la semántica del lenguaje estándar utilizado (como RDF o OWL).
- **Semántica**. Evalúa la consistencia lógica de la ontología, que ésta refleje correctamente el dominio de conocimiento que pretende modelar, y que sus clases, relaciones y axiomas sean coherentes y sin contradicciones.
- **Pragmática**. Se centra en analizar la utilidad, usabilidad y reutilización de la ontología, evaluando aspectos como su interoperabilidad, cobertura del dominio, extensibilidad y claridad.

Para llevar a cabo este proceso, existen herramientas automáticas de validación de ontologías. Es frecuente usar validadores o motores de razonamiento (*reasoners*) automáticos (como HermiT[4], Pellet[5] o FaCT++[6]) que aplican lógica descriptiva para inferir conocimiento implícito a partir de los axiomas explícitos definidos en la ontología, permitiendo detectar errores como inconsistencias, relaciones redundantes, clases no que no pueden tener ninguna instancia válida o problemas de jerarquía mal definidas.

Además, dentro de las herramientas de evaluación automática, OOPS![4] (OntOnlogy Pitfall Scanner)[7] permite detectar problemas estructurales y de diseño, mediante comprobaciones automáticas, revisa las buenas prácticas no cumplidas o errores comunes en el desarrollo ontológico basándose en un catálogo de errores denominados *pitfalls* identificados en su literatura clasificándolas por su impacto (crítico, importante, menor), que pueden ser desde errores leves como nombres mal definidos, hasta problemas graves como clases huérfanas, axiomas contradictorios o mal uso de propiedades.

---

[3] https://protege.stanford.edu/
[4] http://www.hermit-reasoner.com/
[5] https://www.w3.org/2001/sw/wiki/Pellet
[6] http://owl.cs.manchester.ac.uk/tools/fact/
[7] https://oops.linkeddata.es/

Publicación y reutilización

Una vez que la ontología ha sido evaluada y validada, es fundamental, dentro del ciclo de vida ontológico, su publicación y reutilización para hacerla accesible al público, de modo que pueda ser utilizada y extendida maximizando su impacto y alcance.

La publicación de una ontología implica hacerla accesible a la comunidad en un repositorio público, habitualmente en la web, siguiendo los principios de los datos enlazados (*Linked Data*) y las buenas prácticas de publicación semántica que facilitan su descubrimiento, comprensión y uso. Estos principios incluyen: que sean accesibles mediante URIs (*Uniform Resource Identifiers*) y estén publicadas en repositorios públicos dedicados (Como LOV [8], BioPortal [9], AgroPortal[10] o OntoHub[11]); y que cuenten con documentación clara y completa y con metadatos descriptivos [11] sobre el autor, versión, licencia, formato y documentación de uso, seguir formatos estándar (como RDF/XML, Turtle [12] o JSON-LD) para facilitar su integración y procesamiento.

La reutilización de ontologías se refiere al proceso de incorporar, adaptar o extender ontologías existentes que, al compartir una misma base conceptual, pueden intercambiar información de manera más efectiva y sin ambigüedades semánticas. Promueve la interoperabilidad semántica, evita duplicidades y permite construir nuevas ontologías de mayor calidad más rápidamente, partiendo de modelos ya existentes y validados. Esta reutilización puede realizarse extendiendo ontologías existentes, añadiéndoles nuevos elementos para cubrir aspectos más específicos; combinando módulos, formados por varios elementos, de varias ontologías para crear nuevas ontologías; o mediante mecanismos de alineación o mapeo semántico estableciendo correspondencias semánticas entre elementos de diferentes ontologías.

En el contexto de este trabajo de fin de máster, esta fase resulta de especial relevancia, ya que La herramienta propuesta, Ágora, tiene como objetivo principal el desarrollo de un catálogo de ontologías, orientado precisamente a facilitar su descubrimiento, consulta y reutilización por parte de la comunidad.

Además, el seguimiento de los principios de publicación mencionados, especialmente la incorporación de metadatos en las ontologías (como nombre o descripción), ha sido un pilar fundamental para la arquitectura de Ágora ya que, poder procesar automáticamente estos metadatos ha permitido a la herramienta catalogar de manera efectiva las ontologías.

Mantenimiento y evolución

Tras su publicación, una ontología no permanece estática o inmutable. Para garantizar su vigencia, utilidad y coherencia a lo largo de tiempo, la ontología debe seguir un proceso de evolución continua. Es fundamental que siga un proceso de adaptación a los cambios en el dominio del conocimiento que puedan

---

[8] https://lov.linkeddata.es/dataset/lov/
[9] https://www.bioontology.org/
[10] https://agroportal.lirmm.fr/
[11] https://ontohub.org/
https://github.com/ontohub/ontohub

surgir; ya que la información, las relaciones y la terminología pueden cambiar, ampliarse o incluso quedar obsoletas. Por esto, es importante que la ontología se transforme o modifique estructuralmente, o se amplie semánticamente (añadiendo nuevas clases, relaciones, axiomas, etc.) según cambien el dominio, o los requisitos. Así como que la ontología se refine ante la aparición de posibles errores, eliminando redundancias, ambigüedades o inconsistencias lógicas para garantizar que la ontología siga siendo relevante, precisa y útil.

Estos cambios o mejoras que deben llevarse a cabo en la ontología han de estar guiados de la retroalimentación (o *feedback*) de los usuarios y desarrolladores, y deben estar correctamente documentados, siguiendo algún tipo de sistema riguroso de control de versiones que asegure la trazabilidad del historial de cambios.

2.3 Catálogos de ontologías

En el contexto de la ingeniería ontológica, como se ha expuesto anteriormente, es muy importante la reutilización de los recursos ontológicos existentes, por esto, es inherente a su naturaleza, que existan catálogos de ontologías que funcionen como repositorios centralizados que incluyan y organicen ontologías ya publicadas para su búsqueda, visualización y gestión.

Estos catálogos ofrecen distintos **tipos de búsqueda** según su funcionalidad central. Pueden estar centrados en la **búsqueda por palabras clave**, basadas en términos de sus metadatos como el título, el autor, o la descripción; o incluso en la **búsqueda por términos** específicos correspondientes a elementos de las ontologías, buscando entre sus clases o propiedades, o sus relaciones, incluso algunos catálogos incluyen también una **búsqueda jerárquica** que incluye la navegación entre las jerarquías de entidades o las relaciones entre ontologías (importaciones).

Basándonos en el tipo de ontologías que los conforman, se pueden distinguir diferentes grupos de repositorios de ontologías:

- **Catálogos generales**. Recopilan ontologías de diferentes ámbitos, que incluyen desde temas generales hasta dominios más específicos. Un ejemplo representativo es Linked Open Vocabularies (LOV), que incluye vocabularios de múltiples dominios usados en la Web Semántica y Linked Data.

- **Catálogos de dominio**. Son catálogos especializados en ontologías de un determinado dominio como la biomedicina, la agricultura o el transporte, entre otros. Algunos ejemplos destacados son BioPortal, que recoge ontologías biomédicas y clínicas, y AgroPortal, centrado en ontologías del ámbito agroalimentario y medioambiental.

- **Catálogos basados en estándares.** Son catálogos que se enfocan en ontologías que siguen estándares ampliamente aceptados, como puede ser el lenguaje OWL del W3C. Son especialmente útiles cuando se buscan ontologías que se ajusten a ciertos estándares y pautas de desarrollo. Un ejemplo de este tipo es OntoHub, que da soporte a múltiples lenguajes ontológicos y estructuras formalizadas, y permite la gestión distribuida de ontologías siguiendo estándares como OWL, RDF o DOL.

Se pueden distinguir también diferentes grupos de catálogos basándonos en la forma en la que los repositorios se nutren de ontologías:

- **Catálogos colaborativos.** Se tratan de repositorios que están formados por las contribuciones de los usuarios con sus propias ontologías. Estas aportaciones se realizan de forma manual, esto, unido a que después deben pasar por un proceso de revisión y validación que puede ser manual o automático, ralentiza el proceso y hace que su contenido dependa de la proactividad de los usuarios, pero a su vez, garantiza un alta calidad en el contenido del repositorio además promueven la colaboración y la interacción entre la comunidad de desarrolladores de ontologías, fomentando el intercambio de conocimientos y la mejora continua de las ontologías a través de las revisiones y comentarios sobre las ontologías que los conforman.

- **Catálogos de rastreo** (*Crawling*). Son repositorios que realizan un rastreo web en repositorios, ya que gracias a los principios de Linked Data, las ontologías deben poder ser accedidas a través de sus URIs. Este tipo de repositorios son más escalables y por lo general, disponen de un catálogo más extenso de ontologías, pero es importante que dispongan de algún sistema de validación para identificar los archivos de ontologías.

- **Agregadores de catálogos.** Algunos catálogos están formados por la agregación de ontologías ya presentes en otros repositorios.

Es importante tener en cuenta que estás clasificaciones no son disjuntas, es decir un mismo catálogo puede contar con distintos tipos de búsqueda o pertenecer a más de unos de los grupos de repositorios mencionados anteriormente. Además, cabe destacar que algunos repositorios ontológicos cuentan con sus propias APIs de consulta semántica que permiten a otros sistemas externos consumir sus datos y servicios para integrar los catálogos.

A continuación, se hará un breve estudio sobre los catálogos ontológicos existentes más destacables, evaluando diferentes aspectos de éstos para poder discernir qué elementos son claves y cuáles son sus puntos de mejora.

2.3.1 Linked Open Vocabularies (LOV)

Linked Open Vocabularies (LOV) [13] es un catálogo especializado en vocabularios y ontologías utilizadas en la Web Semántica y Linked Data. Incluye más de 900 vocabularios que describen y enlazan datos de la web y que son tanto de dominio general como específico.

LOV se alimenta de ontologías publicadas que, mediante sugerencias de los autores a través de un formulario o por contacto directo. Estas sugerencias posteriormente pasan por una validación manual realizada por un equipo que revisa cada nueva ontología sugerida antes de incorporarla, asegurándose de que siga buenas prácticas, añade metadatos manualmente, valida que sean accesibles por URI y que sigan formatos estándar como RDF/XML o Turtle y que estén correctamente documentadas (con rdfs:label, dc:description, etc.), lo que fomenta mediante un manual de recomendaciones de metadatos. Además, garantiza la persistencia y trazabilidad del catálogo mediante tareas diarias de mantenimiento y actualización, además, mantiene las diferentes versiones de cada elemento del catálogo.

La búsqueda y navegación en este catálogo incluye un ranking de uso y visibilidad para asegurar la reutilización de vocabularios consolidados priorizando la popularidad del término y la coincidencia con etiquetas primarias. Cuenta con mecanismos de búsqueda por palabras clave que incluye autocompletado por URI o prefijo, y permite filtrar por tipo: clase, propiedad, tipos de datos, instancias, etc. Devolviendo como resultado tanto elementos de las ontologías como ontologías completas con información detallad, incluyendo un resumen, número de clases y propiedades, enlaces a su URI persistente. LOV no proporciona visualización de ontologías pero cuenta con una *API RESTful* que permite llamar a LOV mediante una solicitud *HTTP GET* que devuelve una respuesta en formato JSON que puede ser visualizada mediante el formateo con *plugins* del navegador como *JSONView* o puede ser procesada por herramientas automáticas para su posterior uso y reutilización. Esta API incluye diferentes *endpoints* con funcionalidades como */search, /autocomplete, /info*, etc.

2.3.2 BioPortal

BioPortal [14] es "el repositorio más completo de ontologías biomédicas", fue desarrollado por el National Center for Biomedical Ontology (NCBO). Incluye más de 1000 ontologías pertenecientes a en un dominio específico, abarcando diversos temas dentro de la investigación clínica, la salud y la biología, como anatomía, química, genética, enfermedades, etc. Tambien integra ontologías de otras iniciativas como el OBO Foundry.

Se alimenta de ontologías enviadas por expertos del dominio y de proyectos de investigación, permite a cualquier usuario (que esté registrado en su plataforma) subir, publicar o actualizar su ontología, El sistema conserva un historial de versiones de cada ontología y permite proporcionar metadatos como palabras clave, descripciones, versionado, datos de contacto y del autor. Además de las aportaciones de los usuarios, su equipo también agrega ontologías de fuentes externas y monitorea sus actualizaciones. Bioportal permite diferentes formatos de ontologías, los más utilizados en el ámbito de la ingeniería ontológica (OWL, RDF, OBO) y otros formatos más especializados como OWL Rich Realese Format (RRF) o LexGrid.

Su mecanismo de búsqueda es muy avanzado, permite buscar términos a través de las múltiples ontologías de las que dispone. Proporciona autocompletado y sugerencias en la búsqueda y es capaz de reconocer sinónimos, definiciones y textos asociados a conceptos. La búsqueda devuelve resultados ordenados por coincidencia y popularidad de la ontología, incluyendo un árbol de navegación jerárquico para navegar entre las diferentes clases y propiedades de la ontología seleccionada. También, proporciona un grafo visual interactivo que muestra el contexto de un término específico de forma gráfica. BioPortal destaca por la gestión de mapeos entre términos de diferentes ontologías, permitiendo navegar, crear y subir estos mapeos, estableciendo equivalencias y relaciones entre términos de ontologías distintas.

BioPortal ofrece una completa *API RESTful* pública documentada para realizar búsquedas, obtener ontologías, información sobre clases, propiedades, etc. Proporciona servicios de terminología adicionales, como el NCBO *Annotator* que, dado un texto, lo analiza y anota menciones de términos pertenecientes a las ontologías o el Ontology Recommender, que dado un testo o un conjunto de palabras clave, sugiere ontologías relevantes.

2.3.3 Agroportal

AgroPortal [12] es un repositorio de ontologías y artefactos semánticos especializado en agronomía y ciencias agroalimentarias desarrollado por un equipo liderado por el Laboratorio de Informática, Robótica y Microelectrónica de Montpellier (LIRMM) que reutiliza la infraestructura abierta del NBCO BioPortal adaptado para cubrir mejor las necesidades y formatos propios del sector agrícola, lo cual permite incorporar sus funciones de búsqueda navegación y visualización. Está dedicado a dominios agronómicos, vegetal, alimentario y de biodiversidad, cubriendo subdominios como cultivos, taxonomías de plantas y animales, ontologías de prácticas agrícolas, de medio ambiente y ecología. Este repositorio buscaba recoger todas las ontologías y vocabularios existentes de este dominio en una sola plataforma común para facilitar su búsqueda, uso y reutilización.

AgroPortal [15] permite a los usuarios subir y compartir ontologías y otros artefactos semánticos, pudiendo describir sus ontologías con metadatos relevantes. Según sus datos, AgroPortal es utilizado por más de 500 usuarios, incluye más de 230 ontologías, 74 proyectos, 2 millones de clases y 14 mil propiedades. Integra y organiza vocabularios y ontologías de distintas fuentes como la organización de las Naciones unidad para la Alimentación y la Agricultura (FAO [13]), el Grupo Consultivo sobre Investigación Agrícola Internacional (CGIAR[14]) o incluso algunas ontologías de BioPortal. Además, incorpora tesauros y vocabularios en formato SKOS. Los tesauros son vocabularios controlados y estructurados que organizan términos relacionados de manera semántica y jerárquica que permiten indexar, clasificar y recuperar información de forma coherente, así como representar, en este caso conceptos agrícolas y las relaciones entre ellos de forma flexible. Estos tesauros, generalmente, se representan mediante el estándar del W3C, SKOS (*Simple Knowledge Organization System*), un modelo de datos basado en RDF. Una de las mejoras de AgroPortal respecto de Bioportal es que, además de albergar ontologías en formato OWL/RDF, también soporta plenamente tesauros y vocabularios en formato SKOS.

Al estar basado en BioPortal, AgroPortal cuenta con funcionalidades de búsqueda similares. Permite hacer una búsqueda por términos en todas las ontologías que incluye, así como otras funcionalidades de autocompletados. Tiene opciones de filtrado muy avanzadas clasificado por ontología, categorías (como *tech, plant, nut, law,* etc), grupos (como AGBIODATA[15] u OBO-Foundry[16]), formatos (OBO, OWL, SKOS o UMLS), lenguajes (Inglés, Español, Frances y otros), niveles de formalidad (tesauros, glosarios, ontologías, taxonomías, etc), o tipos de ontología (AplicationOntology, DomainOntology, Vocabulary, etc). Además permite incluir en la búsqueda valores de propiedad *(property values*) y vistas de ontología. No sólo permite filtrar por lenguaje la búsqueda si no que también incorpora búsqueda multilingüe soportando índices multilingües y

[12] https://agroportal.lirmm.fr/
[13] https://www.fao.org/home/es
[14] https://www.cgiar.org/
[15] https://www.agbiodata.org/
[16] https://obofoundry.org/

para poder buscar y encontrar términos en diferentes idiomas ya que hay ontologías que incluyen términos en distintos idiomas. Para la visualización, AgroPortal ofrece una "Landscape view" que proporciona una perspectiva general de las ontologías. Permite explorar jerarquías de clases o conceptos y relaciones semánticas con vistas tipo árbol o gráficas, También, ofrece una visualización especializada para vocabularios en formato SKOS.

Al igual que BioPortal, AgroPortal ofrece endpoint SPARQL y una *API RESTful* completa para descargar ontologías, realizar búsquedas, listados, obtener información detallada o realizar anotaciones, etc. También incorporó otras herramientas como el servicio de anotación semántica, similar al NCBO Annotator, un recomendador de ontologías que sugiere vocabularios según el texto dado o un módulo de evaluación FAIR dentro de la plataforma que evalúa los metadatos, las URIs, licencias para determinar en qué grado las ontologías cumplen con los principios FAIR [4] de ser localizables, accesibles, interoperables y reutilizables.

2.3.4 OntoHub

OntoHub [16] es un repositorio de ontologías de código abierto desarrollado en el contexto académico por la Universidad de Bremen como parte de la iniciativa Open Ontology Repository (OOR) [17]. A diferencia de otros catálogos estudiados como BioPortal o AgroPortal, OntoHub[17] se orienta a una gestión distribuida, colaborativa y heterogénea de ontologías. Sigue un enfoque más técnico y formal, ofreciendo soporte integral a lo largo de todo el ciclo de vida de las ontologías y sin limitarse a un dominio temático concreto.

Destaca por organizar las ontologías en diferentes repositorios independientes, cada uno vinculado directamente con un repositorio Git subyacente. Esto le permite aprovechar el sistema de Git de control de versiones, ramificación, fusión de ontologías y sincronización de cambios. Los usuarios pueden subir ontologías directamente desde la web o mediante un *push* de cambios con Git, y por cada *commit* analiza los archivos y crea metadatos versionados, para mantener un historial de la evolución de cada ontología.

En cuanto a su búsqueda, OntoHub cuenta con una barra de búsqueda y una opción "*Search*" en su menú principal para ofrecer sus funcionalidades de búsqueda y exploración de repositorios, ontologías y términos dentro de ellas, como clases y relaciones, Aunque, su buscador no está tan optimizado para usuarios generales como el de LOV o BioPortal, destaca por incorporar la posibilidad de integrar motores de inferencia para distintos lenguajes ontológicos, permitiendo verificar propiedades formales como la consistencia o implicaciones semánticas dentro de las ontologías realizando pruebas lógicas sobre las ontologías gracias a la integración con herramientas de demostración. Además, permite hacer anotaciones y comentar ontologías e incorpora herramientas de evaluación automática para la validación sintáctica, análisis de buenas prácticas y pruebas de regresión, lo que permite asegura la calidad de su contenido.

[17] https://ontohub.org/

Gracias a la implementación del estándar DOL (Distributed Ontology Language) [18] del Object Management Group, (OMG[18]), OntoHub permite gestionar, mapear y conectar ontologías en distintos lenguajes, incluyendo tanto OWL o RDF, como también otros lenguajes lógicos y formales como Common Logic[19], lógica de primer orden, TPTP o lenguajes de orden superior.

OntoHub también provee una API de federación, que permite que múltiples repositorios de ontologías, incluso si no cuentan con una estructuras o lenguaje subyacente común, puedan comunicarse e intercambien información de manera estandarizada. Esta API generaliza las capacidades de integración más allá de un dominio o formato específico (como ocurre en la API de BioPortal), permitiendo así una interoperabilidad técnica avanzada y una visión unificada del conocimiento ontológico distribuido.

2.4 Chowlk.

Chowlk es un *framework*, diseñado por el grupo Ontology Engineering Group (OEG) de la Universidad Politécnica de Madrid que contribuye a 'cerrar la brecha entre el modelado conceptual visual y la implementación formal de ontologías' [2]. Permite transformar diagramas UML con una notación específica para ontologías, diseñados en herramientas como diagrams.net, directamente en ontologías OWL.

