



Universidad Politécnica  
de Madrid



**Escuela Técnica Superior de  
Ingenieros Informáticos**

Grado en Ingeniería Informática

Trabajo Fin de Grado

# **Método de Análisis de Cobertura para Pruebas de Ontologías**

Autor: Diego Ciprés Martínez

Tutor(a): Raul García Castro

Madrid, 06/2024

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

*Trabajo Fin de Grado*

*Grado en Ingeniería Informática*

*Título : Método de Análisis de Cobertura para Pruebas de Ontologías*

*06/2024*

*Autor:* Diego Ciprés Martínez

*Tutor:*

Raul García Castro

ETSI Informáticos

Universidad Politécnica de Madrid

# Resumen

Este proyecto se centra en desarrollar un método de análisis de cobertura para pruebas de ontologías, un campo crucial en la ingeniería del conocimiento y la inteligencia artificial. Las ontologías, como estructuras formales que organizan el conocimiento en áreas específicas, son vitales para facilitar la comprensión y el intercambio de información entre sistemas y usuarios. Garantizar la calidad y efectividad de estas ontologías es esencial, y aquí es donde entra en juego la cobertura de ontologías. La cobertura se refiere a qué tan bien una ontología cubre todos los conceptos y relaciones relevantes de un dominio específico, asegurando que sea completa y precisa.

Nuestro proyecto ha desarrollado un método completo de análisis de cobertura, basado en investigaciones anteriores en ingeniería de software y ontología. Este método ha sido diseñado y validado a través de una serie de estudios de caso, además de ser implementado en un prototipo de software funcional. Hemos considerado aspectos teóricos, metodológicos y prácticos para evaluar la eficacia y aplicabilidad del método en diferentes escenarios.

El método propuesto para el análisis de cobertura se centra en identificar y listar todos los elementos clave dentro de una ontología, como clases, propiedades, subclases y tripletas del tipo clase-propiedad-clase. Hemos prestado especial atención a los principios de diseño de pruebas con Themis y hemos implementado herramientas que facilitan este análisis.

El prototipo de software desarrollado permite cargar ontologías y pruebas de cobertura, generando visualizaciones claras e intuitivas de los resultados. Este software está diseñado para ser adaptable y efectivo en diversos contextos, manejando incluso ontologías de gran tamaño y proporcionando resultados rápidos. Además, ofrece una interfaz amigable para los usuarios, compatible con otros sistemas y ontologías a través de estándares reconocidos, y permite la interacción dinámica con las visualizaciones para explorar los datos.

Hemos evaluado el método utilizando una variedad de ontologías reales de diferentes dominios. Los resultados obtenidos han sido analizados para validar la efectividad del método en situaciones prácticas, identificando áreas de mejora en las ontologías evaluadas. Este análisis ha demostrado que el método es eficaz en términos de exhaustividad y calidad de las ontologías, proporcionando herramientas útiles para identificar rápidamente los elementos más relevantes y críticos.

## Resumen (English)

This project focuses on developing a coverage analysis method for ontology testing, a crucial field in knowledge engineering and artificial intelligence. Ontologies, as formal structures that organize knowledge in specific domains, are vital for facilitating understanding and information exchange among systems and users. Ensuring the quality and effectiveness of these ontologies is essential, and this is where ontology coverage comes into play. Coverage refers to how well an ontology covers all relevant concepts and relationships within a specific domain, ensuring it is comprehensive and accurate.

Our project has developed a comprehensive coverage analysis method, based on previous research in software engineering and ontology. This method has been designed and validated through a series of case studies, in addition to being implemented in a functional software prototype. We have considered theoretical, methodological, and practical aspects to evaluate the effectiveness and applicability of the method in different scenarios.

The proposed coverage analysis method focuses on identifying and listing all key elements within an ontology, such as classes, properties, subclasses, and class-property-class triplets. We have paid special attention to test design principles with Themis and have implemented tools that facilitate this analysis.

The developed software prototype allows loading ontologies and coverage tests, generating clear and intuitive visualizations of the results. This software is designed to be adaptable and effective in various contexts, handling even large ontologies and providing quick results. Additionally, it offers a user-friendly interface, compatible with other systems and ontologies through recognized standards, and allows dynamic interaction with visualizations to explore the data.

We have evaluated the method using a variety of real ontologies from different domains. The results obtained have been analyzed to validate the method's effectiveness in practical situations, identifying areas for improvement in the evaluated ontologies. This analysis has demonstrated that the method is effective in terms of coverage and quality of ontologies, providing useful tools for quickly identifying the most relevant and critical elements.

## **Abstract**

En el ámbito de la ingeniería del conocimiento y la inteligencia artificial, las ontologías son esenciales para representar y organizar el conocimiento en dominios específicos. Este proyecto se centra en el desarrollo de un método integral de análisis de cobertura para pruebas de ontologías, fundamentado en investigaciones previas en ingeniería de software y ontológica. Se han llevado a cabo estudios detallados, el diseño y desarrollo del método, así como la implementación de un software de apoyo. Las evaluaciones realizadas en diversos escenarios demuestran la eficacia y aplicabilidad del método propuesto. Este trabajo profundiza en los fundamentos teóricos de la cobertura de ontologías, explora los principios de diseño de pruebas con Themis y presenta un análisis exhaustivo de casos de estudio para validar el método.

## **Abstract (English)**

In the field of knowledge engineering and artificial intelligence, ontologies play a crucial role in representing and organizing knowledge within specific domains. This project focuses on developing a comprehensive coverage analysis method for ontology testing, based on previous research in software and ontological engineering. Detailed studies have been conducted, leading to the design and development of the method, as well as the implementation of supporting software. Evaluations carried out in various scenarios demonstrate the effectiveness and applicability of the proposed method. This work delves into the theoretical foundations of ontology coverage, explores the principles of test design with Themis, and presents a thorough analysis of case studies to validate the method.

# Tabla de contenidos

|                                                                 |           |
|-----------------------------------------------------------------|-----------|
| <b>1 Introducción</b>                                           | <b>1</b>  |
| 1.1 Objetivos del Trabajo                                       | 1         |
| <b>2 Fundamentos Teóricos</b>                                   | <b>3</b>  |
| 2.1 Definición de Ontología                                     | 3         |
| 2.2 Web Semántica                                               | 4         |
| 2.3 Test de Ontologías                                          | 5         |
| 2.3.1 Verificación Sintáctica                                   | 5         |
| 2.3.2 Verificación Semántica                                    | 5         |
| 2.3.3 Verificación de Consistencia                              | 6         |
| 2.3.4 Verificación de Cobertura                                 | 6         |
| 2.3.5 Pruebas de Competencia                                    | 7         |
| 2.3.6 Cobertura de Ontologías                                   | 7         |
| <b>3 Software</b>                                               | <b>9</b>  |
| 3.1 Pruebas Unitarias                                           | 9         |
| 3.2 Pruebas de Integración                                      | 9         |
| 3.3 Pruebas de Sistema                                          | 10        |
| 3.4 Pruebas de Aceptación                                       | 10        |
| 3.5 Pruebas de Regresión                                        | 10        |
| 3.6 Otras Pruebas                                               | 11        |
| 3.7 Cobertura Software                                          | 11        |
| 3.7.1 Cobertura de Declaraciones                                | 11        |
| 3.7.2 Cobertura de Decisiones                                   | 12        |
| 3.7.3 Cobertura de Condiciones y Decisiones Modificadas (MC/DC) | 12        |
| 3.7.4 Cobertura de Caminos                                      | 13        |
| 3.8 Ontología Vs Software                                       | 13        |
| <b>4 Estado Actual</b>                                          | <b>14</b> |
| 4.1 Evaluación Basada en Métricas                               | 14        |
| 4.1.1 Tamaño de la Ontología                                    | 14        |
| 4.1.2 Profundidad y Anchura de la Jerarquía de Clases           | 14        |
| 4.1.3 Relaciones entre Clases y Propiedades                     | 15        |
| 4.1.4 Cobertura del Dominio                                     | 15        |
| 4.1.5 Consistencia                                              | 15        |
| 4.1.6 Concisión                                                 | 15        |
| 4.1.7 Comprensibilidad                                          | 16        |
| 4.2 Evaluación con Razonadores                                  | 16        |
| 4.2.1 Verificación de Consistencia                              | 16        |
| 4.2.2 Clasificación de la Jerarquía                             | 16        |
| 4.2.3 Inferencia de Conocimientos                               | 17        |
| 4.2.4 Detección de Errores y Anomalías                          | 17        |
| <b>5 Diseño del Método de Análisis de Cobertura</b>             | <b>18</b> |
| 5.1 Definición de Objetivos                                     | 18        |
| 5.2 Definición de Requisitos                                    | 19        |
| 5.2.1 Requisitos Funcionales                                    | 19        |

|                                                          |           |
|----------------------------------------------------------|-----------|
| 5.2.2 Requisitos No Funcionales                          | 19        |
| 5.3 Arquitectura Propuesta                               | 20        |
| 5.3.1 Arquitectura del Código                            | 20        |
| 1. Importaciones.                                        | 20        |
| 5.3.2 Componentes Principales                            | 21        |
| 5.4 Implementación y desarrollo del método               | 21        |
| Paso 1: Configuración Inicial                            | 21        |
| Paso 2: Definición de Funciones de Búsqueda de Tripletas | 21        |
| Paso 3: Ejecución de Tests y Formateo de Resultados      | 22        |
| Paso 4: Evaluación de Resultados y Cobertura             | 22        |
| Paso 5: Generación de Resultados en HTML                 | 22        |
| Paso 6: Vinculación de Clases con sus Tests              | 22        |
| <b>6 Desarrollo y Evaluación del Método</b>              | <b>23</b> |
| 6.1 Creación de Tests en Themis.                         | 23        |
| 6.2 Ejecución de Tests                                   | 24        |
| 6.3 Casos de Estudio y Validación del Método             | 25        |
| <b>7 Trabajo Futuro</b>                                  | <b>27</b> |
| 7.1 Resumen de Resultados                                | 27        |
| 7.2 Limitaciones y Desafíos Enfrentados                  | 27        |
| 7.3 Propuestas para Futuras Investigaciones              | 27        |
| <b>8 Conclusiones</b>                                    | <b>29</b> |
| <b>9 Referencias</b>                                     | <b>30</b> |

# 1 Introducción

En el ámbito de la ingeniería del conocimiento y la inteligencia artificial, las ontologías juegan un papel crucial en la representación y organización del conocimiento en dominios específicos. Una ontología es una descripción formal de los conceptos relevantes y las relaciones entre ellos dentro de un dominio particular. Estas estructuras no solo facilitan la comprensión del conocimiento, sino que también permiten el intercambio de información entre diversos sistemas y usuarios. Sin embargo, para asegurar que una ontología sea efectiva, es fundamental que cubra de manera completa y precisa todos los conceptos y relaciones pertinentes a su dominio. Este proyecto se enfoca en desarrollar un método integral para el análisis de cobertura en pruebas de ontologías.