En términos prácticos, Chowlk una herramienta que permite unificar las fases de conceptualización e implementación de la ontología, etapas que tradicionalmente se realizan de forma separada. Como se ha expuesto anteriormente, la conceptualización de ontologías se suele realizar mediante diagramas que se transforman, generalmente, de forma manual a su implementación formal en lenguaje OWL utilizando editores de ontologías como Protégé. Este proceso manual es propenso a errores y a la pérdida de información.

Chowlk busca que los diagramas de conceptualización sean 'artefactos de primera clase' en el proceso de desarrollo ontológico. Para ello, se basa en tres pilares principales: una notación visual basada en UML, que extiende el perfil UML para OWL y permite representar la ontología y, por tanto, todos sus elementos; plantillas gráficas prediseñadas para diagrams.net; y un conversor automático de conceptualizaciones (en el formato XML usado de diagrams.net) a ontologías en código OWL serializadas en Turtle.

Todo esto hace que Chowlk sea una herramienta muy útil para agilizar el proceso de creación de ontologías, ayuda también a la reutilización de modelos y a reducir significativamente la probabilidad de cometer errores durante la transición de la fase de conceptualización a la formalización.

2.4.1 Patrones Ontológicos en Chowlk

Para la ingeniería ontológica, los patrones ontológicos (*Ontology Design Patterns*, ODPs) [20] representan un recurso esencial para afrontar la complejidad del proceso de diseño de ontologías. Son soluciones reutilizables y generalizadas

[18] https://www.omg.org/
[19] https://www.iso.org/standard/66249.html

que permiten abordar problemas de modelado frecuentes y conocidos durante las fases de conceptualización y formalización de ontologías. El uso de estos patrones fomenta la reutilización, la estandarización y la mejora de la calidad de los desarrollos ontológicos, además de reducir el riesgo de errores tanto conceptuales como técnicos.

Estos patrones pueden abordar aspectos muy distintos del proceso de diseño ontológico, como la representación de relaciones complejas, la organización jerárquica de conceptos o el modelado de restricciones. Entre los distintos tipos existentes, los patrones de contenido (Content ODPs) han ganado relevancia por su capacidad para resolver problemas concretos en dominios específicos. Cada patrón suele acompañarse de una descripción detallada sobre su propósito, aplicabilidad, estructura y consecuencias, lo que facilita su correcta implementación en nuevos proyectos.

En este contexto, Chowlk es una herramienta clave que facilita la incorporación de estos patrones ontológicos al proceso de diseño, gracias a su notación visual basada en UML y su integración con herramientas como draw.io (diagrams.net). Este enfoque permite representar patrones ontológicos de forma explícita y estructurada en diagramas visuales, dotándolos de una semántica clara que puede transformarse automáticamente en OWL sin pérdida de información. Chowlk proporciona plantillas gráficas prediseñadas que representan patrones comunes como: relaciones n-arias, restricciones de cardinalidad, propiedades inversas o simétricas, con conectores bidireccionales que indican la relación recíproca, o bloques de anotación y metadatos. Esta notación permite visualizar la estructura de una ontología de manera más intuitiva y agiliza su construcción y validación.

Para la herramienta Ágora, que gestiona un catálogo de ontologías basadas en esta metodología, la aplicación de ODPs a través de la notación de Chowlk cobra especial relevancia ya que mejora la calidad de las ontologías desarrolladas y fomenta la reutilización de patrones como unidades de conocimiento independientes, que pueden explorarse, adaptarse y reutilizarse en otros contextos, fortaleciendo la interoperabilidad y la coherencia semántica.

3 Fundamento teórico.

En el contexto actual de la gestión del conocimiento, caracterizado por la heterogeneidad y el gran volumen de la información, los mecanismos tradicionales de búsqueda basados en coincidencias coincidencia de palabras clave están limitados a la hora de capturar el significado contextual de los términos. Este problema se acentúa en dominios especializados como es el caso de la ingeniería ontológica, donde los conceptos no siempre se expresan de manera uniforme. Por ello, la búsqueda semántica basada en técnicas de representación vectorial ha resultado ser una solución eficaz para mejorar la precisión y relevancia en la recuperación de información

3.1 Representaciones vectoriales.

La representación vectorial del lenguaje natural es uno de los pilares fundamentales para el procesamiento semántico de texto mediante técnicas de inteligencia artificial. Se basa en la premisa de que las unidades lingüísticas (palabras, frases o documentos) pueden ser codificadas en un formato numérico que permita su interpretación y comparación computacional. Estas codificaciones corresponden a vectores numéricos en un espacio de alta dimensión, en el que la proximidad entre dichos vectores refleja una mayor similitud semántica entre ellos. Es decir, que si dos términos tienen significados similares, sus vectores estarán próximos entre sí. Esta idea se basa en la hipótesis distribucional del lenguaje, según la cual "el significado de una palabra está determinado por los contextos en los que aparece" [19].

Las representaciones vectoriales, conocidas como *embeddings,* son vectores densos de valores continuos que capturan relaciones semánticas y sintácticas entre unidades lingüísticas. [20] A diferencia de los enfoques de representación tradicionales, en los que cada palabra se trata como un token independiente, los *embeddings* no solo reducen la dimensionalidad del espacio textual, si no que codifican el significado y el contexto de los textos en la posición y dirección de su vector.

Estos vectores se generan a través de modelos de aprendizaje profundo entrenados con vastos corpus de texto, identificando patrones de coocurrencia y estructura lingüística, lo que les permite capturar y reflejar relaciones semánticas complejas que van más allá de la proximidad léxica. Los modelos de lenguaje contextualizados, como BERT, generan representaciones dinámicas que dependen del contexto completo en el que aparece cada palabra.

El uso de representaciones vectoriales ha permitido abordar computacionalmente uno de los problemas más complejos del lenguaje natural: la ambigüedad semántica y variabilidad contextual. Esta capacidad resulta especialmente útil en entornos donde el descubrimiento y la reutilización de conocimiento requieren interpretar descripciones textuales de forma automática y precisa.

Esta capacidad de codificar la semántica es lo que permite, no solo generar *embeddings* para palabras individuales, si no también frases, párrafos o incluso documentos completos, como en este caso para documentos de ontología. Lo cual posibilita que una consulta de usuario, incluso aunque no contenga palabras clave exactas, pueda recuperar ontologías que sean conceptualmente

relevantes gracias a la similitud semántica y la proximidad entre sus representaciones vectoriales.

3.2 Modelos de generación de *embeddings*

La representación vectorial de texto ha evolucionado significativamente en la última década, pasando de enfoques basados en estadísticas simples a complejos modelos neuronales de lenguaje. Los primeros avances relevantes en esta área fueron los modelos como Word2Vec [21] y GloVe [22], que permiten proyectar palabras individuales en un espacio vectorial denso. Sin embargo, eran limitados ya que generan una única representación para cada palabra, independientemente del contexto en el que aparezca. Siendo esto un problema especialmente para palabras polisémicas, ya que les correspondería un único vector que no distingue entre sus distintos significados.

Para superar esta limitación, surgieron los modelos de lenguaje contextualizados, entre los que destaca BERT (Bidirectional Encoder Representations from Transformers) [23]. BERT introduce una arquitectura basada en Transformers, que captura dependencias bidireccionales entre palabras y genera representaciones contextuales a nivel de subpalabra, palabra, frase o documento. Esto implica que la misma palabra tendrá diferentes representaciones vectoriales según el contexto en que se utilice, lo que permite una mayor precisión semántica y resuelve el problema de la polisemia. Además, permite que palabras con significados similares o incluso sinónimas, tenga representaciones vectoriales que, aunque no son idénticas, se encuentren en posiciones cercanas dentro de un espacio vectorial multidimensional, reflejando su similitud semántica.

A partir de BERT, se han desarrollado múltiples variantes y bibliotecas orientadas a tareas específicas. Entre las que destaca es Sentence Transformers [24] una biblioteca, en el ámbito de la comparación semántica de texto que adapta BERT para producir *sentence embeddings*, es decir, representaciones vectoriales de frases completas, utilizadas para tareas de recuperación de información, clasificación o detección de similitud semántica. Sentence Transformers utiliza redes siamesas y técnicas de aprendizaje contrastivo, para entrenar modelos en los que dos frases distintas se procesan en paralelo utilizando modelos de lenguaje compartidos. Durante el entrenamiento, el sistema aprende a representar frases con significados similares en puntos cercanos dentro del espacio vectorial, mientras que frases con significados distintos quedan alejadas. Gracias a esto, es posible comparar oraciones calculando la distancia entre sus vectores, por ejemplo, utilizando la similitud del coseno, con resultados muy buenos y precisos para tareas de búsqueda semántica y recuperación de información.

3.3 Medidas de similitud en espacios vectoriales

Una vez que las unidades lingüísticas han sido transformadas en vectores dentro de un espacio de representación semántica, es necesario determinar una medida que permita establecer la similitud entre ellas. En este contexto, las métricas de similitud vectorial son fundamentales para la recuperación de información, la búsqueda semántica o la clasificación de contenidos.

Una de las métricas más utilizadas es la de la similitud del coseno. Calcula el coseno del ángulo que forman dos vectores en un espacio multidimensional, sin importar su magnitud. [25] Su valor varía entre -1 y 1, donde 1 indica que los vectores son exactamente paralelos (máxima similitud), 0 que son perpendiculares (sin relación semántica) y -1 que apuntan en direcciones opuestas. En la práctica, al trabajar con embeddings derivados de modelos de lenguaje, los valores tienden a situarse en el rango entre 0 y 1.

La similitud del coseno es capaz de comparar representaciones semánticas de textos de diferente longitud o con escasa coincidencia léxica. Para los *embeddings* de texto, la dirección de un vector es el factor que realmente codifica el significado, mientras que su longitud no tiene una relevancia semántica directa por lo que, al centrarse en la orientación de los vectores y no en su tamaño, resulta especialmente eficaz para detectar relaciones semánticas profundas más allá de la coincidencia de palabras clave.

En Ágora, se utiliza la similitud del coseno como criterio para determinar la proximidad semántica entre una consulta del usuario y los elementos registrados en el catálogo de ontologías. Lo cual, permite recuperar ontologías, patrones o componentes relevantes incluso cuando el vocabulario utilizado en la consulta difiere del empleado en las descripciones de las ontologías que conforman el repositorio.

3.4 Bases de Datos Vectoriales.

Para que la búsqueda por similitud sea viable en un catálogo con un número creciente de ontologías, se necesita una infraestructura especializada para gestionar grandes volúmenes de vectores. Las bases de datos relacionales tradicionales no están optimizadas para este tipo de operaciones de búsqueda por similitud, que serían muy lentas e ineficientes debido a su alto coste computacional.

Por este motivo, se recurrió a las bases de datos vectoriales, diseñadas específicamente para realizar búsquedas por similitud en espacios de representación de alta dimensionalidad. Estas soluciones permiten encontrar elementos semánticamente cercanos mediante técnicas de búsqueda aproximada de vecinos más cercanos (Approximate Nearest Neighbors, ANN) [26], lo que permite un alto rendimiento incluso con grandes volúmenes de datos.

Entre estas bases de datos, Qdrant [27] implementa algoritmos de indexación eficientes como HNSW (Hierarchical Navigable Small World graphs) [28], lo que permite realizar búsquedas rápidas sin sacrificar precisión. Qdrant destaca por ser compatible con búsquedas basadas en similitud del coseno, una gestión eficiente de vectores con metadatos asociados a cada vector (*payload*), y una arquitectura optimizada para tareas de búsqueda semántica a gran escala.

El uso de una base de datos vectorial es fundamental para garantizar que Ágora pueda ofrecer resultados de búsqueda semántica en tiempo real, manteniendo su rendimiento incluso a medida que el catálogo de ontologías crece en tamaño y complejidad.

3.5 Aplicación en ingeniería ontológica

La ingeniería ontológica, como se ha expuesto anteriormente, tiene como objetivo representar formalmente dominios del conocimiento, generalmente mediante estructuras complejas y vocabularios especializados. En este contexto, la búsqueda semántica basada en representaciones vectoriales permite superar las limitaciones de las búsquedas textuales exactas, facilitando el descubrimiento y comparación de ontologías y sus componentes (clases, propiedades, patrones o metadatos) incluso cuando el vocabulario utilizado varia.

Ágora aplica esta aproximación para ofrecer una interfaz accesible que permite consultar ontologías completas, clases, propiedades o patrones mediante lenguaje natural. Al transformar los recursos ontológicos en vectores semánticos, el sistema puede recuperar resultados conceptualmente relevantes aunque no coincidan literalmente con los términos introducidos por el usuario. Esto habilita un acceso más intuitivo y flexible, permitiendo que tanto usuarios expertos como no especializados interactúen con el catálogo de forma eficaz, sin necesidad de conocer en detalle la terminología o la estructura del dominio modelado.

4 Metodología

4.1 Estructura del repositorio de ontologías Agora-ontos

El catálogo de ontologías de Ágora de alimenta a partir de un repositorio de GitHub (accesible desde la url: https://github.com/oeg-upm/agora-ontos) En él, los usuarios pueden subir sus propias ontologías para que formen parte del catálogo de Ágora. Se ha decidido usar un repositorio de GitHub para esta labor con el fin de ofrecer un sistema centralizado y organizado para el almacenamiento de las ontologías, así como para garantizar la facilitar la integración y reutilización de ontologías por parte de distintas organizaciones.

Se ha propuesto una estructura modular y escalable que busca organizar las ontologías de manera intuitiva, permitiendo categorizar por organización e incluir información esencial para cada ontología. Cada ontología puede alojarse de forma individual en su propia carpeta o agrupada bajo una organización específica. Buscando por un lado, la encapsulación de cada uno de los componentes de cada ontología y mantener la coherencia y facilidad de navegación.

Para cada ontología, el usuario debe indicar aportar los siguientes elementos:

- Carpeta de la ontología:
    - **Imagen logo**. Se trata de una imagen pequeña en formato .jpg o .png de tipo logo o icono que sea representativo de la ontología o la organización a la que pertenece la ontología. Sería la imagen que se incluirá en la vista general del catálogo en la tarjeta correspondiente a la ontología. En caso de tratase del logo de una organización, esta imagen puede ser común en las distintas ontologías de la organización por lo que no sería necesario incluirla una vez por cada ontología.
    - **Imagen de la conceptualización de la ontología**. Se trata de una imagen más grande, también en formato .jpg o .png que representa el diagrama visual que conceptualiza la ontología. Está imagen se puede obtener como una importación en formato imagen del archivo .XML de la ontología desde draw.io.
    - **Fichero de la conceptualización de la ontología.** Documento con el código XML correspondiente a la conceptualización de la ontología construida siguiendo la notación visual Chowlk para construir conceptualizaciones de ontologías. Este fichero es el que se utilizará para poder incrustar un *iframe* que permita editar y extraer patrones de dicha conceptualización directamente desde Ágora.
    - **Fichero OWL de la ontología**: Es el archivo principal de la ontología en formato OWL. Este fichero contiene el código de la ontología, es fundamental para la extracción de clases, propiedades y metadatos y para la indexación en el catálogo.
    - **Fichero XML de patrones:** este fichero es opcional, si la ontología incorpora patrones de diseño, este archivo contendrá la representación XML de la librería de patrones, exportada directamente desde Draw.io. No se trata del mismo archivo que el XML de la conceptualización de la ontología ya que éste no define

una ontología si no que se obtiene de la exportación de los patrones del bloc de notas de draw.io. Estos patrones pueden tener un título que aparecerá en la interfaz de Ágora para poder identificar cada patrón de la librería en caso de que cuenten con más de un patrón.

A continuación, se presenta un diagrama que ilustra la estructura propuesta para el repositorio de ontologías, diseñado para una clara organización y fácil acceso. En este, en morado representa el repositorio de GitHub. Las carpetas se indican en azul (y en cursiva), utilizadas para agrupar ontologías individualmente o por organizaciones, así como recursos como patrones. Los archivos opcionales, como logos y bibliotecas de patrones, se muestran en amarillo (identificados con un asterisco), siendo altamente recomendables para enriquecer el catálogo. Finalmente, los archivos obligatorios, esenciales para la correcta indexación y representación de cada ontología (como el diagrama y el archivo OWL), se destacan en verde y subrayados.

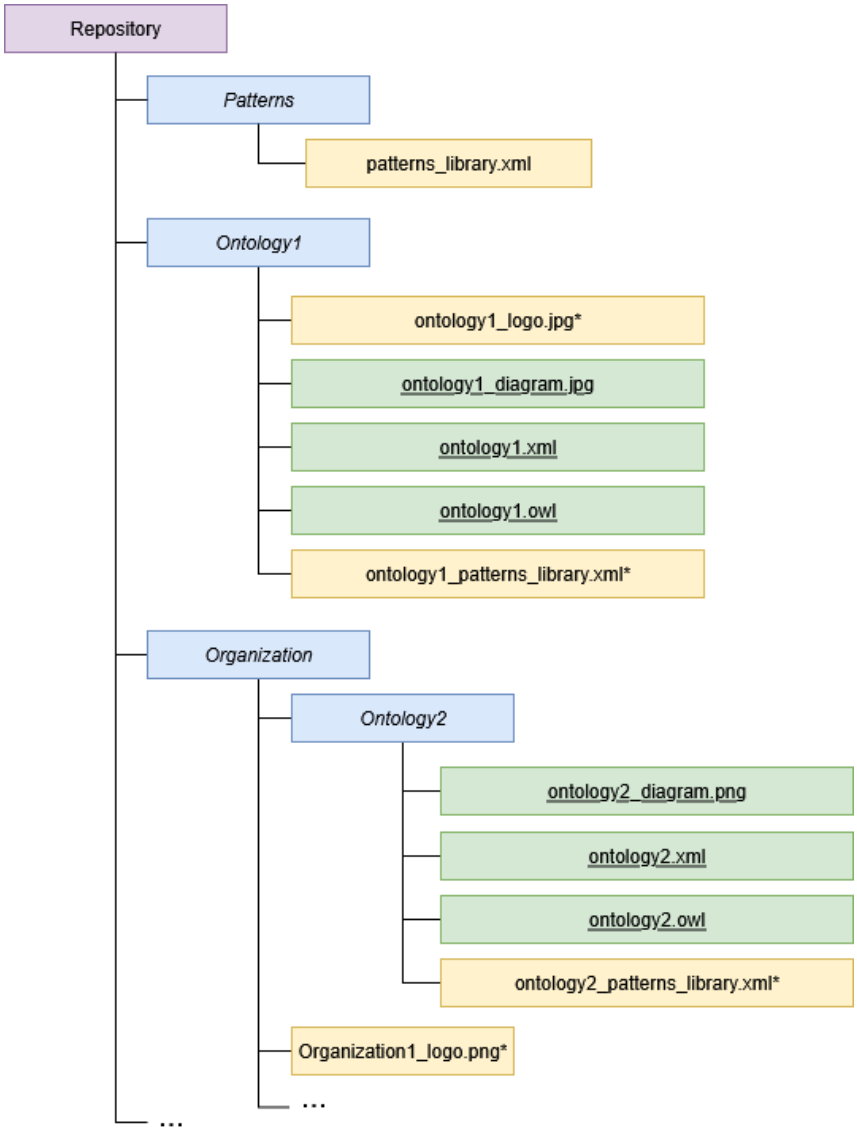

Ilustración 1. Estructura del repositorio agora-ontos

4.2 Obtención de ontologías para el catálogo

El catálogo inicial de Ágora se ha construido a partir de un conjunto seleccionado de ontologías procedentes de diversas fuentes relevantes del ámbito de la Web Semántica y la ingeniería ontológica. Este proceso ha sido crucial para disponer de un conjunto representativo de ontologías que facilitaran el proceso de validación de las funcionalidades de búsqueda, visualización y reutilización del sistema. Para esta etapa fue crucial recopilar ontologías conceptualizadas mediante la notación de Chowlk, o que fueran susceptibles de ser representadas con esa notación, para que fueran compatibles con la estructura del catálogo y su posterior procesamiento automático. Las ontologías que conforman el catálogo inicial se han obtenido mediante un proceso manual. Además, se ha definido un mecanismo para que los usuarios puedan aportar sus propias ontologías al repositorio.

Las ontologías que forma el catálogo inicial de Ágora provienen de distintos orígenes. Las ontologías SAREF for Building Ontology y BIMERR Building Ontology se obtuvieron directamente de la sección de ejemplos que ofrece Chowlk en su web oficial.[20] De ahí se pudieron obtener directamente los archivos necesarios para incorporarlos al repositorio agora-ontos, tanto la imagen del diagrama de conceptualización en formato .jpg, el archivo .XML con la conceptualización de la ontología y el archivo .TTL generado por Chowlk a partir de la conceptualización. Los logos para dichas ontologías se extrajeron manualmente desde los sitios oficiales[21] que albergan las ontologías.