## 1.1 Objetivos del Trabajo

Este trabajo tiene como propósito principal abordar la evaluación de la cobertura en ontologías mediante el desarrollo de un método de análisis integral y su implementación en un prototipo de software funcional. Los objetivos específicos incluyen:

**Evaluar el Estado Actual:** Profundizar en la realidad actual del campo de las ontologías y la cobertura, investigando las metodologías existentes y analizando sus limitaciones y desafíos. Esto permitirá identificar las mejores prácticas y definir áreas de oportunidad para la mejora.

**Diseñar un Método de Análisis de Cobertura:** Con base en los hallazgos de la investigación, diseñar un método de análisis de cobertura que sea adaptable y efectivo en una variedad de contextos. Este método establecerá métricas objetivas y estrategias de prueba que permitan evaluar la exhaustividad y calidad de las ontologías de manera precisa.

**Implementar el Método en un Prototipo de Software:** Implementar el método de análisis diseñado en un prototipo de software funcional. Este software contará con una interfaz intuitiva que facilite la carga de ontologías, la ejecución de pruebas de cobertura y la interpretación de resultados.

**Validar el Método Propuesto:** Probar el prototipo de software desarrollado con una variedad de ontologías reales de distintos dominios. El objetivo es evaluar su eficacia y utilidad en términos de identificar áreas de mejora en las ontologías evaluadas. Analizar los resultados obtenidos para validar la efectividad del método propuesto en situaciones prácticas.

A lo largo de este trabajo, se abordarán en detalle los fundamentos teóricos de la cobertura de ontologías, se explorarán los principios de diseño de pruebas con Themis y se presentará un análisis exhaustivo de casos de estudio para validar la efectividad del método propuesto.

En el ámbito de la ingeniería del conocimiento y la inteligencia artificial, las ontologías juegan un papel crucial en la representación y organización del conocimiento en dominios específicos. Una ontología es una descripción formal de los conceptos relevantes y las relaciones entre ellos dentro de un dominio particular. Estas estructuras no solo facilitan la comprensión del conocimiento, sino que también permiten el intercambio de información entre diversos sistemas y usuarios. Sin embargo, para asegurar que una ontología sea efectiva, es fundamental que cubra de manera completa y precisa todos los conceptos y relaciones pertinentes a su dominio. Este proyecto se enfoca en desarrollar un método integral para el análisis de cobertura en pruebas de ontologías.

## 2 Fundamentos Teóricos

En este punto del trabajo se pretende dar significado y explicar ciertos conceptos relevantes como puede ser la definición de ontología, la web semántica, la cobertura desde un punto de vista del software y como esta se diferenciaría de una cobertura de ontologías

### 2.1 Definición de Ontología

En el ámbito de la informática y la inteligencia artificial, una ontología es una representación formal y explícita de un conjunto de conceptos dentro de un dominio, así como las relaciones entre esos conceptos. Las ontologías son fundamentales para estructurar y formalizar el conocimiento de un dominio específico, permitiendo su comprensión y uso por parte de los sistemas informáticos.

Una ontología generalmente se compone de los siguientes elementos:

**Clases (o Conceptos):** Representan las categorías o tipos de entidades en el dominio. Por ejemplo, en una ontología de zoología, podría haber clases como Animal, Mamífero, Reptil, etc.

**Propiedades (o Atributos):** Describen características o relaciones entre las clases. Las propiedades pueden ser de dos tipos:

**Propiedades de datos:** Relaciona una clase con un tipo de dato (e.g., un Animal puede tener una propiedad edad que es un número).

**Propiedades de objetos:** Relacionan una clase con otra clase (e.g., un Animal puede tener una propiedad hábitat que está relacionada con la clase Hábitat).

**Instancias (o Individuos):** Son los objetos concretos que pertenecen a una clase. Por ejemplo, Leo podría ser una instancia de la clase León.

Además de estos elementos básicos, una ontología puede incluir restricciones (e.g., un Mamífero debe tener la propiedad tiene pelo), axiomas lógicos que establecen reglas sobre cómo los conceptos pueden relacionarse entre sí y reglas de inferencia que permiten deducir nueva información a partir de los datos existentes.

Las ontologías se utilizan para:

**Establecer un vocabulario común:** Proporcionan una terminología estandarizada que facilita la comunicación y el intercambio de información entre diferentes sistemas y aplicaciones.

**Facilitar la interoperabilidad:** Permiten que sistemas diferentes entiendan y procesen datos de manera coherente.

**Mejorar la precisión de la información:** Ayudan a garantizar que los datos se interpreten de manera consistente, reduciendo ambigüedades y errores.

## 2.2 Web Semántica

La Web Semántica es una extensión de la Web actual en la que la información está dotada de un significado bien definido, lo que permite a las máquinas procesar y "entender" los datos de la misma manera que lo harían los humanos. En este contexto, las ontologías juegan un papel crucial al proporcionar la estructura y los mecanismos necesarios para representar el conocimiento de manera formal y compartida.

Tim Berners-Lee, el creador de la Web, definió la Web Semántica como "una web de datos que puede ser procesada directamente e indirectamente por máquinas". La idea central es enriquecer los datos de la web con metadatos que describan su contenido y contexto, utilizando lenguajes y estándares como RDF (Resource Description Framework), OWL (Web Ontology Language) y SPARQL (SPARQL Protocol and RDF Query Language).

**RDF:** Proporciona un modelo para describir los recursos y sus relaciones mediante tripletas (sujeto, predicado, objeto).

**OWL:** Es un lenguaje para definir y estructurar ontologías, permitiendo representar clases, propiedades y relaciones complejas de manera más rica que RDF.

**SPARQL:** Es un lenguaje de consulta para recuperar y manipular datos almacenados en formato RDF.

En la Web Semántica, las ontologías permiten:

**Integración de Datos:** Facilitan la combinación de información de múltiples fuentes, proporcionando una comprensión común de los datos.

**Descubrimiento y Recuperación de Información:** Permiten a los sistemas encontrar y utilizar datos de manera eficiente y precisa.

**Razonamiento Automático:** Las ontologías permiten que las máquinas realicen inferencias y deducciones lógicas, descubriendo nueva información implícita a partir de los datos existentes.

## 2.3 Test de Ontologías

Los tests de ontologías son un área crucial en la ingeniería del conocimiento y el desarrollo de sistemas basados en ontologías. Estos tests son fundamentales para garantizar que una ontología esté bien diseñada, sea consistente y cumpla con los requisitos esperados.

Realizar tests de ontologías es vital para asegurar que estas estructuras sean:

- Consistentes: Libres de contradicciones internas.
- Completas: Contienen toda la información necesaria para el dominio de aplicación.
- Correctas: Representan el conocimiento de manera precisa y sin errores.
- Usables: Adecuadas para las aplicaciones que las van a utilizar.

### 2.3.1 Verificación Sintáctica

La verificación sintáctica asegura que la ontología esté correctamente escrita según las reglas del lenguaje en el que está definida, como OWL (Web Ontology Language) o RDF (Resource Description Framework). Este tipo de test es crucial para evitar errores que podrían surgir durante la carga o interpretación de la ontología en las herramientas que la consumen. Herramientas como Protégé, TopBraid Composer, y OWL API pueden realizar esta verificación automáticamente.

Ejemplo de error sintáctico: La falta de un corchete o una etiqueta mal cerrada puede hacer que la ontología no sea procesada correctamente.

Herramientas: Protégé incluye un editor que resalta errores sintácticos, mientras que reasoners como Pellet o Hermit también pueden detectar problemas sintácticos al intentar procesar la ontología.

Una ontología mal formada puede llevar a errores de interpretación y fallos en los sistemas que la utilizan [1]

### 2.3.2 Verificación Semántica

La verificación semántica se centra en la validez lógica de las definiciones y axiomas dentro de la ontología. Se asegura de que no haya contradicciones entre las relaciones y conceptos definidos, y que las inferencias lógicas que se pueden hacer a partir de la ontología sean coherentes.

Ejemplo de error semántico: Definir que una clase A es disjunta con una clase B, pero luego añadir una instancia que pertenece a ambas clases.

El análisis semántico de una ontología es esencial para asegurar que las relaciones definidas son coherentes y no conflictivas [2]

### **2.3.3 Verificación de Consistencia**

La verificación de consistencia comprueba que no existan contradicciones entre los axiomas y las instancias definidas en la ontología. Este proceso es fundamental para garantizar que la ontología no contiene errores lógicos que podrían comprometer su uso. Por ejemplo, si se define que todas las instancias de una clase deben tener ciertas propiedades, se debe verificar que todas las instancias realmente cumplan con esa condición.

Ejemplo de inconsistencia: Una ontología que define que todas las instancias de la clase "Vehículo" deben tener una propiedad "número de ruedas", pero algunas instancias carecen de esta propiedad.

.

La consistencia es un requisito mínimo para cualquier ontología que pretenda ser útil en aplicaciones prácticas [3]

### **2.3.4 Verificación de Cobertura**

La verificación de cobertura evalúa si la ontología cubre completamente el dominio de interés, asegurando que todos los conceptos y relaciones relevantes están representados. Esto es importante para garantizar que la ontología sea útil y aplicable en su dominio de aplicación. La cobertura puede ser medida comparando la ontología con un conjunto de requisitos o con otros modelos de dominio.

Ejemplo de falta de cobertura: Una ontología de medicina que no incluye conceptos críticos como "Diagnóstico" o "Tratamiento".

Una ontología debe ser suficientemente rica para representar todos los aspectos relevantes del dominio que modela [4]

### **2.3.5 Pruebas de Competencia**

Las pruebas de competencia verifican si la ontología puede responder correctamente a un conjunto de preguntas o consultas relevantes para el dominio. Este tipo de prueba se basa en el uso de consultas SPARQL o preguntas específicas que la ontología debe poder responder, asegurando que cumple con los requisitos de información que se espera de ella.

Ejemplo de prueba de competencia: En una ontología de e-commerce, preguntar "¿Cuáles son los productos disponibles en la categoría de electrónicos?" y verificar que la respuesta sea correcta y completa.

Las pruebas de competencia son una forma efectiva de validar la utilidad práctica de una ontología [5]

### **2.3.6 Cobertura de Ontologías**

La cobertura de ontologías es un aspecto crucial en la validación y aseguramiento de la calidad de las ontologías. La cobertura de ontologías se refiere a la exhaustividad con la que una ontología ha sido probada para garantizar que todas sus clases, propiedades y restricciones están correctamente definidas y funcionan como se espera. Este proceso es vital para asegurar que la ontología representa fielmente el dominio de conocimiento que pretende modelar y que puede ser utilizada de manera efectiva en aplicaciones prácticas.

La cobertura de ontologías puede definirse como una medida del grado en que los elementos y relaciones de una ontología han sido evaluados y verificados mediante pruebas sistemáticas.