Con el fin de tener algún ejemplo de ontología que contase con su propia librería de patrones, llevó a cabo un proceso manual de extracción de patrones ontológicos a partir del archivo XML de la conceptualización generado por Chowlk. Este procedimiento consistió en abrir el archivo en draw.io y extraer manualmente cada patrón identificado, asignándoles un nombre y almacenándolos como una librería de patrones independientes. Esta librería y exportó para agregarla a los archivos de la ontología proporcionados al repositorio agora-ontos y para, posteriormente, enviarla como librería en el *iframe* principal de conceptualización de la ontología completa. Además, la librería de patrones fue procesada para separar individualmente los patrones y enviar cada uno de ellos a los *iframes* específicos de la aplicación dedicados a visualización de patrones.

Además, se incluyeron otras ontologías disponibles a través de la sección "*About*"[22] de Chowlk, que proporciona enlaces a proyectos externos desarrollados con esta metodología. Entre ellos, se obtuvieron las ontologías del proyecto Cogito[23]: COGITO Process Ontology[24], COGITO Facility Ontology[25], COGITO Resources Ontology[26], COGITO Quality ontology[27], COGITO Safety

[20] https://chowlk.linkeddata.es/examples
[21] https://saref.etsi.org/saref4bldg/v2.1.1/
https://bimerr.iot.linkeddata.es/def/building/
[22] https://chowlk.linkeddata.es/about
[23] https://cogito.iot.linkeddata.es/
[24] https://cogito.iot.linkeddata.es/def/process/
[25] https://cogito.iot.linkeddata.es/def/facility/
[26] https://cogito.iot.linkeddata.es/def/resource/
[27] https://cogito.iot.linkeddata.es/def/quality/

Ontology[28], COGITO IoT ontology [29]y COGITO WoTDT.[30] La descarga de los archivos correspondientes a la imagen del diagrama de conceptualización en formato .png o jpg, el archivo .TTL de la ontología y el logo de la organización, necesarios para incluirlas en agora-ontos, se realizó desde su sitio oficial. Aunque para este caso particular, fue necesario realizar una extracción manual de cada uno de los diagramas de conceptualización de estas ontologías, ya que la plataforma no ofrecía una opción de descarga directa. Este caso sirvió como ejemplo de casos en los que las ontologías no son independientes si no que forman parte de una misma organización.

Por último, también se integraron las ontologías del proyecto Delta[31]: DELTA Ontology [32] y OpenADR Ontology[33], obtenidas también desde las referencias de Chowlk. Esta ontología se incorporó en el repositorio siguiendo el mismo proceso de extracción, revisión y adaptación que las anteriores.

4.3 Mecanismo de Contribución para Usuarios

Ágora se ha concebido como un catálogo abierto y colaborativo, que busca que ampliar sus ontologías mediante las contribuciones de los usuarios con sus propias ontologías desarrolladas con la notación Cholwk. Para facilitar este proceso de contribución al repositorio de GitHub agora-ontos, existen distintas formas de participación adaptadas a diferentes niveles de experiencia técnica.

Aquellos usuarios que estén familiarizados con Git y GitHub pueden realizar un fork del repositorio para crear una copia propia que puedan clonar localmente y crear una nueva carpeta para su ontología, siguiendo la estructura y los requisitos de archivos definidos en agora-ontos, incorporar su ontología incluyendo, al menos, el archivo .XML de la conceptualización, el archivo OWL en formato .TTL que pueden generar con Chowlk a partir de la conceptualización, una imagen de vista previa; asegurándose de que los nombres y formatos sean los esperados. Para, posteriormente, confirma (commit) y subir (push) los cambios a su copia del repositorio y finalmente, enviar una solicitud de incorporación de cambios (*pull request*) a la rama principal del repositorio (main), que será posteriormente revisada. Este método, permite mantener un control de calidad y trazabilidad sobre los recursos añadidos.

Para los usuarios sin experiencia en Git, alternativamente, pueden realizar la subida directamente a través de la interfaz web de GitHub. Navegando la carpeta deseada dentro del repositorio agora-ontos y mediante la opción "*Add file*" y, *"Upload files"*, arrastrar y soltar los archivos de su ontología (incluyendo los archivos requeridos mencionados anteriormente). Una vez subidos, se requiere un mensaje descriptivo de *commit* para registrar los cambios. También como alternativa, es posible crear un *issue* en el repositorio, adjuntando un enlace de descarga a los archivos o solicitando ayuda para su inclusión. Este método facilita una participación más accesible sin comprometer la integridad del repositorio.

---

[28] http://cogito.iot.linkeddata.es/def/safety
[29] https://cogito.iot.linkeddata.es/def/iot/
[30] https://w3id.org/def/dtw#0.5.0
[31] http://delta.iot.linkeddata.es/
[32] http://delta.linkeddata.es/def/core/index-en.html
[33] https://albaizq.github.io/OpenADRontology/

Para facilitar a los usuarios la colaboración en Ágora con sus propias ontologías, se añadido al repositorio agora-ontos, un documento detallado "*contributing.md*" que explica cómo pueden contribuir. Este documento se encuentra adjuntado en el anexo I.

4.4 Procesamiento del repositorio de ontologías

Con el fin de integrar las ontologías contenidas en el repositorio agora-ontos, alojado en GitHub, al catálogo de la aplicación web, sistema Ágora integra una funcionalidad automatizada para procesar y mantener actualizado el catálogo de ontologías. Este procesamiento es fundamental para asegurar que el catálogo siempre refleje los cambios más recientes en el repositorio y que las ontologías estén siempre disponibles, correctamente indexadas y listas para su exploración, descarga o reutilización dentro de la plataforma.

Para una mejor comprensión del proceso automatizado de integración y actualización de las ontologías en el catálogo de la aplicación web, a continuación se presenta un diagrama de flujo que ilustra las distintas etapas involucradas:

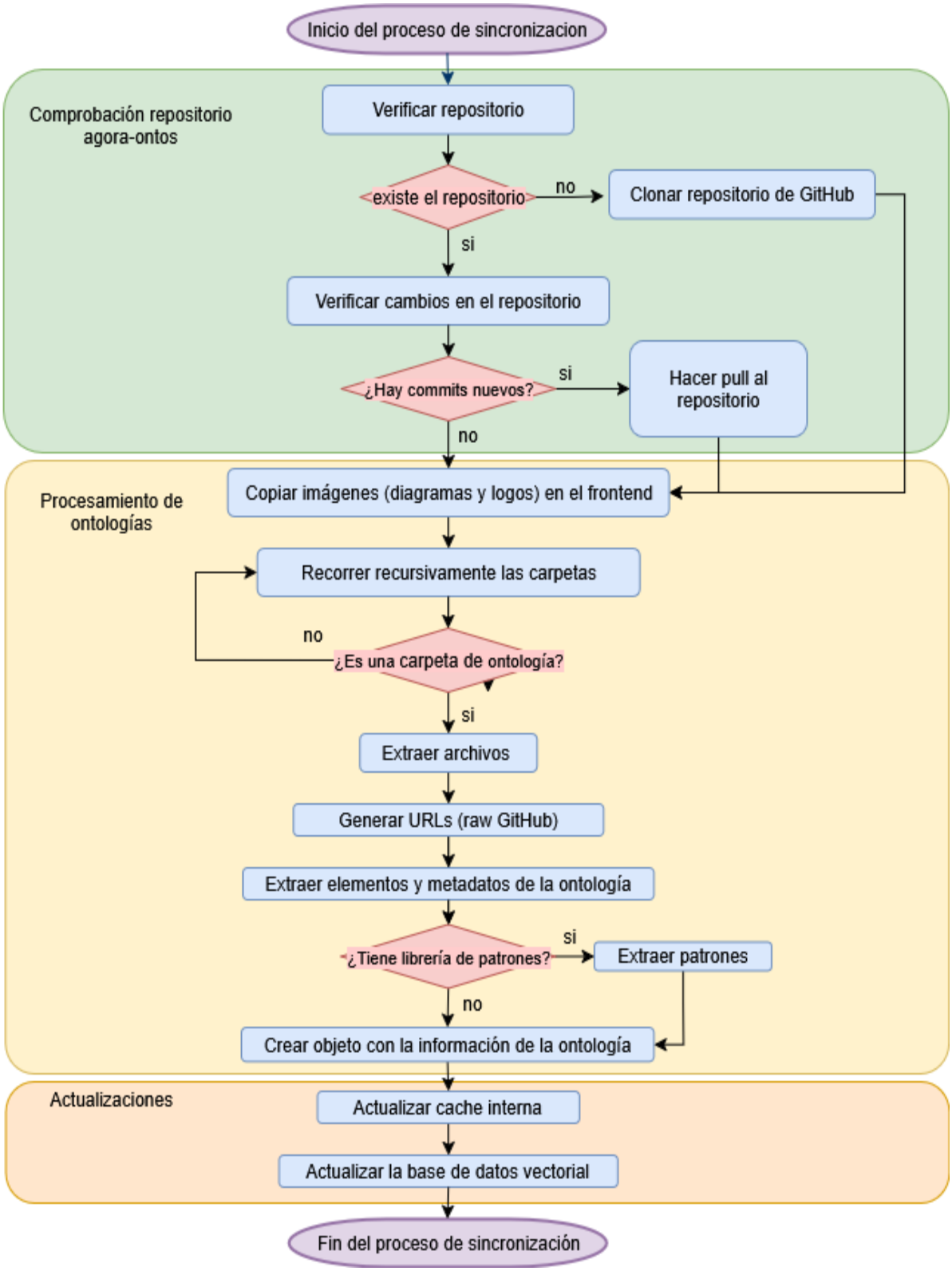

Ilustración 2. Diagrama de flujo del procesamiento automático del repositorio agora-ontos

Esta tarea se realiza siguiendo los siguientes pasos:

1. **Detección y sincronización de cambios en el repositorio remoto**.

   Para mantener el catálogo sincronizado con las últimas contribuciones y actualizaciones, se ejecuta una tarea asíncrona en segundo plano en el *backend* de FastAPI. Esta tarea, definida implementada como un servicio asíncrono, se ejecuta periódicamente (actualmente, cada hora según la configuración, o cada vez que la caché de ontologías caduca o se fuerza una actualización). a comprobación de cambios se realiza utilizando la librería GitPython[34], que permite la interacción con repositorios Git directamente desde Python. El servicio compara el hash del último *commit* de la rama *main* del repositorio local con el del repositorio remoto. En caso de detectar nuevos cambios (commits), se ejecuta automáticamente un proceso de actualización que clona, mediante git.Repo.clone_from() (en caso de no estar clonado ya) o sincroniza el contenido más reciente, mediante un *pull* de Git (repo.remotes.origin.pull('main')). Este repositorio se clona en una carpeta local del proyecto (/backend/static/agora-ontos). Este proceso garantiza que tanto las nuevas ontologías se incorporen como las existentes se actualicen de forma inmediata en el catálogo de la aplicación sin necesidad de intervención manual.

2. **Sincronización de archivos estáticos**.

   Después de cualquier actualización o clonación del repositorio, los archivos de imagen (diagramas de conceptualización y logos de ontologías) presentes en el repositorio Git se copian automáticamente a un directorio local (/web/static/agora-ontos) dentro de la estructura del *frontend* de Flask. Este paso es fundamental para que las imágenes puedan ser servidas y renderizadas correctamente por la aplicación web y se visualicen en las tarjetas del catálogo, las vistas detalladas de ontologías y en los *iframes* interactivos ya que Flask sólo puede servir las imágenes que se encuentren en la carpeta */static.*

3. **Procesamiento estructurado de las ontologías**.

   Una vez clonado o sincronizado el repositorio, se activa un servicio que recorre de forma recursiva la estructura de carpetas del repositorio, interpretando cada directorio como una posible ontología individual o como una agrupación de varias ontologías pertenecientes a una misma organización. Para cada carpeta identificada como una ontología (es decir, que contiene al menos un archivo .XML de conceptualización), se realiza lo siguiente:

   - **Extracción de archivos.** Se identifican y extraen el archivo correspondiente a la ontología formalizada en OWL (generalmente .TTL), el archivo XML de la conceptualización de Chowlk, la imagen del diagrama de conceptualización (.jpg o .png) y, si en caso de existir, el archivo XML de la librería de patrones y un logo específico de la ontología

[34] https://gitpython.readthedocs.io/en/stable/

(o se hereda uno de un directorio padre en caso de ser ontologías pertenecientes a un proyecto u organización).

- **Generación de URLs.** Se construyen URLs de acceso directo a los archivos *raw* de GitHub para los ficheros OWL, XML y de patrones. Estas URLs son necesarias para permitir la descarga directa de los archivos por parte del usuario y para incrustar los diagramas en los *iframes* interactivos de draw.io.

- **Análisis y Extracción de elementos y metadatos de la ontología.** El archivo OWL de cada ontología es parseado utilizando la librería RDFLib[35]. Se extrae automáticamente metadatos como título, descripción, autor, licencia, versión, entre otros, que cumplen con estándares como Dublin Core[36] (DC/DCTERMS) y OWL, así como información sobre las clases, propiedades de objeto y propiedades de tipo de dato definidas en la ontología. Esta información se almacena en una instancia de un objeto (DTO) interno (*OntologyFromRepoDto*) que representa cada ontología en la memoria del sistema.

- **Extracción de patrones.** En caso de que la ontología cuente con una librería de patrones, el archivo .XML correspondiente se procesa para obtener cada patrón individualmente y darle el formato necesario para poder incrustarlo individualmente en su propio *iframe* debido a que drawio es capaz de reconocer automáticamente un patrón extraído de la librería de patrones como un diagrama.

- **Actualización de la Caché en Memoria.** Una vez procesadas, las ontologías se almacenan en una caché interna gestionada por el backend. La caché tiene un tiempo de vida determinado, tras el cual se actualiza automáticamente, o bien se actualiza inmediatamente cuando se detecta un cambio en el repositorio remoto. Esta caché es la fuente principal de datos para la visualización del catálogo en la interfaz web de Ágora y para su descarga. Además, se utiliza para mostrar información detallada en los *iframes* de la aplicación, visualizar los diagramas en draw.io o cargar la librería de patrones, cuando exista.

4. **Actualización de la Base de Datos Vectorial (Qdrant)**

Después de cada actualización de la caché, se inicia el proceso de reindexación de ontologías en la colección correspondiente en la base de datos vectorial Qdrant, lo que asegura la coherencia entre los datos del repositorio y la búsqueda semántica.

4.5 Construcción de la base de datos vectorial para la búsqueda semántica.

Con el objetivo de ofrecer una búsqueda semántica sobre el catálogo de ontologías de Ágora, se ha integrado un sistema de indexación basado en

[35] https://rdflib.readthedocs.io/en/stable/
[36] https://www.dublincore.org/specifications/dublin-core/dcmi-terms/

embeddings y almacenamiento vectorial. Este sistema permite transformar el contenido textual de las ontologías y sus elementos (clases, propiedades, patrones y metadatos) en vectores numéricos de alta dimensión y organizarlos en una base de datos vectorial permitiendo consultas que van más allá de las coincidencias literales de términos. Para ello, se han seguido los siguientes pasos:

1. **Configuración e inicialización de la colección en Qdrant.**

   Al iniciar el servicio de búsqueda, el sistema de Ágora verifica la existencia de la colección de ontologías en Qdrant[37]. En caso de no existir o de requerir una actualización, la colección se recrea. Para la configuración, se definen parámetros esenciales como el tamaño de los vectores (384 dimensiones) y la métrica de distancia utilizada para la búsqueda de similitud, que en este caso, es el coseno (*models.Distance.COSINE*). Esta métrica es fundamental, ya que mide el ángulo entre dos vectores, permitiendo recuperar los elementos más cercanos conceptualmente a una consulta de entrada, siendo 1 para vectores idénticos, es decir aquellos con alta similitud semántica, y -1 para opuestos. Esto asegura que la base de datos esté lista para almacenar y comparar los *embeddings*.

2. **Generación de embeddings.**

   Para cada ontología de la caché (como se describe en la Sección 3.3), se extrae su contenido textual relevante para transformarlo en *embeddings*. Este proceso es llevado a cabo por un servicio de *embeddings* que utiliza un modelo de *sentence-transformers*, concretamente el modelo all-MiniLM-L6-v2. Este modelo, basado en BERT (*Bidirectional Encoder Representations from Transformers*), ha sido elegido por equilibrar la precisión semántica y eficiencia computacional, siendo idóneo para tareas de búsqueda semántica en dominios heterogéneos como el de las ontologías. De cada ontología se extraen los siguientes elementos para generar los *embeddings*:

   - **Ontologías.** Se genera un *embedding* para la ontología completa de la cache, concatenando todos sus elementos: título, descripción, resumen y los nombres de sus clases y propiedades (de objeto y de tipo de dato).

   - **Metadatos.** Se generan *embeddings* tanto para bloques de metadatos (como el conjunto completo de metadatos de Dublin Core), como para campos de metadatos individuales de la ontología (título, descripción, licencia, creador, etc.)

   - **Elementos ontológicos.** Para cada clase, propiedad de objeto (*object property*) y propiedad de tipo de dato (*datatype property*) se genera un *embedding* utilizando su nombre o URI.

   - **Patrones:** En caso de que la ontología incluya patrones, se genera en un *embedding* para cada patrón individual (con su título y contenido XML).

[37] https://qdrant.tech/documentation/

A partir de estos elementos, se construyen representaciones textuales normalizadas, realizando una limpieza básica para eliminar espacios en blanco antes de la generación del vector. Esta granularidad del texto utilizado para generar *embeddings* es esencial para la precisión en la búsqueda semántica.

3. **Almacenamiento en base de datos vectorial.**

Los vectores generados se almacenan en la colección de Qdrant, una base de datos optimizada para trabajar con vectores en espacios de alta dimensión, permitiendo realizar búsquedas por similitud de manera eficiente. Cada *embedding* se almacena como un "punto" en la base de datos vectorial.

Cada punto indexado en la colección contiene:

- El **vector numérico.**

- Una **carga útil** (*payload*) con metadatos adicionales como el nombre de la ontología, el tipo de elemento (clase, propiedad, metadato, patrón, etc.), su URI o descripción.

- Un **id único** consistente generado a partir de su identificador semántico, que consiste en una cadena de texto única que combina la id de la ontología y el tipo o id del elemento, usando un hash SHA256 para asegurar la unicidad y consistencia entre actualizaciones, evitando que se inserten los mismos elementos varias veces en la colección.

Los puntos (vectores con sus ids y *payloads*) se recolectan y se insertan por lotes en Qdrant mediante una operación de *upsert*, lo que optimiza el rendimiento de la indexación. Este diseño permite recuperar, tanto los resultados, como su contexto ontológico; pudiendo así manejar el resultado de la búsqueda sin tener que consultar la caché nuevamente.

4. **Indexación y consistencia.**

Tras cada actualización del repositorio de ontologías, se reconstruye completamente la colección de Qdrant para asegurar la coherencia entre el estado actual del catálogo y la base de datos vectorial. Esta tarea se realiza de automáticamente como parte del proceso de actualización del *backend*, asegurando que la funcionalidad de búsqueda semántica esté siempre alineada con el contenido real del repositorio.

4.6 Mecanismo de búsqueda y presentación de resultados en el catálogo

Ágora, como se puede observar en los prototipos, dispone de una barra de búsqueda donde el usuario puede introducir consultas para explorar el catálogo de ontologías. Con el objetivo de que está búsqueda sea lo más completa posible y para ayudar a los usuarios con menos experiencia en el desarrollo ontológico en su labor, Ágora incorpora una funcionalidad de búsqueda semántica sobre el catálogo de ontologías, permitiendo a los usuarios realizar consultas en

lenguaje natural y recuperar resultados relevantes, filtrando no solo a nivel de coincidencia textual, sino también en términos de similitud semántica. Esta funcionalidad se implementa mediante un flujo de interacción entre el cliente web (Flask), el *backend* (FastAPI) y la base de datos vectorial Qdrant, que gestiona los vectores generados a partir de las ontologías y sus componentes.

La búsqueda semántica no se limita solo a encontrar ontologías dentro del catálogo, sino que permite localizar elementos internos como clases, propiedades, patrones o incluso metadatos específicos, permitiendo una exploración más precisa y contextualizada.

Este proceso puede dividirse en dos tareas, la gestión de las consultas y la gestión de los resultados.

4.6.1 Gestión de consultas en el catálogo

Este proceso, a su vez, puede dividirse en diferentes fases:

1. **Captura y envío de la consulta**

   Cuando un usuario introduce una consulta en la barra de búsqueda de la interfaz web de Ágora, se valida que la consulta no esté vacía y es enviada mediante una petición POST al *endpoint /api/search* del servidor Flask. Además, se actualiza la interfaz, ocultando el catálogo principal y mostrando un mensaje de espera en el contenedor de resultados. El controlador correspondiente en Flask, que actúa como un *proxy,* recibe esta solicitud y la reenvía, utilizando la librería *requests*[38], como una petición POST al *endpoint* /api/catalogue/search del *backend* en FastAPI, enviando como cuerpo de la solicitud un JSON con la cadena de búsqueda proporcionada por el usuario.