La cobertura de propiedades se centra en asegurar que todas las propiedades, tanto de datos como de objetos, funcionen correctamente y enlacen las clases apropiadamente. Esto es crucial para mantener la coherencia de las relaciones y atributos dentro de la ontología, permitiendo una navegación y consulta precisas.

La cobertura de restricciones valida que las restricciones definidas en la ontología son correctas y efectivas. Esto incluye la verificación de restricciones cardinales, de dominio y de rango, garantizando que las reglas y limitaciones impuestas por la ontología se respeten y se apliquen correctamente.

Por último, la cobertura de instancias se encarga de comprobar que las instancias de las clases están correctamente clasificadas y cumplen con las restricciones impuestas por la ontología. Esto asegura que los datos específicos adheridos a las clases cumplen con las definiciones y restricciones establecidas, manteniendo la integridad y validez de la información representada en la ontología.

La cobertura de ontologías garantiza que la ontología es completa y precisa, representando adecuadamente el conocimiento del dominio. Esto es fundamental para asegurar que las aplicaciones que dependen de la ontología funcionen correctamente y proporcionen resultados fiables.

Una ontología bien cubierta facilita la interoperabilidad entre diferentes sistemas y aplicaciones. Al garantizar que todas las clases y propiedades están

bien definidas y probadas, se asegura que los datos pueden ser interpretados y utilizados de manera coherente en diferentes contextos.

La cobertura exhaustiva de una ontología ayuda a identificar errores y omisiones que podrían causar problemas en su uso práctico. Esto incluye errores semánticos, como relaciones mal definidas, y errores sintácticos, como restricciones mal formuladas.

Las ontologías con alta cobertura tienden a ser más eficientes, ya que los errores e inconsistencias se han resuelto, lo que reduce la carga de procesamiento y mejora el rendimiento de las aplicaciones que las utilizan.

Una ontología bien cubierta es más fácil de mantener y evolucionar. Al tener una comprensión clara de cómo se utilizan todas las partes de la ontología, los desarrolladores pueden hacer cambios y mejoras con confianza, sabiendo que no introducirán nuevos errores.

## **3 Software**

En este apartado se tratará de explicar lo que es un software , como se prueba, que niveles de cobertura tienen y las diferencias existentes entre cómo se aplican test y cobertura al software frente a las ontologías

Las pruebas de software son un conjunto de actividades diseñadas para evaluar la calidad de un software y asegurar que cumple con los requisitos especificados.

### **3.1 Pruebas Unitarias**

Las pruebas unitarias se centran en verificar el correcto funcionamiento de las unidades individuales del código, como funciones, métodos o clases. Estas pruebas son esenciales para detectar y corregir errores en una etapa temprana del desarrollo.

Ejemplo de prueba unitaria: Verificar que una función que suma dos números devuelve el resultado correcto.

Las pruebas unitarias permiten identificar y solucionar errores en componentes específicos, mejorando la calidad del software desde sus primeras fases de desarrollo [6]

### **3.2 Pruebas de Integración**

Las pruebas de integración tienen como objetivo verificar la interacción entre diferentes módulos o componentes del software. Estas pruebas aseguran que los módulos funcionan correctamente cuando se combinan.

Ejemplo de prueba de integración: Comprobar que un módulo de autenticación interactúa correctamente con un módulo de base de datos.

Las pruebas de integración son cruciales para garantizar que los diferentes componentes del sistema colaboran de manera efectiva y sin conflictos [7]

### **3.3 Pruebas de Sistema**

Las pruebas de sistema evalúan el sistema completo para verificar que cumple con los requisitos funcionales y no funcionales. Estas pruebas abarcan desde la interfaz de usuario hasta la base de datos y otros componentes integrados.

Ejemplo de prueba de sistema: Probar una aplicación web completa, desde la navegación por las páginas hasta la gestión de datos en la base de datos.

Las pruebas de sistema proporcionan una visión integral de la calidad y el comportamiento del software en un entorno realista [8]

### **3.4 Pruebas de Aceptación**

Las pruebas de aceptación son realizadas para determinar si un sistema cumple con los criterios de aceptación y está listo para ser utilizado por el usuario final. Estas pruebas se basan en escenarios de uso reales y son generalmente realizadas por los clientes o usuarios.

Ejemplo de prueba de aceptación: Validar que todas las funcionalidades de una aplicación de e-commerce funcionan como se espera antes del lanzamiento.

Las pruebas de aceptación son el último paso antes de la implementación, asegurando que el software cumple con las expectativas del cliente y es adecuado para su propósito [9]

### **3.5 Pruebas de Regresión**

A menudo errores son descubiertos o cambios son actualizados en las versiones posteriores de un software es aquí donde bajo mi opinión encontramos las pruebas más importantes a realizar.

Las pruebas de regresión se realizan para verificar que las nuevas modificaciones en el software no han introducido errores en las funcionalidades existentes. Estas pruebas son cruciales cada vez que se realiza un cambio en el código.

Su importancia radica en no solo probar las nuevas funcionalidades sino realizar unas pruebas de manera exhaustiva que incluye las anteriormente mencionadas con el objetivo de garantizar que la introducción del nuevo código no entra en conflicto con todo lo anterior.

Ejemplo de prueba de regresión: Después de corregir un error en la funcionalidad de búsqueda, verificar que todas las otras funcionalidades relacionadas siguen funcionando correctamente.