2. **Vectorización y búsqueda semántica**

   En el *backend*, el controlador correspondiente redirige la consulta al servicio que gestiona el proceso de búsqueda. La cadena de texto de la consulta del usuario se transforma en un vector semántico, utilizando el modelo all-MiniLM-L6-v2 de Sentence Transformers. Este vector de consulta se utiliza para realizar la búsqueda por similitud en el espacio vectorial en la colección de ontologías, creada anteriormente, de Qdrant (self.qdrant.search()). La búsqueda puede incluir filtros opcionales (ej., tags). Se obtienen los top_k elementos más similares (por configuración, 10), que pueden incluir ontologías completas, clases, propiedades o patrones individuales. Recibiendo también el *payload* de cada punto para obtener los metadatos asociados.

4.6.2 Gestión de resultados de las consultas

Una vez que el motor de búsqueda vectorial devuelve los resultados, estos deben ser procesados y estructurados para presentarlos en la interfaz de usuario.

[38] https://pypi.org/project/requests/

1. Procesamiento de Resultados en el Backend

El servicio encargado de gestionar el proceso de búsqueda recibe una lista de *ScoredPoint* de Qdrant, que incluyen el vector de resultado, el score de similitud y el *payload* con los metadatos. A continuación, se extraen los resultados, obteniendo la información de cada *payload* como nombre e id de la ontología a la que pertenecen, el tipo de elemento (ontología, clase, propiedad, metadato o patrón), la *label* con el nombre del elemento, la URI, el id del patrón, entre otros. Estos datos se transforman en objetos DTOs (Data Transfer Objects) estructurados para facilitar su transmisión al *frontend.*

2. Presentación de Resultados

Los resultados se ordenan según su grado de similitud (*score*), identificando el id de la ontología con la mayor puntuación para destacarla. Si esta no es una ontología completa sino un elemento (clase, propiedad), se recupera la información completa de la ontología a la que pertenece ese elemento. Los resultados se agrupan según el tipo de elemento para la visualización en el panel lateral de resultados de la interfaz. Flask renderiza la plantilla HTML parcial correspondiente a la vista de los resultados, pasándole la consulta original, los resultados agrupados y la ontología destacada. Esta plantilla es la que define la estructura del panel izquierdo con los resultados por tipo y el panel derecho con la visualización contextual. El HTML renderizado se inyecta en el contenedor de resultados de la página, mostrándolos al usuario y permitiéndole interactuar con ellos.

5 Diseño

El diseño de una herramienta web es una fase crítica en el proceso de desarrollo, especialmente cuando se trata de una aplicación que busca facilitar la reutilización y visualización de recursos complejos como las ontologías. En este contexto, el proceso de prototipado cobra una importancia fundamental, ya que los prototipos son una representación preliminar de la aplicación final, permitiendo visualizar y probar la interfaz antes de realizar el desarrollo completo, ya que los cambios posteriores una vez la herramienta ha sido desarrollada son más costosos en cuanto a tiempo y recursos que hacer modificaciones en los prototipos. Además, facilita la identificación temprana de posibles mejoras o errores de diseño, y contribuye directamente a mejorar la usabilidad y la experiencia de usuario (UX) de la herramienta, permitiendo obtener una herramienta final que sea usable, intuitiva, eficiente y responda a las necesidades de los usuarios.

Para el diseño de Ágora, se ha buscado una interfaz que, aunque busca la innovación en ciertos aspectos, sigue patrones y convenciones de diseño presentes en otros catálogos de ontologías o plataformas existentes, lo que permite aprovechar la familiaridad del usuario con interfaces similares, reduciendo la curva de aprendizaje y facilitando una adopción más rápida de la herramienta.

5.1 Árbol web

Para poder estructurar y organizar el contenido del sistema se ha decidido realizar un árbol web. Un árbol web consiste en una representación esquemática la estructura del sistema es decir, establece la forma en la que se muestra la información a los usuarios, estableciendo la relación entre las diferentes páginas y secciones que lo conforman.

Esta herramienta nos permite por un lado, diseñar un sistema con una navegación lógica e intuitiva para que el sistema sea fácil de utilizar para los usuarios y por otro lado, nos permite realizar una mejor planificación del desarrollo del sistema antes de iniciar la implementación del código.

Para la realización de este árbol, se ha empezado por recopilar el contenido que la aplicación debe incluir, como sus funcionalidades principales y los datos a mostrar (ontologías, patrones, información del proyecto, etc.) después, se establecieron las páginas y secciones y sus jerarquía y relaciones entre ellas.

Aunque este árbol ha pasado por diferentes versiones, para la estructura final de la aplicación quedaría de la siguiente manera:

Página	URL	Descripción breve
Catálogo principal	/ (también accesible por /index y /catalogue)	Página inicial del sistema con la barra de búsqueda y el catálogo de ontologías.
Contribute	/contribute	Donde los usuarios obtienen información sobre cómo pueden añadir sus ontologías al catálogo.

How to	/howto	Información sobre el funcionamiento y el modo de uso de la aplicación
Github	https://github.com/oeg-upm/agora-ontos	Repositorio GitHub, Ágora-Ontos, donde se encuentran todas las ontologías que forman el catálogo.
Chowlk	https://chowlk.linkeddata.es/	Enlace a la herramienta Chowlk, debido a su relación y complementariedad con Ágora.

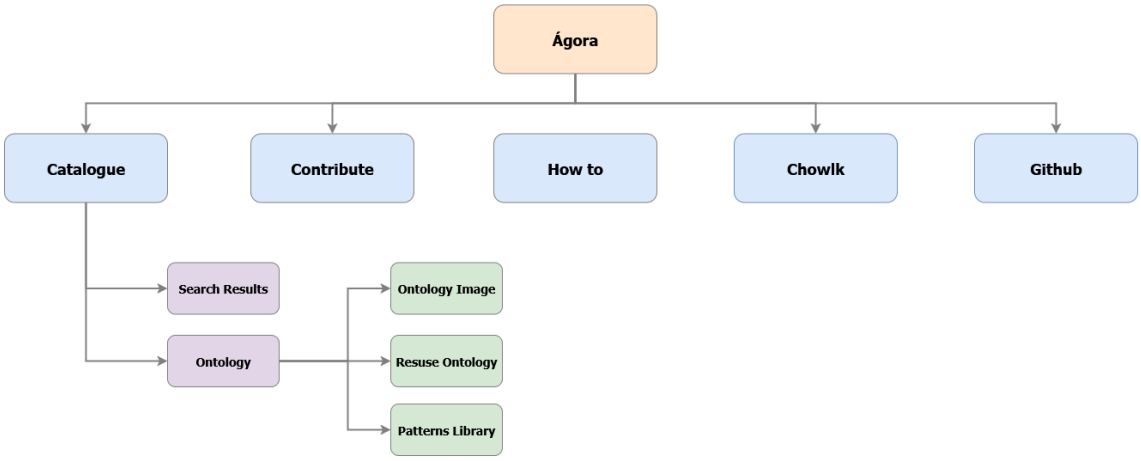

Ilustración 3. Árbol web

Este árbol web fue es la base para la creación de los prototipos de baja y alta fidelidad, con el fin de asegurar que el diseño de la interfaz tenga una base de información estructurada y una navegación bien definida. Por lo que, una vez establecida la estructura básica de la web, con esta información se puede proceder al prototipado.

5.2 Prototipado

Como se ha expuesto anteriormente, el proceso de prototipado es fundamental para el diseño de este tipo de herramientas. El proceso de prototipado se inició con la creación de prototipos de baja fidelidad.

5.2.1 Prototipo de baja fidelidad

Los prototipos de baja fidelidad [30] son representaciones aproximadas de conceptos que nos ayudan a validar esos conceptos al principio del proceso de diseño. Se le llama de baja fidelidad debido a que su aspecto no necesariamente debe tener un elevado grado de exactitud respecto del aspecto de la interfaz final del sistema aunque sí debe operar de la misma manera.

Este tipo de prototipos son muy comunes en los diseños centrados en el usuario ya que permiten obtener información sobre la interacción de los usuarios con el posible nuevo diseño. Son representaciones muy esquemáticas y sencillas de la interfaz de usuario Este método requiere menos tiempo y recursos especializados, es decir, es una forma barata y rápida de evaluar el diseño. Esto

permite realizar un mayor número de prototipos y evaluaciones a los usuarios, permitiendo realizar cambios en los prototipos con mayor facilidad e incluso realizar varios prototipos.

El estilo de la interacción se ha realizado con el fin de hacer que el uso del sistema sea más sencillo e intuitivo, para así mejorar su usabilidad de acuerdo con las heurísticas de Nielsen [31].

La creación del prototipo de baja fidelidad se ha realizado con la herramienta Figma.[39] Se ha elegido esta herramienta por ser una herramienta online, que no requiere ninguna instalación y por contar con una interfaz intuitiva con una alta capacidad de maquetación que permite crear *wireframes* y esquemas de pantalla con facilidad. Además, Figma ofrece la flexibilidad de pasar a prototipos de alta fidelidad fácilmente, lo que la convierte en una herramienta muy útil para todo el proceso de diseño.

Inicialmente, el catálogo de ontologías y sus conceptualizaciones, se concibió como una posible incorporación a la herramienta web ya existente, Chowlk, ya que las conceptualizaciones se llevan a cabo siguiendo su notación. La idea era integrar la funcionalidad del catálogo dentro de su ecosistema actual. Sin embargo, durante el desarrollo del proyecto, se llegó a la conclusión de que debido a la magnitud y las necesidades específicas del catálogo de ontologías era necesario la creación de una herramienta nueva e independiente. Como se ha mencionado anteriormente, esta nueva herramienta ha sido llamada Ágora. Este cambio estratégico permite una mayor flexibilidad y un control total sobre la arquitectura y las funcionalidades del catálogo.

Durante el proceso de diseño se realizaron dos prototipos de baja fidelidad debido a que se encontraron limitaciones en el prototipo inicial, se optó por la creación de un segundo prototipo. Estos prototipos de diferencian principalmente por la forma en la que se visualiza el catálogo de ontologías. A continuación se expondrán ambos prototipos para explicar sus diferencias y la mejora de la segunda versión frente al primero.

5.2.1.1 Inicio

La pantalla de inicio o pantalla principal del catálogo que se muestra a continuación corresponde a la primera interfaz que visualizaría el usuario al entrar en el catálogo. Esta pantalla es una de las que más ha cambiado en la segunda versión del prototipo respecto de la primera.

5.2.1.1.1 Versión 1

Como se muestra en la Ilustración 4 la versión inicial contemplaba la presentación de las ontologías en un formato de listado en una tabla. Esta tabla mostraba las ontologías y los patrones de manera estructurada, con columnas para metadatos como el título, la URI, el creador, el idioma y el tipo (ontología o patrón). Este diseño priorizaba la información sobre la facilidad de comprensión a primera vista. Aunque era un diseño funcional, se consideró que podría no ser la forma más atractiva visualmente ni la más eficaz para destacar las características individuales de cada ontología.

[39] https://www.figma.com

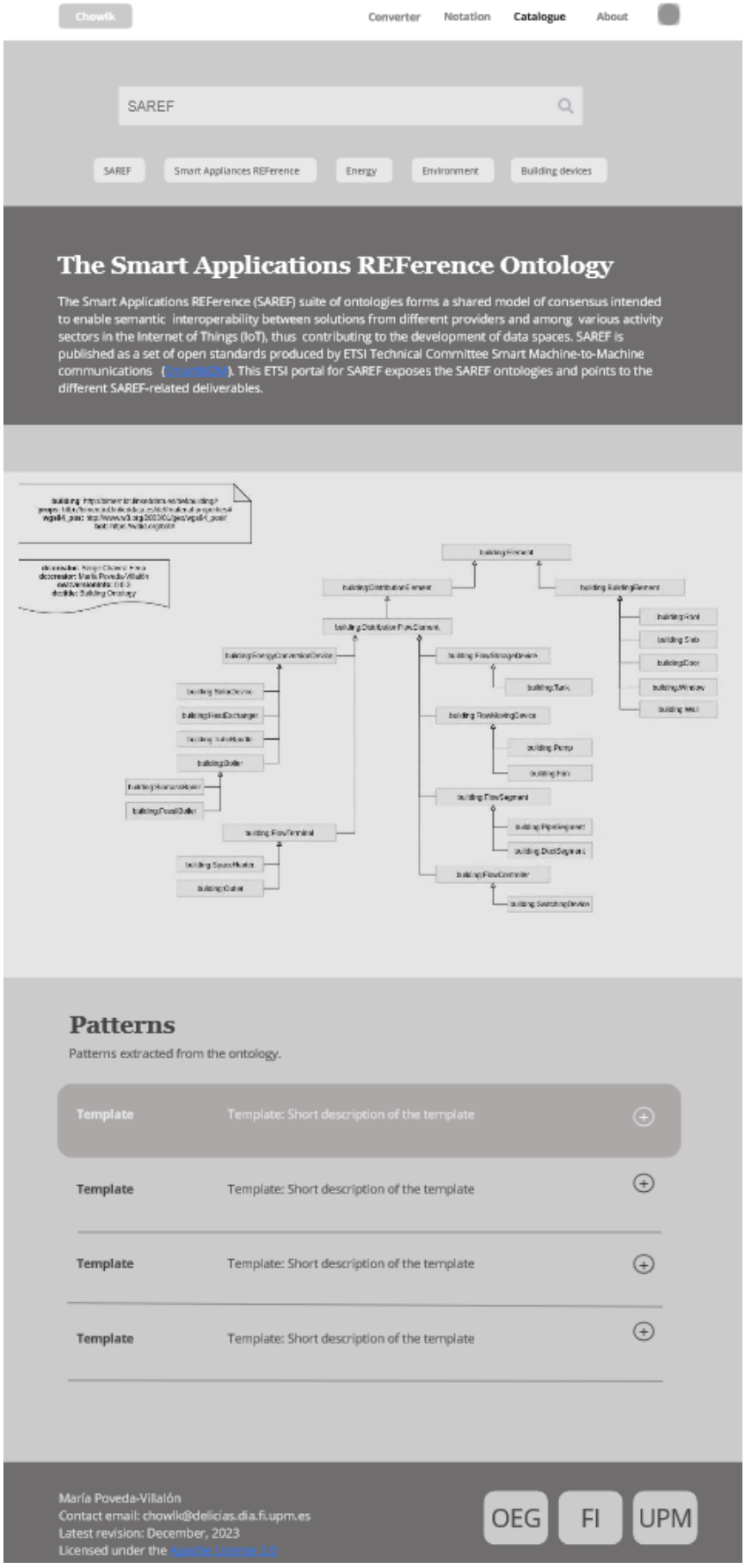

Ilustración 4. Pantalla de inicio de la Versión 1 del prototipo de baja fidelidad

5.2.1.1.2 Versión 2

Debido a las limitaciones de este diseño, en la versión posterior se optó por una disposición del catálogo de las ontologías en formato de "*cards*" o tarjetas, como se puede apreciar en siguiente Ilustración 5 Este enfoque ofrece una visualización más rica y modular, permitiendo incluir una representación más visual, con el logo, y la descripción de cada ontología, en caso de que ésta esté incluida en sus metadatos. La disposición en tarjetas mejora significativamente la experiencia de usuario al ser más intuitiva y visualmente más atractiva, facilitando la exploración y el descubrimiento de ontologías de interés.

Además de esta mejora en la presentación de las ontologías, el diseño ambas versiones del prototipo incluían elementos clave para la funcionalidad y la coherencia de la interfaz. Se incluyó una barra de búsqueda en la parte superior, donde los usuarios pueden introducir consultas para realizar búsquedas específicas dentro del catálogo. Este elemento es fundamental en esta herramienta. Además, para poder asegurar una buena experiencia de usuario en toda la herramienta, se estableció un diseño común para el *layout* general, manteniendo el mismo *header* y *footer* en todas las vistas de la aplicación, garantizando, no sólo la consistencia de la interfaz, sino que también facilitando la familiaridad y el uso intuitivo.

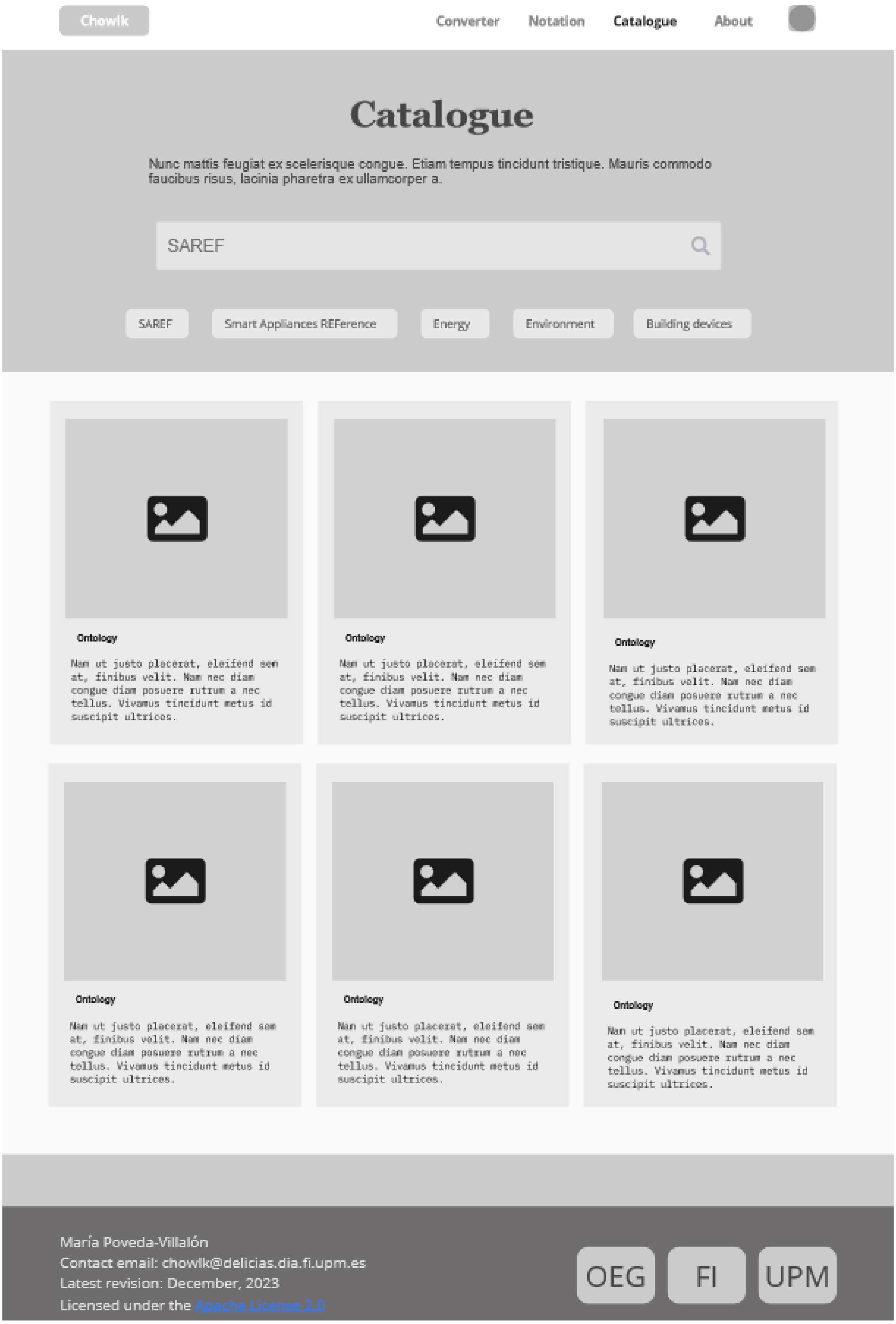

Ilustración 5. Pantalla de inicio de la Versión 2 del prototipo de baja fidelidad

5.2.1.2 Resultados de la búsqueda

La funcionalidad de búsqueda es un pilar fundamental en cualquier catálogo de gran volumen, y especialmente, como se ha explicado anteriormente, para el caso de esta herramienta. Por extensión, también es crucial para la aplicación la forma en que se presentan los resultados.

5.2.1.2.1 Versión 1

En la primera versión del prototipo, la visualización de resultados de búsqueda (como se puede ver en la Ilustración 6*)* consistió en una presentación similar a la del catálogo completo, en formato de tabla pero con ontologías filtradas según la consulta del usuario.

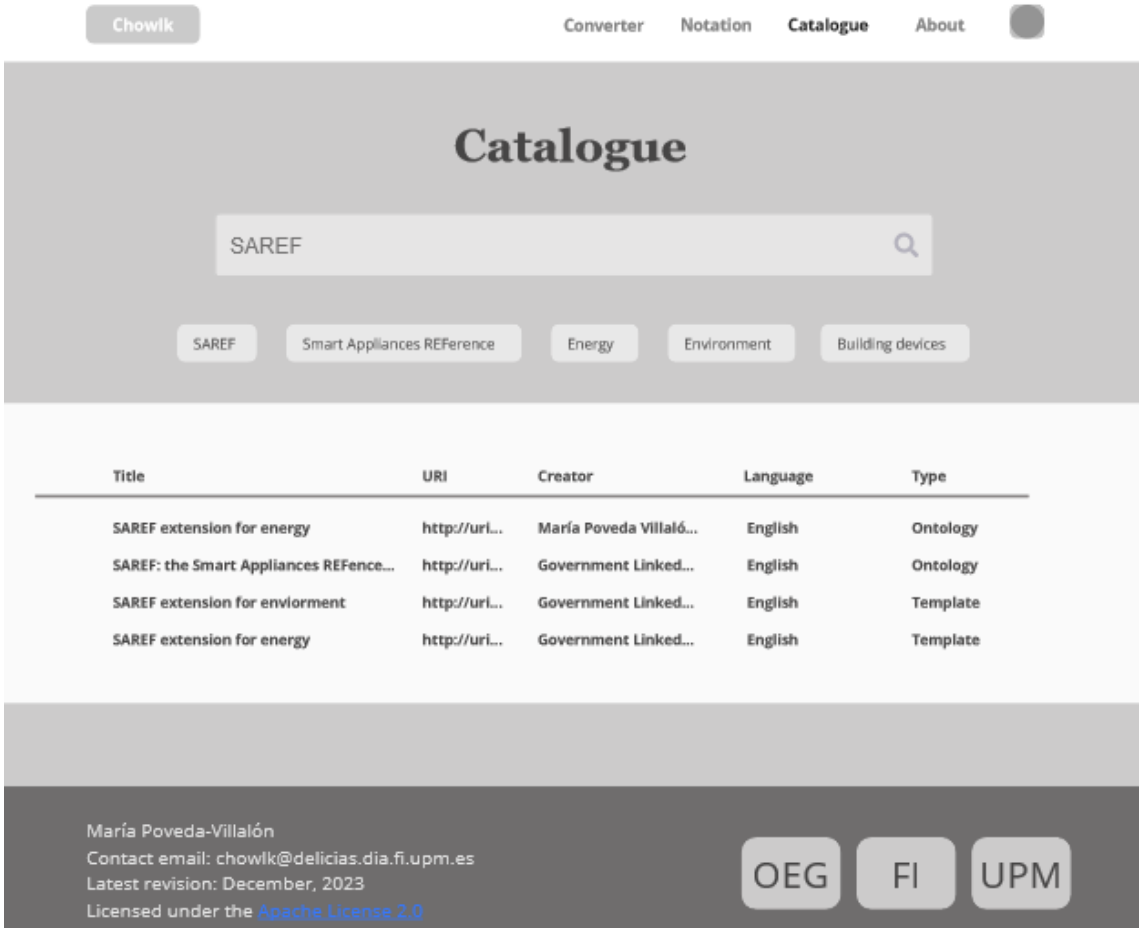

Ilustración 6. Pantalla de resultados de la búsqueda de la Versión 1 del prototipo de baja fidelidad

Además, en *Ilustración 7* se puede ver que para esta vista, se planteó una previsualización de la ontología preseleccionada de entre los resultados de la búsqueda. Esta funcionalidad estaba pensada para permitir al usuario obtener una visión más detallada de un elemento sin tener que navegar a una página aparte. Sin embargo, esta idea fue finalmente descartada en la posterior versión del prototipo por considerar que añadía complejidad visual que podía sobrecargar la interfaz.

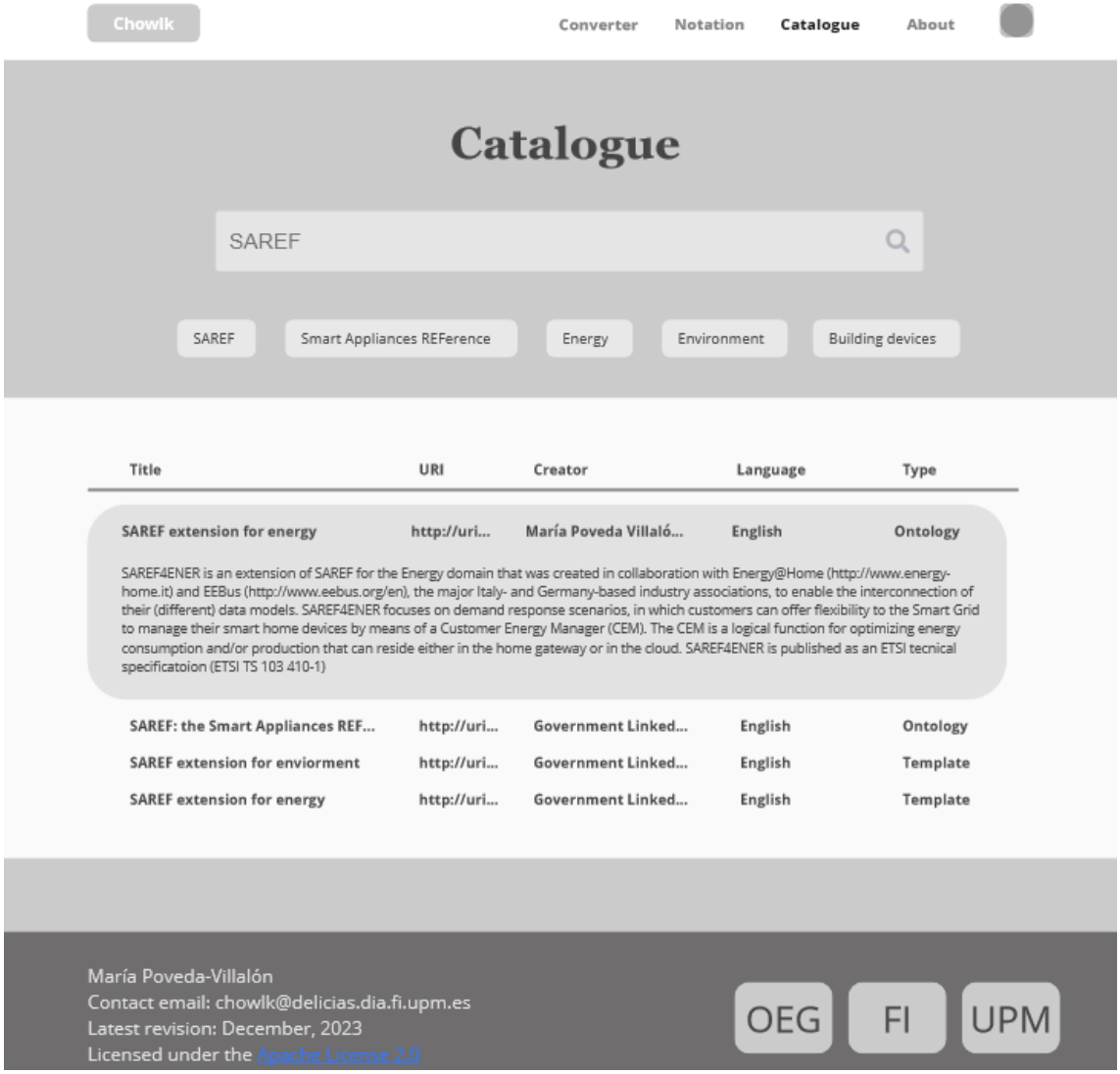

Ilustración 7. Pantalla de previsualización de la ontología resultado de la búsqueda de la Versión 1 del prototipo de baja fidelidad

5.2.1.2.2 Versión 2

Para la segunda versión del prototipo, se decidió adoptar una presentación de los resultados de búsqueda en el ya establecido formato de "*cards*" o tarjetas, tal como se muestra en la Ilustración 8. Este cambio de diseño mantiene la coherencia con la pantalla principal del catálogo lo que hace la experiencia de usuario unificada y predecible.

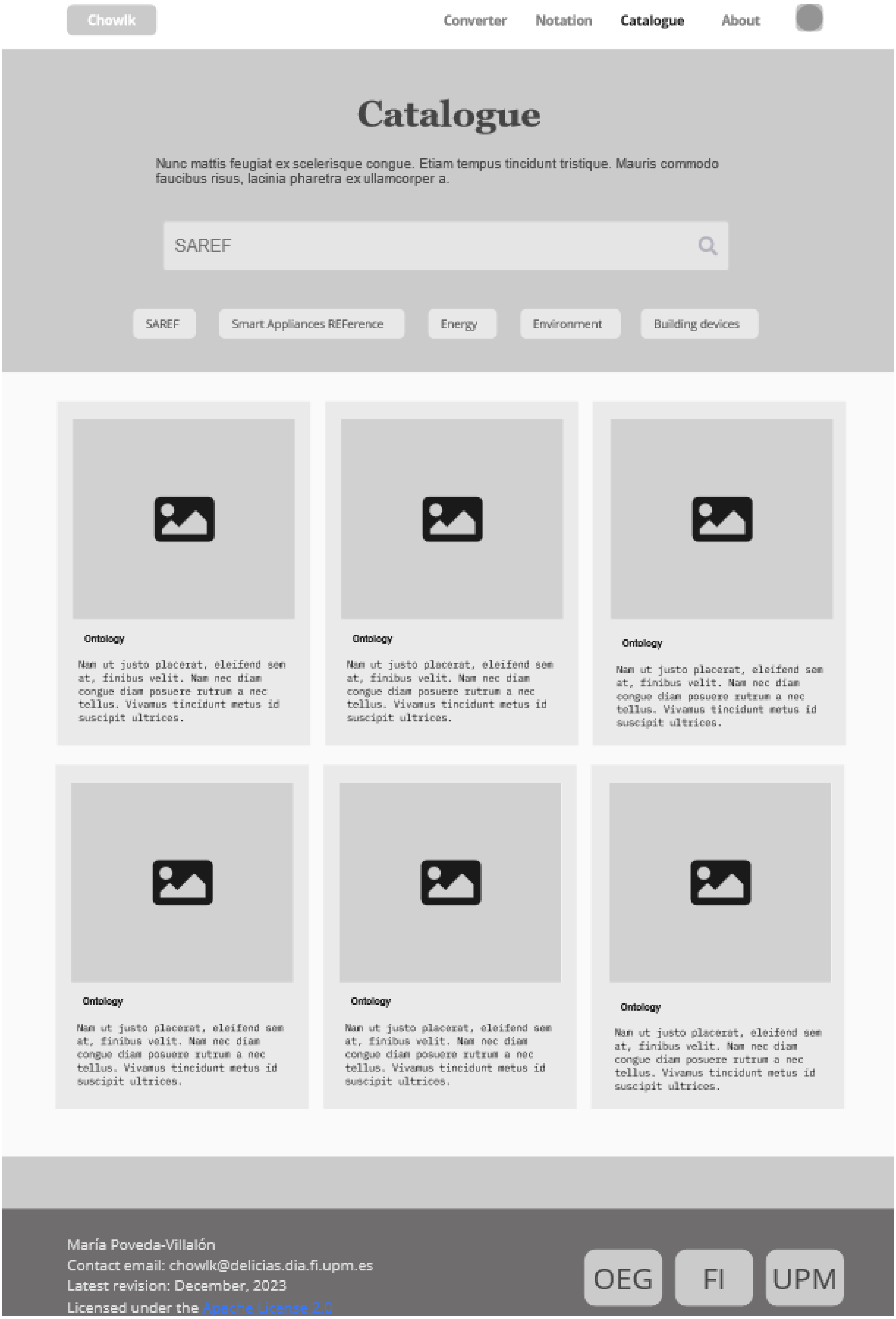

Ilustración 8. Pantalla de resultados de la búsqueda de la Versión 2 del prototipo de baja fidelidad

5.2.1.3 Ontología

Una vez que el usuario selecciona una ontología específica del catálogo, ya sea desde la pantalla principal o desde los resultados de búsqueda, accede a una vista detallada de la ontología. Esta sección presenta información sobre la ontología seleccionada, como su descripción, un diagrama de conceptualización y, en caso de que los hubiera, los patrones.

5.2.1.3.1 Versión 1

Como se puede ver en la Ilustración 9. En esta primera versión de la vista detallada de la ontología mostraba los patrones asociados en un formato de listado simple, similar a como se mostraban las ontologías en el catálogo. Cada patrón se presentaba con su título y una descripción, junto a un botón para expandir los detalles del patrón y poder visualizarlo y ver más detalles. El botón desplegaba una previsualización más amplia del patrón, incluyendo su diagrama, como se observa en la Ilustración 10. Si bien este diseño permitía acceder a la información detallada, la visualización de los patrones estaba inicialmente oculta, requiriendo una acción adicional. Este diseño no ofrecía una visualización rápida del patrón lo que puede hacer que la selección de un patrón deseado fuese menos interactiva.

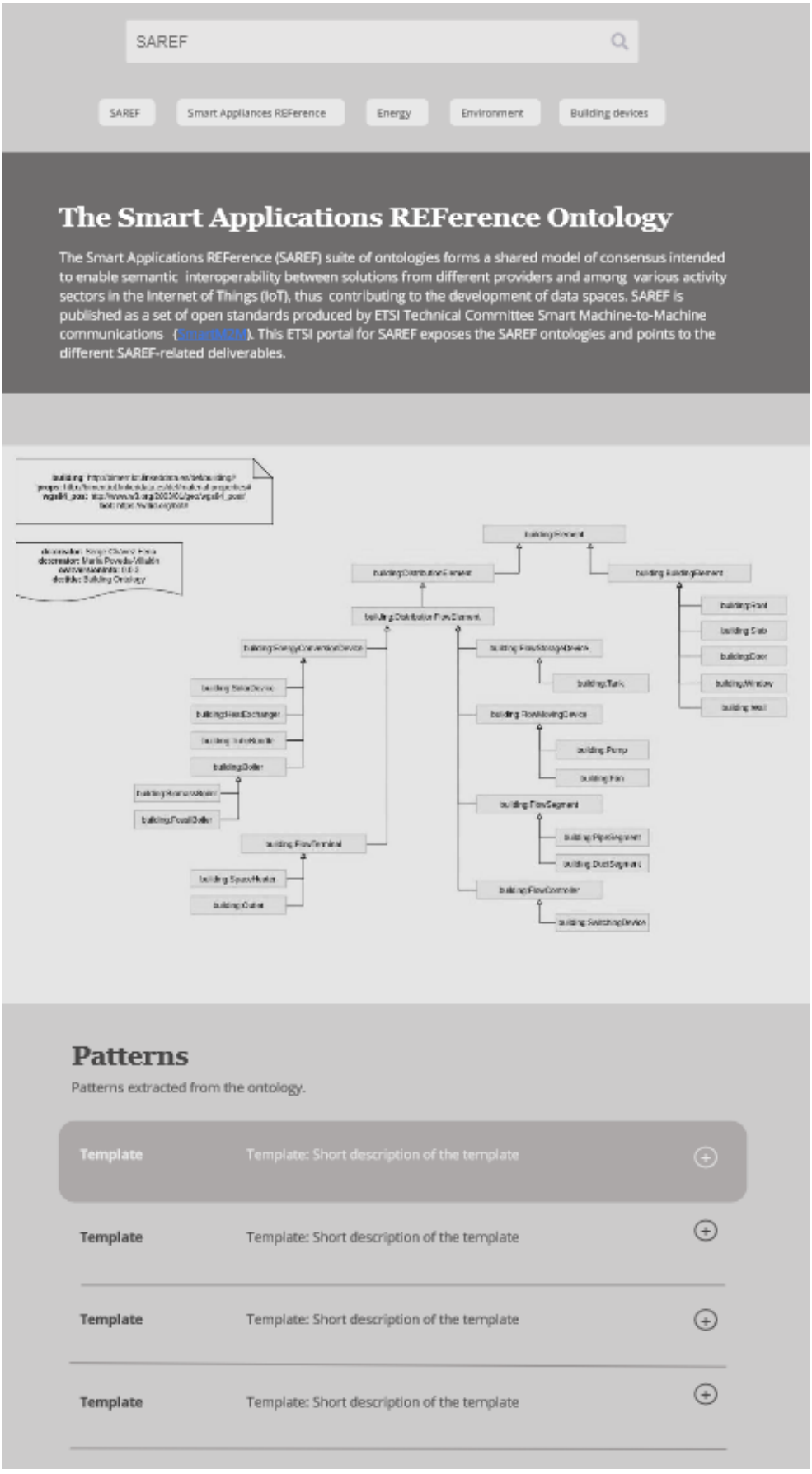

Ilustración 9. Pantalla de ontología de la Versión 1 del prototipo de baja fidelidad

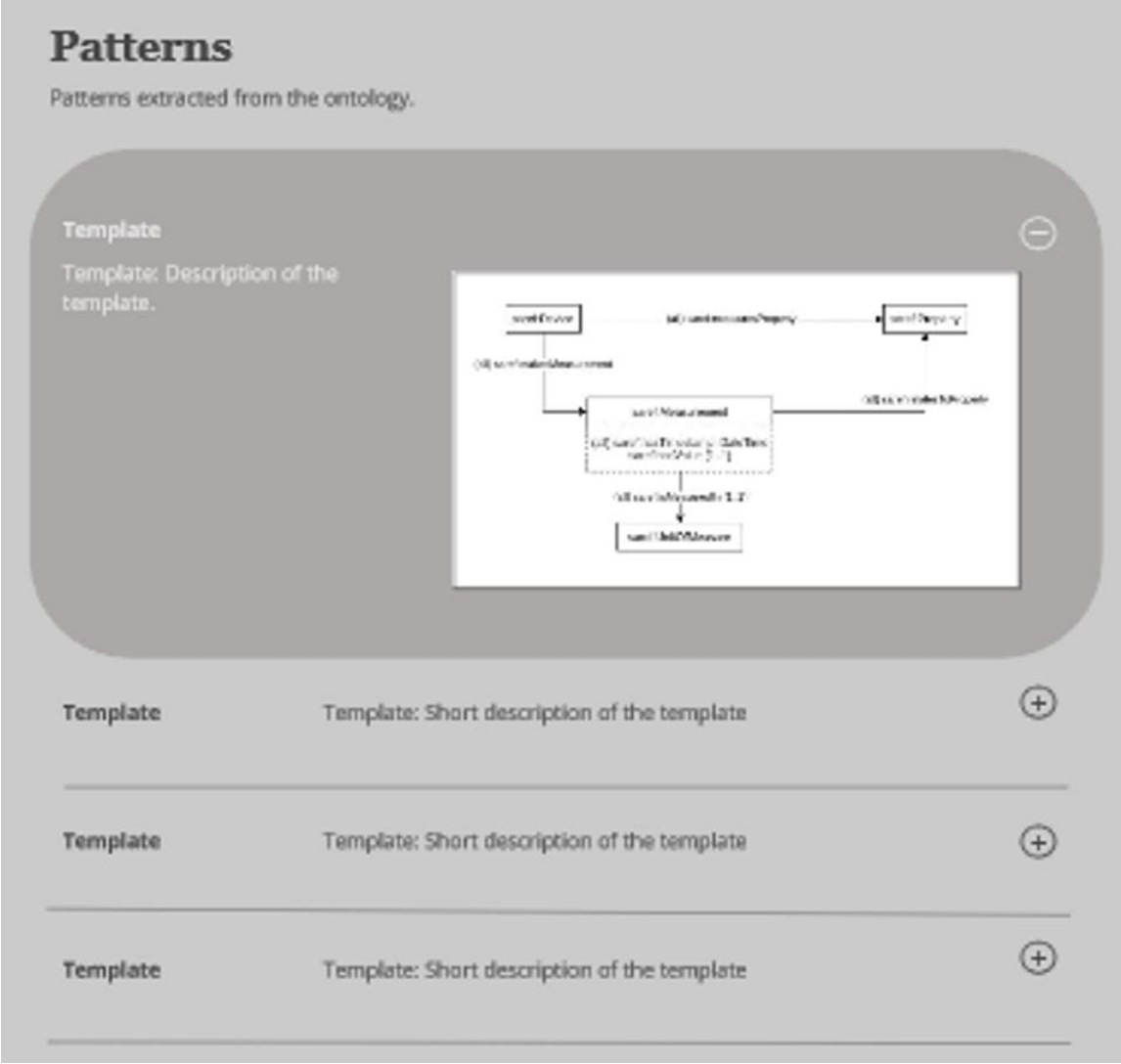

Ilustración 10. Vista expandida de un patrón de ontología de la Versión 1 del prototipo de baja fidelidad.

5.2.1.3.2 Versión 2.

Debido a estas limitaciones de la versión inicial, la versión final del prototipo de baja fidelidad, que se muestra a continuación en la ilustración correspondiente a la Ilustración 11, tiene un enfoque más visual y coherente con el diseño del resto del catálogo, presentando los patrones en formato de tarjetas. Cada patrón se presenta en su propia tarjeta modular que incluye la incrustación del archivo de draw.io del patrón y un título y una descripción concisa. Este diseño con tarjetas facilita al usuario la exploración y el descubrimiento de patrones relevantes de una forma más atractiva e intuitiva. Además, nuevamente, la coherencia del uso de tarjetas a lo largo de toda la aplicación contribuye a una experiencia de usuario unificada y predecible.