Las pruebas de regresión son esenciales para mantener la estabilidad del software a lo largo de su ciclo de vida, especialmente en entornos de desarrollo ágil [10]

### **3.6 Otras Pruebas**

Las pruebas de rendimiento evalúan cómo se comporta el sistema bajo diferentes cargas de trabajo. Esto incluye pruebas de carga, pruebas de estrés y pruebas de volumen para asegurar que el sistema pueda manejar la cantidad esperada de usuarios y datos.

Las pruebas de seguridad se centran en identificar vulnerabilidades y asegurar que el sistema protege adecuadamente los datos y mantiene la integridad y confidencialidad de la información

Las pruebas de usabilidad evalúan la facilidad de uso de la interfaz del software, asegurando que los usuarios puedan interactuar con el sistema de manera efectiva y eficiente

### **3.7 Cobertura Software**

La cobertura de software se refiere a la medida en que el código de un programa ha sido ejecutado por un conjunto de pruebas. Evaluar la cobertura de software es fundamental para asegurar que se han probado suficientes caminos del código, lo que aumenta la confianza en su calidad y funcionamiento.

#### **3.7.1 Cobertura de Declaraciones**

La cobertura de declaraciones (statement coverage) mide el porcentaje de declaraciones ejecutadas en el código durante las pruebas. Asegurar una alta cobertura de declaraciones ayuda a detectar errores en líneas de código individuales.

De una forma equívoca se podría entender que una declaración es una línea de código , a pesar de que no es así y que en una línea puede haber varias declaraciones es cierto que estos dos conceptos suelen asociarse como uno solo

La cobertura de declaraciones es un indicador básico de qué tan bien se ha probado el código, pero no garantiza que todas las posibles condiciones se hayan evaluado [11]

#### **3.7.2 Cobertura de Decisiones**

La cobertura de decisiones evalúa el porcentaje de decisiones (if, switch, loops, etc.) en el código que han sido probadas en todas sus ramas posibles. Esto asegura que cada camino de decisión haya sido ejecutado al menos una vez.

En el caso de un “IF” se trataría de pasar tanto por la rama de verdad como la falsa.

La cobertura de decisiones es más rigurosa que la cobertura de declaraciones, ya que verifica la ejecución de todos los caminos lógicos en el código [12]

### **3.7.3 Cobertura de Condiciones y Decisiones Modificadas (MC/DC)**

La cobertura MC/DC asegura que cada condición dentro de una decisión afecte de manera independiente el resultado de la decisión. Este tipo de cobertura es especialmente importante en sistemas críticos donde se requiere un alto nivel de fiabilidad por ejemplo en sistemas ferroviarios

Supongamos esta condición

“if (A or B) and C”.

Caso 1: A = True, B = False, C = True (La decisión es verdadera)

Caso 2: A = False, B = True, C = True (La decisión es verdadera)

Caso 3: A = True, B = True, C = False (La decisión es falsa)

Caso 4: A = False, B = False, C = True (La decisión es falsa)

Caso 1 y Caso 2: Demuestran que A y B pueden ser falsos y la decisión aún puede ser verdadera si el otro es verdadero y C es verdadero.

Caso 3 y Caso 4: Demuestran que C afecta la decisión. Si C es falso, toda la decisión es falsa, independientemente de A y B

MC/DC es un estándar exigente utilizado en la industria aeroespacial y de automoción debido a su capacidad para asegurar una alta fiabilidad del software [14]

### **3.7.4 Cobertura de Caminos**

La cobertura de caminos intenta asegurar que todos los caminos posibles a través del código han sido ejecutados al menos una vez. Esto incluye todas las combinaciones posibles de declaraciones y decisiones.

Esta cobertura no siempre es del todo necesaria aunque abre la puerta a los llamados caminos críticos , aquellos en los que justo se entra o sale de cierta condición o empieza determinada declaración.

Por ejemplo en un bucle de 2000 iteraciones no haría falta probarlas todas pero sí sería conveniente probar la entrada, la salida por condición externa (caso de while) o la salida por fin de iteración (caso de for).

La cobertura de caminos es la más exhaustiva de todas las métricas de cobertura, pero también la más difícil de alcanzar debido a la combinatoria explosiva de caminos posibles[15]

### **3.8 Ontología Vs Software**

En este punto se trata de explicar la gran diferencia existente entre las ontologías (representación de información) y el software (código que define comportamientos específicos a través de instrucciones ejecutables).

Un software trata de funcionalidad y requisitos, es decir mediante la elaboración de un código ejecutable se realiza una funcionalidad como podría ser una aplicación “calculadora” por ejemplo.

En este punto las pruebas realizadas probarían no solo la funcionalidad (que las operaciones de la calculadora sean exactas) si no también los requisitos añadidos (por ejemplo que solo de resultados con máximo 2 decimales).

Para este caso la cobertura de código se basaría en analizar cuantas funcionalidades o requisitos son probados en los tests (esto puede variar dependiendo de la cobertura, líneas, funciones, condiciones...)

La ontología se basa en un significado semántico dicho de otra forma que los datos sean coherentes con su estructura, de esta manera en una ontología en la que se defina una propiedad de los datos edad restringida a un número entero para la clase persona no sería coherente decir por ejemplo que la edad de Juan es “Hola Mundo”.

Es aquí donde radica la importancia de los tests y la cobertura en la ontología, los tests para asegurar que cierta restricción ha sido impuesta sobre los datos (edad solo puede ser un número entero) y la cobertura donde se podría analizar todas aquellas situaciones que ya han sido probadas y verificadas en los tests (edad es un número entero y además en una propiedad de los datos de la clase persona).

## **4 Estado Actual**

A medida que las ontologías se vuelven cada vez más críticas para aplicaciones en la Web Semántica, la biomedicina, la inteligencia artificial y otros dominios, la necesidad de técnicas y herramientas robustas para garantizar su calidad y eficacia se ha vuelto más evidente. A continuación, se presenta un análisis del estado actual de la cobertura de ontologías, destacando los avances, las herramientas disponibles y los desafíos persistentes.

### **4.1 Evaluación Basada en Métricas**

La evaluación de ontologías es crucial para asegurar su calidad, utilidad y eficiencia. Una de las metodologías más utilizadas para esta evaluación es la Evaluación Basada en Métricas. Este enfoque utiliza diversas métricas cuantitativas para medir diferentes aspectos de una ontología.

#### **4.1.1 Tamaño de la Ontología**

Esta métrica incluye el conteo de elementos como clases, propiedades y axiomas dentro de la ontología.

Clases: Número total de clases definidas.

Propiedades: Número total de propiedades (objetos y datos).

Axiomas: Número total de axiomas.

El tamaño de una ontología proporciona una visión general de su complejidad y alcance [17].

#### **4.1.2 Profundidad y Anchura de la Jerarquía de Clases**

Profundidad: La distancia desde la clase más alta (superclase) hasta la clase más baja (subclase).

Anchura: Número de subclases directas de una clase.

La profundidad y la anchura de la jerarquía de clases ayudan a entender la estructura y organización de la ontología [18]

### **4.1.3 Relaciones entre Clases y Propiedades**

Relaciones de herencia: Número de relaciones de tipo subclase (is-a).

Relaciones de dominio y rango: Definiciones de dominio y rango de las propiedades.

Analizar las relaciones entre clases y propiedades es fundamental para asegurar que la ontología refleja correctamente las dependencias del dominio [19].

### **4.1.4 Cobertura del Dominio**

Esta métrica evalúa si la ontología abarca todos los conceptos y relaciones relevantes para el dominio específico que pretende modelar.

Una ontología debe ser suficientemente amplia para cubrir todos los aspectos críticos del dominio de interés [20].

### **4.1.5 Consistencia**

Verifica si la ontología está libre de contradicciones internas. Esto se puede hacer utilizando razonadores automáticos que validan la coherencia lógica de la ontología.

"La consistencia es un aspecto crítico, ya que una ontología inconsistente puede llevar a inferencias incorrectas" [21].

### **4.1.6 Concisión**

Evalúa la eliminación de redundancias dentro de la ontología, asegurando que no hay definiciones duplicadas o superfluas.

La concisión mejora la eficiencia y la manejabilidad de la ontología, facilitando su mantenimiento y evolución [22].

#### **4.1.7 Comprensibilidad**

Mide qué tan fácil es para los usuarios entender la ontología. Esto puede incluir la claridad de la nomenclatura, la documentación disponible y el uso de comentarios explicativos.

Una ontología comprensible es esencial para su adopción y correcta utilización por parte de los usuarios finales [23].

### **4.2 Evaluación con Razonadores**

Los razonadores son herramientas esenciales en la evaluación de ontologías, ya que permiten automatizar la verificación de la lógica y la inferencia de nueva información a partir de las definiciones existentes. A continuación, se describe en detalle el funcionamiento de los razonadores, sus capacidades y su importancia en el desarrollo y mantenimiento de ontologías.

Los razonadores verifican que una ontología no contenga contradicciones lógicas. Por ejemplo, aseguran que un concepto no pueda pertenecer simultáneamente a dos clases distintas.

A partir de las definiciones y axiomas existentes, los razonadores pueden inferir nuevas relaciones y clasificaciones que no están explícitamente presentes en la ontología.

Por ejemplo, si la ontología define que "todos los perros son mamíferos" y "Fido es un perro", un razonador puede inferir que "Fido es un mamífero".

Los razonadores identifican problemas en las restricciones de las propiedades, como dominios y rangos mal definidos, o cardinalidades que no se cumplen.

#### **4.2.1 Verificación de Consistencia**

Los razonadores comprueban si la ontología es consistente, es decir, si no contiene contradicciones lógicas. Esto es esencial para asegurar que la ontología pueda ser utilizada de manera confiable en aplicaciones prácticas.

La consistencia es un requisito fundamental para cualquier ontología que se utilice en aplicaciones críticas, ya que garantiza que no existan contradicciones internas. [24].

#### **4.2.2 Clasificación de la Jerarquía**

Los razonadores pueden clasificar automáticamente las clases y propiedades dentro de la ontología, organizándolas en una jerarquía coherente basada en sus definiciones y axiomas.

La capacidad de los razonadores para clasificar jerarquías de manera eficiente permite a los desarrolladores de ontologías asegurar que su estructura es lógica y bien organizada [25]

#### **4.2.3 Inferencia de Conocimientos**

Los razonadores pueden inferir nuevos hechos y relaciones que no están explícitamente definidos en la ontología pero que se derivan lógicamente de los axiomas existentes.

La inferencia de nuevos conocimientos es una de las ventajas más poderosas de utilizar razonadores, permitiendo descubrir relaciones y hechos implícitos [26]

#### **4.2.4 Detección de Errores y Anomalías**

Mediante el uso de razonadores, es posible detectar errores y anomalías en la ontología, como definiciones redundantes o relaciones innecesarias.

Los razonadores son herramientas efectivas para identificar errores sutiles que pueden pasar desapercibidos durante la construcción manual de una ontología. [27].

## **5 Diseño del Método de Análisis de Cobertura**

En este apartado del trabajo explicaremos el diseño realizado para el método de análisis de cobertura.