Ambas versiones comparten la sección superior con la información clave de la ontología (nombre, descripción, y diagrama principal), manteniendo un diseño común para el *header* y *footer* que asegura la consistencia de la interfaz en toda la aplicación.

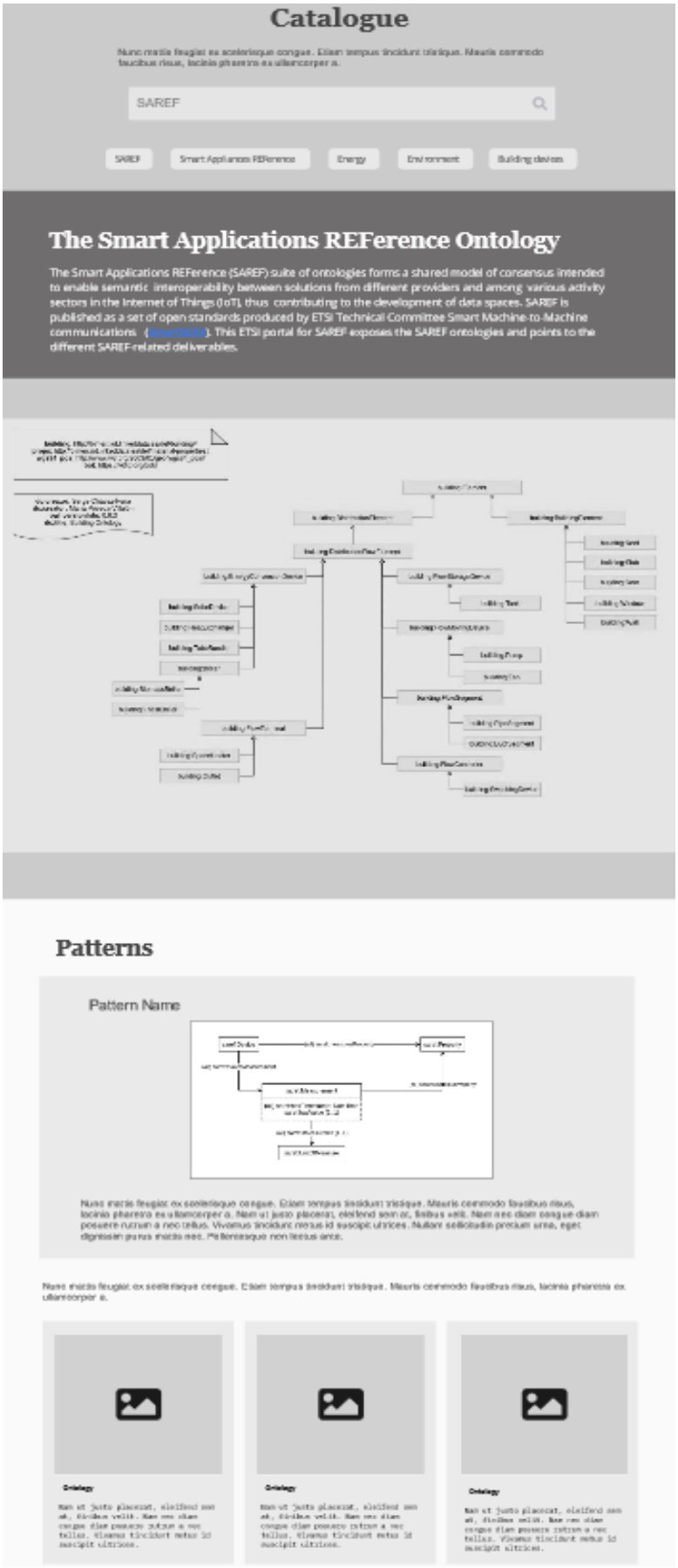

Ilustración 11. Vista expandida de un patrón de ontología de la Versión 2 del prototipo de baja fidelidad

5.2.2 Prototipo de alta fidelidad

Una vez analizado y validado el prototipo de baja fidelidad, se ha procedido a la realización de un prototipo de alta fidelidad. Esta clase de prototipos se realizan fundamentalmente con el objetivo de tener una imagen mucho más precisa y realista del diseño del nuevo sistema. Son prototipos que, aunque tienen un mayor coste en tiempo y recursos debido a su nivel de detalle y similitud con el producto final, son imprescindibles para visualizar el diseño definitivo, validar la experiencia de usuario con mayor exactitud y realizar las correcciones necesarias antes de proceder con la fase de implementación de Ágora.

Este prototipo se diferencia de anterior principalmente por su alto grado de exactitud en la apariencia y el comportamiento respecto del sistema final. Mostrará una interfaz de usuario prácticamente idéntica o muy similar a la interfaz que se desarrollará, incorporando elementos gráficos definitivos, tipografías, paleta de colores, márgenes y los logos reales que se utilizarán.

Para la realización del prototipo de alta fidelidad se ha tomado como base el diseño de la versión 2 del prototipo de baja fidelidad. Se han incorporado los cambios estéticos y de detalle necesarios para alcanzar el realismo buscado en este tipo de prototipos. Al igual que para el de baja fidelidad, la creación de esta versión de alta fidelidad se ha realizado mediante la herramienta Figma, ya que facilitaba la transición entre ambos prototipos.

5.2.2.1 Catálogo

La pantalla de inicio o pantalla principal del catálogo que se muestra a continuación corresponde a la primera interfaz que encuentra el usuario al acceder a Ágora.

Como se observa en la Ilustración 12, el diseño mantiene y mejora el concepto de la versión 2 del prototipo de baja fidelidad, donde las ontologías se presentan en formato de tarjetas. Sin embargo, en este se incorporan las especificaciones de diseño finales. Se han aplicado los colores corporativos y las tipografías definitivas. La vista incorpora el *header* y el *footer* definitivos, que serán consistentes en todas las páginas. El *header* incluye el logo de Ágora, que actúa también como elemento de navegación al inicio, así como otros elementos de navegación como *"Catalogue", "About", "How to"* y un enlace a Chowlk y a GitHub. El *footer* contiene información de contacto, la licencia y los logos de las instituciones pertinentes (OEG, FI, UPM). En la parte superior, se puede ver el nombre de la aplicación junto con un mensaje de bienvenida. Además, se mantiene la barra de búsqueda con la que los usuarios pueden introducir consultas y filtrar catálogo. Las tarjetas de las ontologías tienen los logos reales de las organizaciones (como COGITO o BIMERR), incorporan el nombre de la ontología, su descripción y un botón de *"View Details"* para acceder a la vista detallada de la ontología.

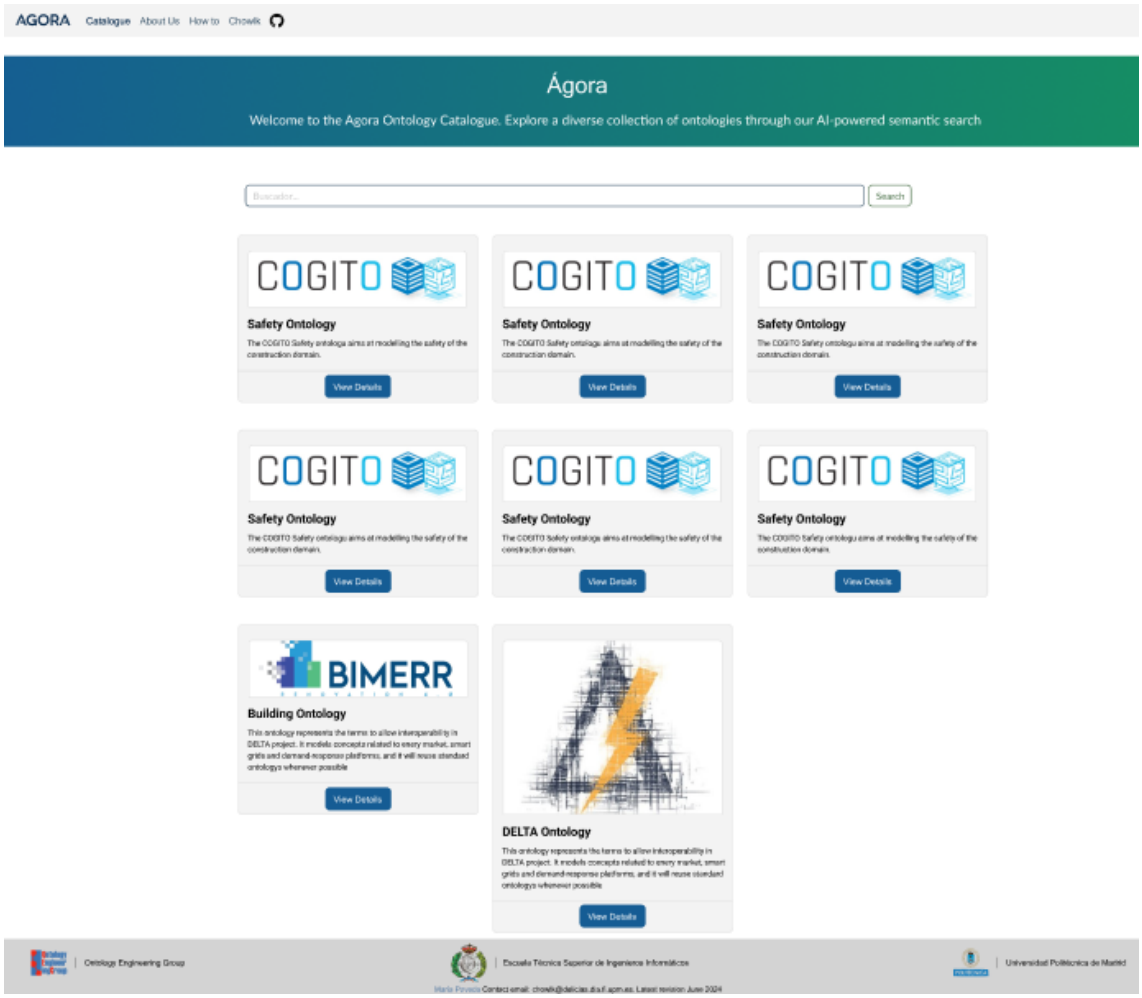

Ilustración 12. Pantalla de inicio del catálogo del prototipo de alta fidelidad

5.2.2.2 Ontología

Una vez que el usuario hace clic en el botón "*View details*" de una ontología concreta del catálogo, accede a una vista detallada de la ontología. Esta sección presenta información sobre la ontología seleccionada, como su nombre, descripción, imagen del diagrama de conceptualización y, en caso de que los hubiera, los patrones. Se puede ver en la Ilustración 13.

La vista mantiene la consistencia del *layout* general de la aplicación, incluyendo el *header* y el *footer* unificados. El cuerpo central contiene la información específica de la ontología seleccionada, presentando su nombre, una descripción y una serie de botones, *"Download XML" y "Download OWL"*, que permiten descargar los archivos de la ontología; y "*Reuse Ontology/ View Image",* que permiten al usuario cambiar entre las diferentes opciones de visualización de la conceptualización de la ontología.

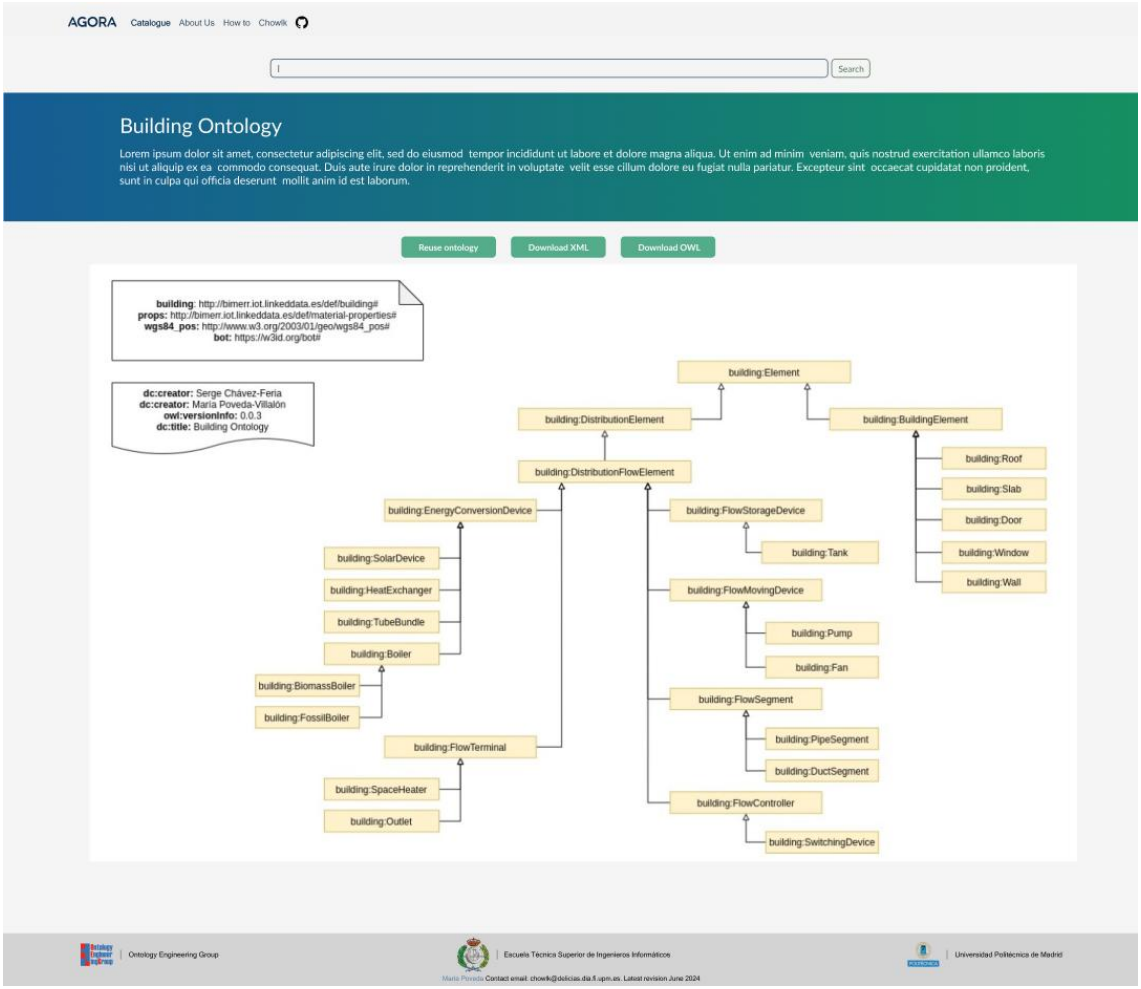

Ilustración 13. Pantalla de ontología del prototipo de alta fidelidad

Inicialmente, la conceptualización se muestra como una imagen, como se puede ver en la ilustración de la Ilustración 13para tener una primera inspección rápida sin necesidad de interacción. Pero, para poder reutilizar la ontología, se ha implementado una funcionalidad interactiva que incrusta un *iframe* de Draw.io con el archivo XML de la ontología previamente cargado, como se muestra en la ilustración correspondiente a la Ilustración 14. Pantalla de reutilización de la ontología del prototipo de alta fidelidad, permitiendo a los usuarios interactuar directamente con el diagrama de la ontología dentro de la propia aplicación. El *iframe* no solo carga el XML de la ontología, también importa la librería de patrones de Chowlk[40] y, si la ontología lo dispone, su propia librería de patrones, lo cual facilita la reutilización y edición de la ontología.

[40] https://chowlk.linkeddata.es/static/resources/chowlk-library-complete.xml

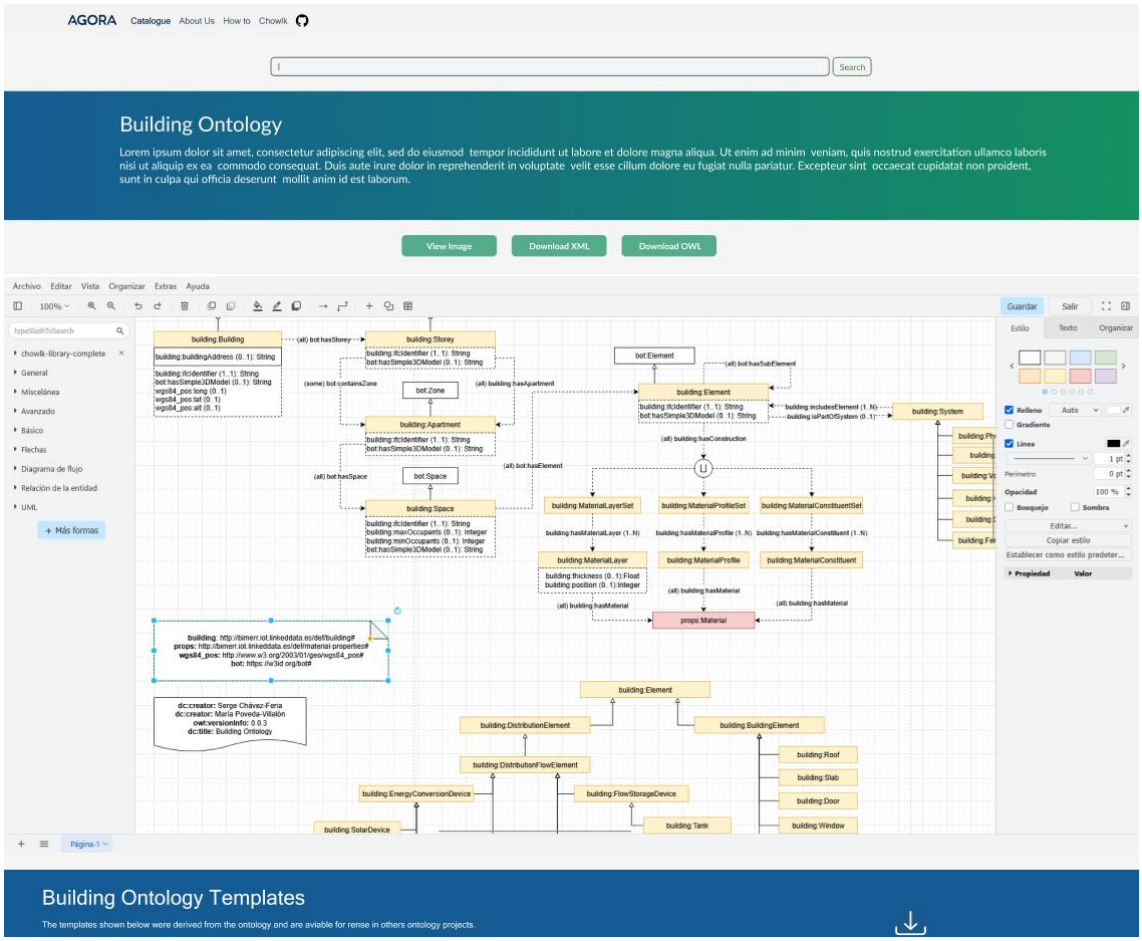

Ilustración 14. Pantalla de reutilización de la ontología del prototipo de alta fidelidad

5.2.2.3 Resultados de la búsqueda

La forma en la que se muestran los resultados de una consulta en el catálogo es un punto clave en la aplicación, ya que determina la facilidad con la que se puede encontrar y explorar la información deseada. Para este caso, el prototipo ha iterado entre dos versiones de esta vista.

Inicialmente, la pantalla de resultados de búsqueda en el prototipo de alta fidelidad mantuvo el diseño del prototipo de baja fidelidad con un formato de tarjetas, similar a la vista principal del catálogo, como se ilustra en la Ilustración 15. Aunque esta opción era coherente con el diseño general, para los resultados de una búsqueda específica, especialmente si se trata de elementos de una ontología, como clases o propiedades, un listado de tarjetas de ontologías podría no proporcionar el contexto o la información suficiente.

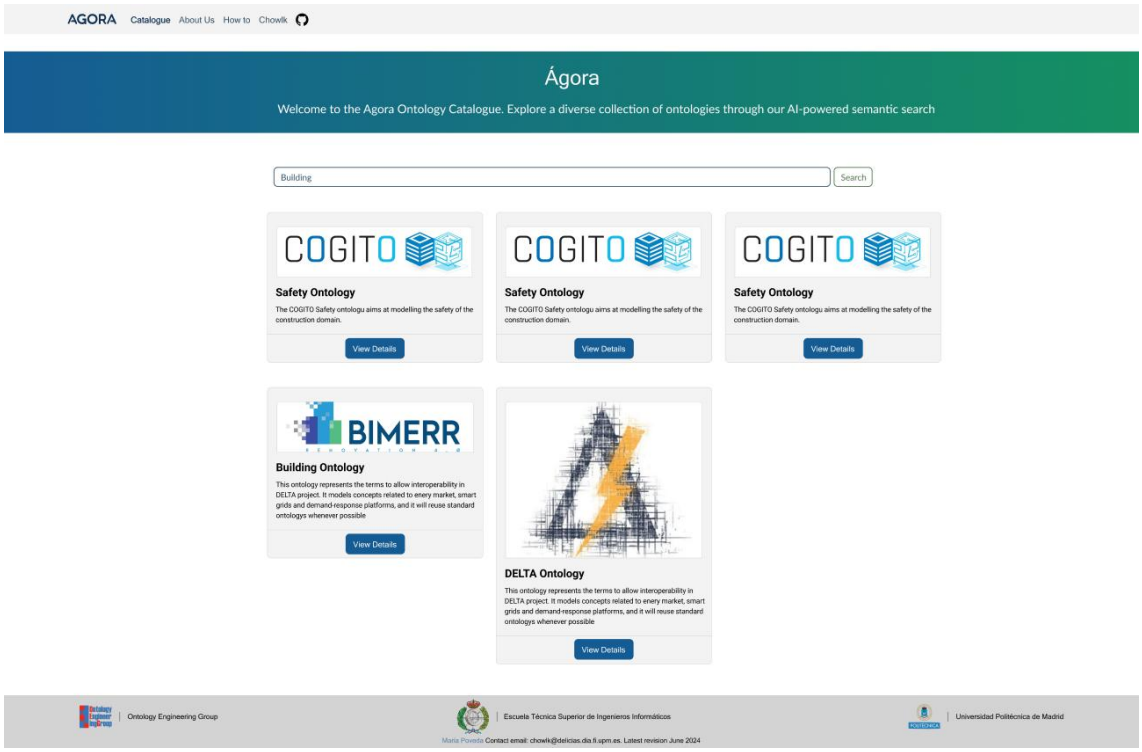

Ilustración 15. Pantalla de resultados de la búsqueda Versión 1 del prototipo de alta fidelidad

Debido a las limitaciones de esta primera versión, finalmente, como se muestra en la Ilustración 16, se optó por un diseño diferente, que se basa en dos elementos principales:

- **Panel *dropdown***. En el lado izquierdo, se puede ver un panel lateral que contiene los resultados proporcionados por Ágora de la consulta en el catálogo introducida por el usuario. Estos resultados son navegables y se agrupan según el tipo del elemento del resultado, pudiendo ser, ontologías, clases, propiedades (*datatype properties o object properties*) o metadatos. Además, la vista contiene un botón "*Hide/Show search results*", que oculta o muestra este panel para poder visualizar el resultado seleccionado más cómodamente.

- **Vista de ontología.** En el lado derecho, inicialmente, se muestra la vista detallada de la ontología que el sistema considera más relevante para la consulta (a la que pertenezca el elemento que ha obtenido un score más alto en la búsqueda). Pero a medida que el usuario seleccione algún otro resultado del panel izquierdo, la visualización de la derecha se adaptará mostrando la ontología a la que pertenece ese elemento y añadiendo una sección destacada con la información específica del elemento seleccionado (nombre y URI), proporcionando así un contexto directo y relevante del resultado dentro de su ontología.

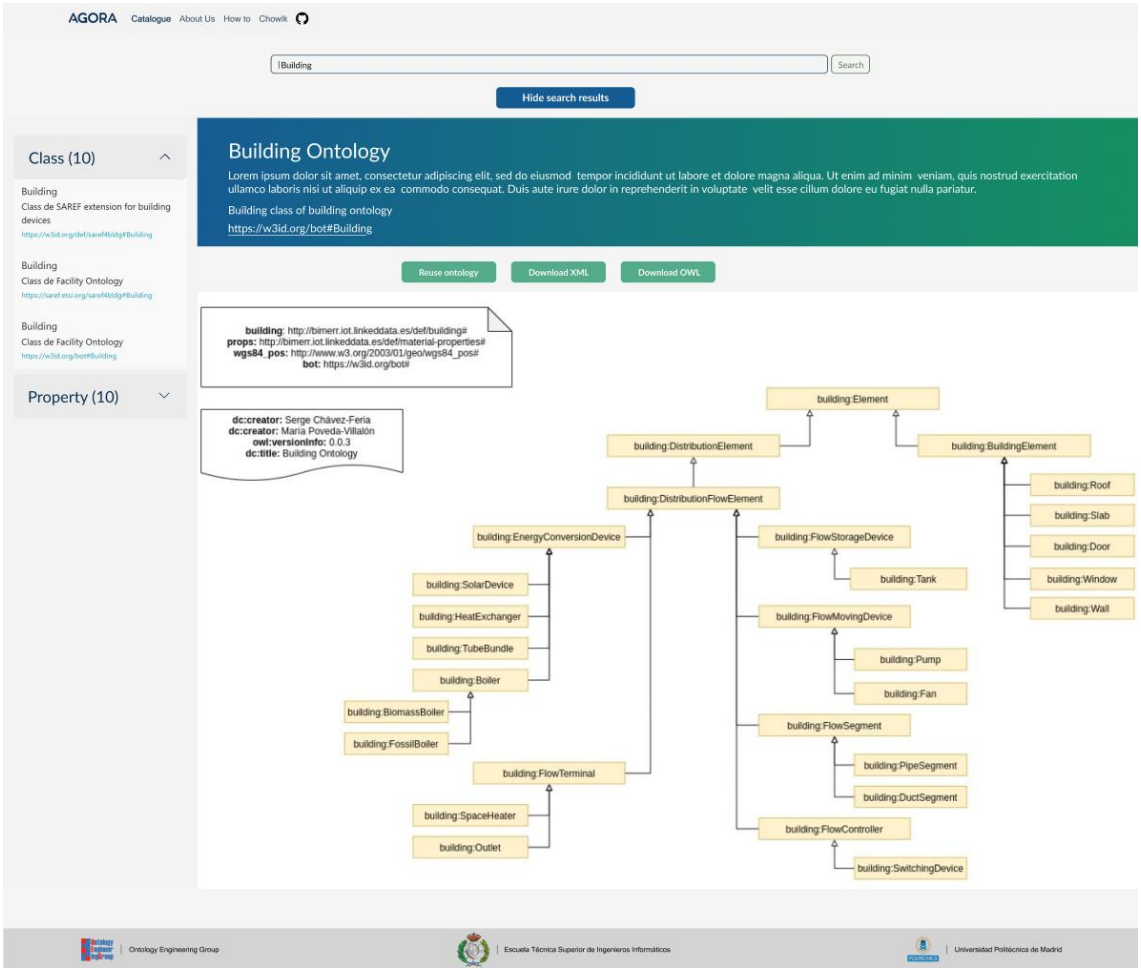

Ilustración 16. Pantalla de resultados de la búsqueda Versión 2 del prototipo de alta fidelidad

5.2.2.4 Patrones

En caso de que la ontología cuente con una librería de patrones, tanto para la vista de la ontología seleccionada desde el catálogo, como para la vista de la ontología desde los resultados de la búsqueda, se ha diseñado la siguiente vista parcial para la visualización de dichos patrones. En el diseño de esta sección cada patrón se muestra dentro de su propia tarjeta modular con su nombre, si es que lo tiene y su visualización. En vez de mostrar una imagen estática, contiene un *iframe* incrustado de Draw.io, dentro del cual se carga el XML del *template* del patrón, permitiendo la visualización e interacción con él. Además, se ha añadido un botón de descarga de la librería de templates, que puede resultar útil para los usuarios en caso de que quiera exportarla desde drawio durante el desarrollo de sus propias ontologías. Se puede ver en la Ilustración 17.

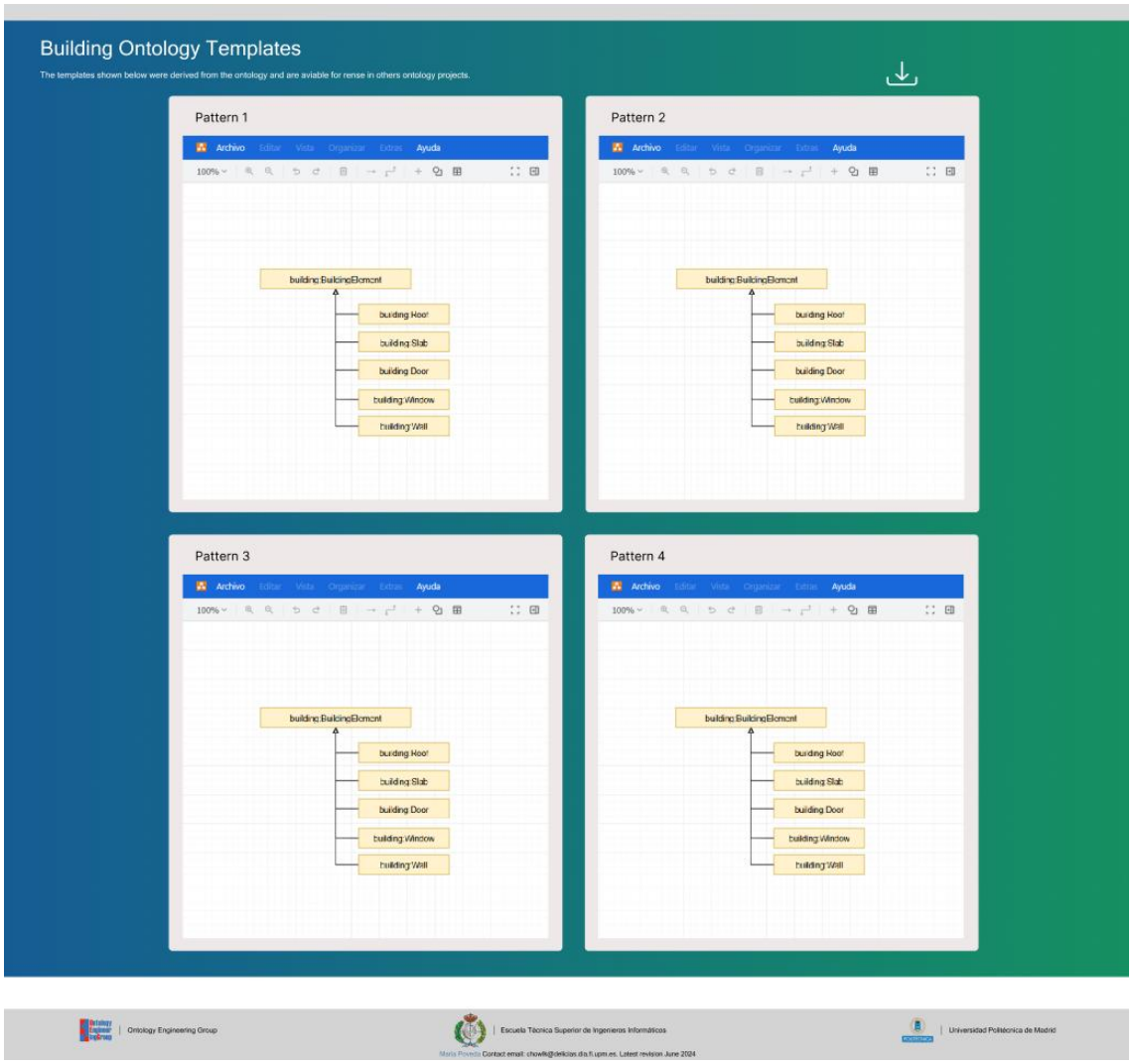

Ilustración 17. Vista parcial de los patrones de la ontología del prototipo de alta fidelidad

6 Desarrollo

A continuación se detalla el proceso de desarrollo e implementación del sistema web *Ágora.* Se explicarán las decisiones tomadas en durante este proceso para la elección de las herramientas y lenguajes utilizados para tal objetivo, con el fin de comprender el funcionamiento y la arquitectura de la aplicación.

6.1 Entorno de desarrollo

Para el desarrollo de la aplicación web propuesta se ha optado por utilizar Visual Studio Code[41] (VS Code) como entorno de desarrollo integrado (IDE). Decisión que responde a la combinación de diversos criterios técnicos, prácticos y de eficiencia.

Visual Studio Code es un editor de código fuente desarrollado por Microsoft que tiene una alta compatibilidad y un soporte avanzado con Python[42] y otros lenguajes utilizados para el desarrollo de la herramienta. Además, dispone de extensiones específicas para *frameworks* como Flask[43] y FastAPI[44] lo convierten en una opción muy adecuada para este proyecto.

A diferencia de otros entornos que ofrecen características similares, VS Code presenta un menos consumo de recursos, lo que aumenta velocidad de ejecución y facilita el desarrollo con recursos limitados. También, permite lanzar servidores de desarrollo y la depuración de aplicaciones web, lo cual ayuda a mejorar la productividad y la resolución de errores. Además el soporte integrado para Git facilita el control de versiones directamente desde el editor.

En definitiva, las características de Visual Studio Code, expuestas anteriormente, hacen que sea una herramienta especialmente adecuada para el desarrollo de Ágora.

6.2 Lenguajes

Para la implementación del sistema se ha utilizado, principalmente el lenguaje de programación Python, cuya elección se explicará a continuación.

Además, por tratarse de una aplicación web, para el desarrollo del *frontend,* para construir la interfaz de usuario y mejorar la experiencia y usabilidad, se han utilizado otros lenguajes complementarios: HTML, CSS y JavaScript.

- HTML[45]. Lenguaje de Marcado de Hipertexto. Es un lenguaje de marcado descriptivo escrito en forma de etiquetas se utiliza para especificar la estructura básica de una página, definir los elementos visuales de la aplicación y desplegar los contenidos de las páginas web.

[41] https://code.visualstudio.com/
[42] https://www.python.org/
[43] https://flask.palletsprojects.com/en/stable/
[44] https://fastapi.tiangolo.com/
[45] https://www.w3.org/html/

- CSS[46]. *Cascading Style Sheets*. Se trata de un lenguaje de hojas de estilo en cascada. Se utiliza para dar estilo a los elementos descritos en los lenguajes de marcado, para este caso se ha utilizado HTML. Se denomina Hoja de Estilos en Cascada porque las características establecidas se aplican de arriba a abajo mediante un esquema de prioridad. Con este lenguaje se da estilos a la aplicación, permitiendo establecer características como los colores de fondo, tamaño y color de fuente, posición de los elementos, etc.

- JavaScript[47]. Este lenguaje de secuencias de comandos para la web es comúnmente usado para complementar a HTML y CSS en la creación de páginas web. Esto es debido a que los archivos .js se ejecutan directamente en un navegador web. Este lenguaje permite añadir interactividad a la interfaz de usuario mediante validaciones dinámicas, llamadas asíncronas (AJAX/Fetch) a la API y actualizaciones dinámicas de contenido sin recarga.

Esta combinación de tecnologías es muy habitual en el desarrollo web actual y ayuda a separar las responsabilidades de lógica del negocio (en Python) y interfaz (en HTML/CSS/JS), lo que permite que la herramienta tenga una estructura modular y que sea más escalable y mantenible.

6.2.1 Python

La elección de Python como lenguaje principal se debe a diversas cuestiones. Se trata de un lenguaje ampliamente adoptado en el ámbito científico y tecnológico, especialmente en inteligencia artificial.

Cuenta con una gran cantidad de librerías y *frameworks* que facilitan el proceso de implementación del sistema, tanto para el *frontend*, que se ha utilizado Flask, como para el *backend*, que consiste en una API REST con FastAPI. Además, para el proceso de búsqueda en el catálogo, se ha realizado mediante técnicas de procesamiento de lenguaje natural para generar representaciones vectoriales (*embeddings*) de textos, utilizando la biblioteca *sentence-transformers* y una base de datos vectorial Qdrant[48], para búsquedas por similitud semántica que cuenta con un cliente en Python (*qudrant-client*) que facilita y mejora la indexación y consulta de vectores.

6.3 Librerías y *frameworks*

Como se ha comentado anteriormente, y se explicará más adelante, para el desarrollo de *Ágora* se ha optado por una arquitectura modular, separando la interfaz web (correspondiente al *frontend* de la aplicación) de la lógica de negocio y los servicios, (correspondiente al *backend* de la aplicación). Para desarrollar ambas partes se han utilizado dos *frameworks* comúnmente utilizados en aplicaciones en Python*:* Flask *y* FastAPI.

---

[46] https://www.w3.org/Style/CSS/Overview.en.html
[47] https://www.javascript.com/
[48] https://qdrant.tech/

6.3.1 Flask

Flask[49] es un *micro-framework* de desarrollo web para Python que se ha elegido para el desarrollo del *frontend* de Ágora. Integra el sistema de plantillas Jinja2[50], que permite la generación y renderización dinámica de páginas HTML a partir de los datos proporcionados por el *backend*, lo que ha facilitado la construcción de interfaces dinámicas de forma rápida y sin necesidad de configuraciones complejas. Además, permite servir archivos estáticos (como hojas de estilo CSS, scripts JavaScript o imágenes) de forma directa, lo que facilita la gestión de recursos visuales.

En este proyecto, se ha utilizado principalmente para gestionar las rutas HTTP que renderizan las vistas HTML, incluyendo la paginación y la personalización de la presentación de los resultados de búsqueda, la visualización de ontologías y otros componentes del catálogo. También se encarga de gestionar la interacción con el usuario en la capa de la vista y realizar las llamadas correspondientes a la API del backend.

Su naturaleza modular se alinea perfectamente con la arquitectura de responsabilidades separadas adoptada en Ágora, favoreciendo un desarrollo limpio, mantenible y escalable.

6.3.2 FastAPI

FastAPI[51] es un *framework* de desarrollo web moderno y eficiente para construir APIs con Python, que destaca por su rendimiento, facilidad de uso y por satisfacer los requisitos de una aplicación que trabaja con datos estructurados y búsquedas semánticas. Ha sido utilizado para implementar el *backend* y los servicios que gestionan la lógica de negocio del sistema.

Una de sus principales ventajas es la generación automática de documentación interactiva de la API mediante SwaggerUI[52], lo que proporciona una descripción clara y actualizada de todos los *endpoints*, sus parámetros y respuestas; eso facilita el procedo de desarrollo y de testeo de la aplicación así como la integración de servicios.

Además, FastAPI soporta de forma nativa la programación asíncrona (*async/await*), lo cual permite manejar múltiples solicitudes concurrentes sin bloquear el hilo principal. Esto es esencial para un *backend* que realiza operaciones intensivas de E/S, como las consultas a la base de datos vectorial Qdrant o el procesamiento de datos como los del repositorio de ontologías agora-ontos.

Otra ventaja de FastAPI es que proporciona la infraestructura necesaria para integrar la librería sentence-transformers, utilizada para la generación de *embeddings,* y con el cliente *qdrant-client* para la interacción con la base de

[49] https://flask.palletsprojects.com/en/stable/
[50] https://jinja.palletsprojects.com/en/stable/
[51] https://fastapi.tiangolo.com/
[52] https://swagger.io/docs/open-source-tools/swagger-ui/development/setting-up/

datos vectorial Qdrant, permitiendo realizar las operaciones para ofrecer las capacidades avanzadas de búsqueda semántica sobre el catálogo de ontologías.

6.4 Otras herramientas

6.4.1 Bootstrap

Bootstrap[53] es un *framework* de diseño web de código abierto que proporciona una gran colección de estilos predefinidos y componentes responsivos desarrollados con CSS, JavaScript y HTML. Es ampliamente utilizado en el desarrollo *frontend* para crear interfaces limpias, coherentes y adaptables a múltiples dispositivos sin tener que diseñarlas desde cero. Es decir, es un conjunto de librerías que ofrecen funciones y herramientas que acelera significativamente el proceso de desarrollo ya que proporcionan una base sólida y estandarizada para los elementos de la interfaz. Permite ofrecer al usuario una experiencia más agradable cuando navega por un sitio web. Esto se consigue gracias a que Bootstrap permite crear interfaces más limpias y con un diseño responsive.

En el desarrollo de la interfaz de Ágora, tal como se definió en el prototipo de alta fidelidad, se han utilizado varios de los componentes de Bootstrap, entre los que destacan:

- **Botones** (*btn, btn-primary*) para acciones como visualizar detalles de una ontología, reutilizar una ontología, mostrar u ocultar los resultados de la búsqueda y descargar archivos como los .XML o .OWL de cada ontología o su librería de patrones.
- **Barra de navegación** (*navbar*) situada en el *header* de la aplicación, ofrece una navegación intuitiva entre las distintas secciones de Ágora.
- **Cards** (*card*) utilizadas para mostrar la información individual de cada ontología o patrón, incluyendo su título, descripción e imagen.
- **Sistema de rejillas** (*grid*) (container, row, col, g-4, etc.) para organizar el contenido en columnas y filas de forma responsiva.
- **Acordeón** (*accordion*) utilizado en el panel izquierdo de la vista de resultados de la búsqueda para mostrar los resultados de la consulta.
- **Iconos** (*Bootstrap Icons*) Como el icono de GitHub (bi-github) del *navbar* o el icono de descarga de la librería de patrones (bi-download)

[53] https://getbootstrap.com/

7 Evaluación y resultados

La evaluación del sistema resulta fundamental para comprobar si se han alcanzado los objetivos propuestos, si la herramienta funciona correctamente y si proporciona una experiencia de usuario adecuada. En este caso, debido a la naturaleza del proyecto y a las limitaciones de tiempo, no fue posible realizar un estudio de usabilidad formal con usuarios finales.

Sin embargo, se optó por una estrategia de validación práctica, basada en dos enfoques complementarios: por un lado, la realización de pruebas manuales durante el desarrollo; y por otro, la validación iterativa del sistema junto con los directores del trabajo, que actuaron como usuarios expertos y proporcionaron retroalimentación continua a lo largo del proceso.

Estas acciones permitieron comprobar el correcto funcionamiento de las funcionalidades implementadas, evaluar la solidez del procesamiento de los datos y ajustar aspectos clave de la interfaz, incorporando las sugerencias recibidas para mejorar la usabilidad y la coherencia general de la herramienta.

7.1 Pruebas de Desarrollo Internas

Durante todo proceso de desarrollo, se llevaron a cabo paralelamente pruebas para asegurar la calidad y el correcto funcionamiento de cada componente del sistema Ágora. Estas pruebas se centraron en:

1. **Procesamiento del repositorio de ontologías**. Se comprobó que la tarea asíncrona encargada de detectar y sincronizar cambios en el repositorio remoto agora-ontos funcionaba según lo previsto. Se validó tanto la clonación inicial del repositorio como las actualizaciones mediante pull, así como la copia automática de archivos estáticos (imágenes de diagramas y logotipos) en la carpeta correspondiente del frontend. Para ello, se realizaron pruebas con la adición, modificación y eliminación de ontologías o archivos en el repositorio, observando que los cambios se reflejaban correctamente en el catálogo de la aplicación.

2. **Extracción y análisis de metadatos**. Se verificó que la librería RDFLib interpretara adecuadamente los archivos OWL. Se comprobó que la información esencial, como el título, la descripción, el autor, la licencia, la versión, así como las clases y propiedades de las ontologías, se extrajera y almacenara correctamente en los objetos internos utilizados por la aplicación. Estas pruebas se aplicaron sobre diferentes ontologías para asegurar la integridad y consistencia de los datos procesados.

3. **Generación de URLs y gestión de patrones**. Se revisó que las URLs (raw) a los archivos en GitHub se generaran correctamente, permitiendo su descarga o visualización integrada. También, en los casos en los que las ontologías incluían una librería de patrones, se validó su procesamiento e incrustación individualizada en los *iframes* embebidos de Draw.io.

4. **Actualización de la caché** y **base de datos vectorial.** Se confirmó que, tras cada procesamiento del repositorio, los datos se almacenaban en caché para su consulta desde la interfaz web, y que la colección vectorial

se reconstruía adecuadamente, garantizando la coherencia entre el contenido del catálogo y el sistema de búsqueda semántica.

5. **Funcionalidades de la interfaz web** Se probaron todas las interacciones de usuario, incluyendo la navegación por el catálogo, la funcionalidad de búsqueda, la visualización detallada de ontologías (incluyendo los *iframes* con diagramas y patrones) y la descarga de archivos. Estas pruebas aseguraron que la interacción con el sistema fuese fluida, comprensible y coherente con el propósito de la herramienta.

6. **Evaluación de la Interfaz de Usuario (UI) y Experiencia de Usuario (UX).** Se comprobó la consistencia visual del sistema, asegurando el uso uniforme de la paleta de colores definida y de componentes ampliamente utilizados en diseño web actual, como tarjetas, sombras y jerarquías tipográficas, los cuales están respaldados como buenas prácticas de diseño y usabilidad. Asimismo, se realizaron pruebas exploratorias para validar la intuitividad de la navegación, confirmando que la disposición de los elementos resultara lógica y que las acciones ofrecieran una experiencia predecible para el usuario. Por último, se revisó la presentación de los datos, prestando especial atención a la legibilidad de los textos y a la claridad de las visualizaciones empleadas, lo que permitió asegurar una correcta interpretación de la información representada en el catálogo.

7. **Pruebas de búsqueda semántica.** Se llevaron a cabo pruebas para validar el comportamiento del sistema de búsqueda semántica, con el objetivo de comprobar que las consultas devolvían resultados relevantes y coherentes con la intención del usuario. Estas pruebas incluyeron tanto búsquedas dirigidas, por ejemplo, utilizando nombres específicos de clases o propiedades existentes en las ontologías del catálogo, como consultas expresadas en lenguaje natural del tipo *"quiero una ontología que hable sobre edificios"*. Las pruebas se realizaron en español e inglés, confirmando que el sistema era capaz de interpretar ambos idiomas y recuperar resultados conceptualmente relevantes, incluso cuando no coincidían literalmente con los términos utilizados en las ontologías. Se validó también que los elementos devueltos estuvieran correctamente contextualizados dentro de su ontología, incluyendo su URI, tipo (clase, propiedad, etc.) y vista detallada asociada.

7.2 Validación con usuarios expertos.

Aunque no se contó con la participación de usuarios finales externos en la fase de evaluación, sí se realizaron sesiones de validación con usuarios expertos involucrados directamente en el desarrollo y supervisión del proyecto. Estas sesiones, llevadas a cabo de forma iterativa durante las reuniones de seguimiento, permitieron recoger observaciones valiosas desde el punto de vista tanto funcional como de diseño. Entre los usuarios expertos se encontraba la directora del Trabajo de Fin de Máster, cuya experiencia y conocimiento del contexto ontológico aportaron una visión especialmente útil para orientar los ajustes en el sistema.

El feedback recibido a lo largo del proceso abordó distintos aspectos clave del sistema:

- **Reutilización de patrones:** Una de las sugerencias más significativas fue reemplazar la visualización de los patrones mediante imágenes estáticas por la incrustación directa de los archivos XML en *iframes* de Draw.io, permitiendo así su reutilización directa por parte del usuario. Además, se propuso cargar automáticamente tanto la librería de patrones de la ontología como la librería general de Chowlk en el *iframe* principal de la conceptualización.
- **Visualización de resultados de búsqueda:** Se recomendó enriquecer la vista de resultados para cada elemento encontrado (clase, propiedad o metadato) incluyendo en la interfaz información contextual relevante. Se incorporó una sección específica en la vista de ontología donde se muestra el nombre del elemento destacado, su tipo (clase, propiedad, etc.) y su URI correspondiente.
- **Ajustes estéticos:** En el plano estético, se sugirió suavizar la apariencia visual del contenedor de los iframes ajustando el color de fondo, con el fin de reducir el contraste con el resto de la interfaz.
- **Terminología:** También se señalaron oportunidades de mejora en la forma de presentar ciertos metadatos y en la terminología utilizada en algunas secciones, cambiando por ejemplo el término *templates* por *patterns* para referirse a los patrones ontológicos. Para una mayor claridad y coherencia con los estándares.

Las recomendaciones recibidas durante estas sesiones fueron incorporadas progresivamente al sistema y resultaron fundamentales para mejorar tanto la usabilidad de la herramienta como la adecuación de su diseño a los objetivos del proyecto. Esta validación informal pero continuada aportó un enfoque crítico que permitió detectar detalles relevantes, pulir la experiencia de usuario y asegurar la coherencia entre la funcionalidad del sistema y las expectativas del entorno académico y profesional en el que se enmarca.

8 Conclusiones y líneas futuras

8.1 Conclusiones

Este Trabajo Fin de Máster ha abordado con éxito el desarrollo de Ágora, una aplicación web concebida como un catálogo para la gestión y reutilización de ontologías. El sistema cumple con los objetivos definidos al inicio del proyecto, ofreciendo una solución práctica y funcional a una necesidad concreta del diseño ontológico. Entre las contribuciones fundamentales de este trabajo, se destacan:

- **Desarrollo de un catálogo web funcional.** Se ha construido una plataforma web que permite acceder a un conjunto de ontologías, visualizarlas y explorarlas a través de una interfaz web clara e intuitiva.

- **Implementación de búsqueda semántica**. Se ha integrado un sistema de búsqueda avanzada que transforma tanto las consultas como el contenido de las ontologías en vectores mediante técnicas de *embedding*, utilizando el modelo all-MiniLM-L6-v2. Estos vectores se almacenan y consultan en una base de datos vectorial (Qdrant), lo que permite recuperar resultados relevantes superando las limitaciones inherentes a las búsquedas basadas únicamente en coincidencias textuales.

- **Soporte para notación Chowlk**. Ágora ha sido diseñada específicamente para catalogar y visualizar ontologías cuya conceptualización se realiza mediante la notación de Chowlk, lo que incluye la ingesta de archivos .xml, la visualización de diagramas y el tratamiento de patrones como elementos reutilizables.

- **Reutilización de patrones ontológicos**: una de las funcionalidades más relevantes es la posibilidad procesar, visualizar y permitir la reutilización de patrones ontológicos (*patterns*) definidos dentro de las ontologías, lo que aporta un valor añadido en fases tempranas del diseño.

- **Arquitectura modular y escalable**: la aplicación se ha diseñado se ha construido con una arquitectura modular que separa el *frontend* (desarrollado con Flask) y el *backend* (desarrollado con FastAPI), lo que asegura la eficiencia, facilita futuras extensiones, mejora la mantenibilidad y permite una separación clara de responsabilidades.

- **Mecanismo de contribución abierta**: el uso de un repositorio público de GitHub como repositorio central de ontologías, facilita la contribución externa de nuevas ontologías por parte de la comunidad, siguiendo una estructura predefinida para asegurar la consistencia del catálogo, fomentando así la actualización continua del catálogo y un enfoque colaborativo.

En conclusión, Ágora se presenta como una herramienta útil y funcional para el desarrollo ontológico, que satisface la necesidad de un catálogo de ontologías inteligente y reutilizable y que no sólo permite encontrar y explorar ontologías, sino también descubrir patrones, promover la reutilización y mejorar la eficiencia del diseño conceptual.

8.2 Líneas futuras

Aunque la versión actual de Ágora cumple con sus objetivos iniciales, se han identificado varias oportunidades de mejora que podrían explorarse en futuros desarrollo.

- **Formulario web para contribuciones.** Actualmente, la contribución de ontologías se gestiona directamente a través del repositorio de GitHub agora-ontos. Se plantea como posible mejora la implementación de un formulario web intuitivo dentro de Ágora que permita a los usuarios subir sus archivos directamente. Automatizando la creación de la carpeta de la ontología, la validación preliminar de los archivos y la generación de un *pull request* automatizado, facilitando así el proceso de contribución usuarios menos familiarizados con Git. Esto sería ideal para la escalabilidad de la base de ontologías del catálogo.

- **Sistema de métricas y ranking de reutilización.** Desarrollar un sistema que permita medir la popularidad y el uso de las ontologías y patrones incluidos en el catálogo, basándose en visualizaciones, descargas o referencias.

- **Ampliación del catálogo de ontologías.** Incorporar nuevas ontologías de distintos dominios para enriquecer el corpus y mejorar el rendimiento del sistema de búsqueda.

- **API pública.** Crear una API *RESTful* pública y documentada para Ágora. Para exponer los *endpoints* clave como la búsqueda semántica, la descarga de archivos o la obtención de metadatos. La API podría llamar internamente a los *endpoints* ya existentes del backend de FastAPI, facilitando la interoperabilidad con otros sistemas externos.

- **Búsqueda con Drag-and-Drop de Archivos.** Implementar un sistema de arrastrar y soltar (*drag-and-drop*) en la barra de búsqueda para que los usuarios puedan introducir archivos de ontologías, patrones (como .ttl, .rdf, .owl) o, incluso imágenes para realizar búsquedas.

- **Integración con Herramientas de Evaluación de Ontologías.** Integrar herramientas para la evaluación automática de la calidad de las ontologías, como OOPS! o validadores de principios FAIR, en el proceso de ingesta o como una funcionalidad consultable, proporcionando *feedback* a los contribuidores y mejorando la calidad general del catálogo.

9 Bibliografía

[1] R. Studer, V. R. Benjamins, and D. Fensel, “Knowledge engineering: Principles and methods,” *Data & Knowledge Engineering*, vol. 25, no. 1–2, pp. 161–197, 1998. doi: 10.1016/S0169-023X(97)00056-6

[2] S. Chávez-Feria, R. García-Castro, and M. Poveda-Villalón, "Chowlk: From UML-Based Ontology Conceptualizations to OWL," in The Semantic Web, vol. 13489, A. Harth, C. Kirrane, A. Ngonga Ngomo, H. Paulheim, A. Rula, and A. Sequeda, Eds. Cham: Springer, 2022, pp. 338–352. [Online]. Available: https://doi.org/10.1007/978-3-031-06981-9_20

[3] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowl. Acquis.*, vol. 5, no. 2, pp. 199–220, 1993

[4] M. Poveda-Villalón and D. Garijo, “Best Practices for Implementing FAIR Vocabularies and Ontologies on the Web,” *Semantic Web*, vol. 13, no. 2, pp. 307–323, 2022. doi: 10.3233/SW-210434t

[5] O. Lassila and R. R. Swick, “Resource Description Framework (RDF) Model and Syntax Specification,” W3C Recommendation, Feb. 1999. [Online]. Available: https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[6] D. L. McGuinness and F. van Harmelen, “OWL Web Ontology Language Overview,” W3C Recommendation, Feb. 2004. [Online]. Available: https://www.w3.org/TR/owl-features/

[7] B. Motik, R. Shearer, and I. Horrocks, “Hypertableau reasoning for description logics,” J. Artif. Intell. Res., vol. 36, pp. 165–228, 2009. doi: 10.1613/jair.2802

[8] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, “Pellet: A practical OWL-DL reasoner,” J. Web Semant., vol. 5, no. 2, pp. 51–53, 2007. doi: 10.1016/j.websem.2007.03.004

[9] M. Uschold and M. Gruninger, “Ontologies: Principles, methods and applications,” Knowledge Engineering Review, vol. 11, no. 2, pp. 93–136, 1996.

[10] M. Poveda-Villalón, A. Gómez-Pérez, and M. C. Suárez-Figueroa, “OOPS! (Ontology Pitfall Scanner!): An on-line tool for ontology evaluation,” International Journal on Semantic Web and Information Systems, vol. 10, no. 2, pp. 7–34, 2014. doi: 10.4018/ijswis.2014040102

[11] U. D. C. Castro, R. C. Corcho, A. Gómez-Pérez, and O. C. Poveda, "Knowledge Representation and Management: A State of the Art," IEEE Intelligent Systems, vol. 20, no. 1, pp. 19-27, 2005.

[12] D. Beckett and T. Berners-Lee, “Turtle – Terse RDF Triple Language,” W3C Recommendation, Feb. 2014. [Online]. Available: https://www.w3.org/TR/turtle/

[13] Linked Open Vocabularies - ERCIM News, fecha de acceso: julio 10, 2025, https://ercim-news.ercim.eu/en96/special/linked-open-vocabularies

[14] D. L. Rubin, D. A. Moreira, P. P. Kanjamala, and M. A. Musen, "BioPortal: A Web Portal to Biomedical Ontologies," in *AAAI Spring Symposium on AI in Medicine*, 2008, pp. 69-74.

[15] C. Jonquet *et al.*, "AgroPortal: A vocabulary and ontology repository for agronomy," *Computers and Electronics in Agriculture*, vol. 144, pp. 126–143, Feb. 2018, doi: 10.1016/j.compag.2017.10.012.

[16] M. Codescu, E. Kuksa, O. Kutz, T. Mossakowski, and F. Neuhaus, "Ontohub: A semantic repository engine for heterogeneous ontologies," *Applied Ontology*, vol. 12, pp. 275–??, Nov. 2017, doi: 10.3233/AO-170190

[17] K. Baclawski and T. Schneider, "The Open Ontology Repository Initiative: Requirements and Research Challenges," in *CEUR Workshop Proceedings*, vol. 514, 2009.

[18] Object Management Group (OMG), *The Distributed Ontology, Modeling, and Specification Language™ (DOL), Version 1.0*, Dec. 2016. [Online]. Available: https://www.omg.org/spec/DOL/1.0/PDF

[19] J. R. Firth, "A synopsis of linguistic theory 1930–1955," in *Studies in Linguistic Analysis*, Oxford University Press, 1957.

[20] T. Mikolov, K. Chen, G. Corrado and J. Dean, "Efficient Estimation of Word Representations in Vector Space," arXiv preprint arXiv:1301.3781, 2013.

[21] T. Mikolov et al., "Distributed representations of words and phrases and their compositionality," in *Proc. of NeurIPS*, 2013.

[22] J. Pennington, R. Socher and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *Proc. of EMNLP*, 2014.

[23] J. Devlin, M. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proc. of NAACL, 2019.

[24] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in Proc. of EMNLP-IJCNLP, 2019.

[25] C. D. Manning, P. Raghavan and H. Schütze, *Introduction to Information Retrieval*, Cambridge University Press, 2008.

[26] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC '98)*, Dallas, TX, USA, May 1998, pp. 604–613, doi: 10.1145/276698.276876.

[27] E. Öztürk and A. Mesut, “Performance Analysis of Chroma, Qdrant, and Faiss Databases,” in *UniTech – Selected Papers*, Thematic Session “Computer System and Technologies”, UniTech 2024, 2024, doi:10.70456/TBRN3643.

[28] Y. A. Malkov and D. A. Yashunin, "Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 4, pp. 824-836, 1 April 2020, doi: 10.1109/TPAMI.2018.2889473.

[29] A. Gómez-Pérez, M. Fernández-López, and O. Corcho, Ontological Engineering. Berlin, Germany: Springer, 2004.

[30] «The skeptic’s guide to low-fidelity Prototyping», Smashingmagazine.com, 06-oct-2014. [En línea]. Disponible en: https://www.smashingmagazine.com /2014/10/the-skeptics-guide-to-low-fidelity-prototyping/

[31] J. Nielsen, «Mejora del poder explicativo de las heurísticas de usabilidad», 1994, pp. 152–158

10 Anexos

10.1 Anexo I. Instrucciones de contribución al repositorio Agora-ontos (Contributing.md)

 # Contributing to the `agora-ontos` repository

Thank you for your interest in contributing to the Ágora ontology catalog! This repository is designed to collect and organize ontologies conceptualized using the ****Chowlk**** notation, with the goal of facilitating their visualization, search, and reuse through the Ágora application.

Contribution Requirements

You can contribute any ontology as long as:
- It is ****conceptualized using Chowlk notation**.**
- It includes the required files in the appropriate formats.
- It follows the structure described below.

Required Structure for Each Ontology

Each ontology must be stored in its own folder within the repository, with the following files:

<pre>
/ontology-name/
│
├── ontology_name.xml   # Chowlk conceptualization file
├── ontology_name.ttl   # OWL generated from Chowlk (Turtle format)
├── ontology_name.jpg or .png   # Diagram image
├── logo.png   # (Optional) Project or source logo (must include "logo")
└── ontology_name_patterns.xml   # (Optional) Pattern library (must include "patterns")
</pre>

Alternatively, you can group multiple ontologies belonging to the same project or organization under a shared folder:

<pre>
/organization-name/
│
├── logo.png   # (Optional) Organization logo (must include "logo")
├── /ontology-name/
│   ├── ontology_name.xml   # Chowlk conceptualization file
│   ├── ontology_name.ttl   # OWL generated from Chowlk (Turtle format)
│   ├── ontology_name.jpg or .png   # Diagram image
│   ├── logo.png   # (Optional) Specific ontology logo (must include "logo")
│   └── ontology_name_patterns.xml   # (Optional) Pattern library (must include "patterns")
├── /.../   # Additional folders for other ontologies
</pre>

Ways to Contribute Ontologies

There are multiple ways to contribute to the `agora-ontos` GitHub repository.

1. GitHub Pull Request (for users familiar with Git)

This is the preferred method for users comfortable with Git and GitHub:

1. Fork the repository `agora-ontos`.
2. Clone your fork locally.
3. Create a new folder following the required structure above.
4. Add at least the following:
   - The `.xml` file with the Chowlk conceptualization.
   - The `.ttl` OWL file generated from Chowlk.
   - A preview image (`.jpg` or `.png`).
5. Commit and push the changes to your fork.
6. Open

---

2. Upload Files via GitHub Web Interface

For users not familiar with Git, files can be uploaded directly via the GitHub interface:

1. Navigate to the desired folder in the repository.
2. Click on **"Add file" → "Upload files"**.
3. Drag and drop the required files (as described above).
4. Add a brief commit message.
5. Submit the changes.

---

3. Submit via GitHub Issue

For users who prefer not to upload files themselves:

1. Open a new **issue** in the [Issues](https://github.com/oeg-upm/agora-ontos/issues) section.
2. Title it as `New ontology contribution`.
3. Provide:
   - A description of the ontology.
   - A link to download the required files.
   - Any additional relevant details.

The maintainers will assist in reviewing and integrating the ontology into the repository.