### **5.1 Definición de Objetivos**

- Identificar y enumerar todas las clases definidas en la ontología, proporcionando una visión completa de la estructura de la misma.
- Enumerar todas las propiedades (tanto de objeto como de datos) utilizadas en la ontología, detallando sus dominios y rangos.
- Identificar y listar todas las relaciones de subclase (jerarquías) dentro de la ontología, mostrando la estructura jerárquica completa.
- Extraer y listar todas las tripletas del tipo clase-propiedad-clase, que describen cómo las clases están relacionadas a través de propiedades.
- Ofrecer una forma efectiva y visualmente intuitiva de representar los resultados del análisis, facilitando la comprensión y el análisis por parte de los usuarios.
- Proveer herramientas que permitan a los usuarios identificar rápidamente los elementos más relevantes y críticos de la ontología.

## **5.2 Definición de Requisitos**

### **5.2.1 Requisitos Funcionales**

- Listar Clases: Herramientas para identificar y enumerar todas las clases definidas en la ontología.
- Listar Propiedades: Herramientas para enumerar todas las propiedades, detallando sus dominios y rangos.
- Listar Subclases: Herramientas para identificar y listar todas las relaciones de subclase. Para este caso incluir también aquellas tripletas de nodo vacío con solo un nivel de indirección
- Listar Tripletas: Herramientas para extraer y listar todas las tripletas del tipo clase-propiedad-clase. (que previamente ya hemos listado en clases y propiedades)
- Visualización de Resultados: Herramientas para generar visualizaciones claras e intuitivas de los resultados.

### **5.2.2 Requisitos No Funcionales**

- Interfaz de usuario amigable y fácil de usar.
- Capacidad de manejar ontologías grandes y proporcionar resultados rápidamente.
- Posibilidad de extender el método para cubrir nuevos requisitos y dominios de conocimiento.
- Compatibilidad con otros sistemas y ontologías a través de estándares reconocidos.
- Capacidad de interactuar con las visualizaciones para explorar los datos de manera dinámica.

## 5.3 Arquitectura Propuesta

### 5.3.1 Arquitectura del Código

1. Importaciones.
  - a. Bibliotecas: El código importa Graph y URIRef de rdflib, y RDF, RDFS de rdflib.namespace.
  - b. OS: También importa el módulo os para manejar rutas de archivos
2. Funciones Utilitarias.
  - a. obtener\_prefijo\_para\_uri(prefijos, uri): Esta función busca el prefijo correspondiente a una URI dada de un diccionario de prefijos
  - b. obtener\_prefijos\_desde\_turtle(ruta\_archivo, g): Extrae y retorna los prefijos de un archivo Turtle cargado en un grafo RDF.
  - c. obtener\_recurso\_desde\_uri(uri): Extrae el recurso (el último fragmento) de una URI.
  - d. lanzar\_test(): Mediante la API proporcionada por themis lanza la batería de test en formato json retornando los resultados.
3. Funciones de Búsqueda de Tripletas
  - a. buscar\_tripleta\_type(g, sc, pc, oc): Busca la tripleta (sc,pc,oc) del tipo *type* en el grafo RDF, diferenciando entre clases e instancia
  - b. buscar\_tripleta\_subclase(g, sc, pc, oc): Busca tripletas que representan relaciones de subclase (subClassOf) en el grafo RDF.
  - c. buscar\_tripleta\_subclase\_herencia(g,sc,rc,pc,oc): Esta función es para buscar tripletas en un caso especial en el que la clase sea subclase de un *nodo vacío*
  - d. buscar\_tripleta\_range\_domain(g,sc,pc,oc): Esta función busca la tripleta de rango o dominio .
  - e. buscar\_tripleta\_cpc(g,sc,pc,oc): Esta función busca la tripleta para la estructura de clase-propiedad-clase.
4. Variables Globales:

Utilización de una gran cantidad de listas set para listar los distintos elementos de la ontología , separados tanto por clase , propiedad como si están cubiertos o no.

### 5.3.2 Componentes Principales

El código abarca varias funciones esenciales para el procesamiento y análisis de ontologías. Las funciones de procesamiento son fundamentales para cargar y manejar ontologías, extrayendo los prefijos y recursos necesarios, y buscando relaciones específicas dentro del grafo RDF. Estas operaciones permiten una manipulación efectiva de los datos de la ontología, preparando el terreno para el análisis de cobertura.

Las funciones dedicadas al manejo de prefijos y URIs son cruciales para la interpretación correcta de las URIs dentro de la ontología. Estas funciones facilitan la identificación y extracción de elementos clave, permitiendo un manejo más eficiente de las diversas partes de la ontología. Esto es especialmente útil cuando se trabaja con grandes conjuntos de datos RDF que requieren una gestión precisa de los prefijos y las URIs.

En cuanto a la cobertura de ontologías, las funciones “buscar\_tripleta...” están diseñadas específicamente para verificar la cobertura de la ontología en términos de instancias y jerarquías de clases. Estas funciones buscan asegurarse de que todas las instancias y relaciones jerárquicas están adecuadamente representadas y cubiertas, lo que es esencial para garantizar la integridad y la exhaustividad de la ontología.

Finalmente, una vez que todos los elementos han sido listados y verificados, se llama a la función que genera los resultados de la cobertura. Esta función es responsable de compilar y presentar los datos de cobertura de manera clara y comprensible, facilitando la interpretación de los resultados y la identificación de áreas que puedan necesitar mejoras. La generación de resultados es el paso final en el proceso de análisis de cobertura, proporcionando una visión completa y detallada del estado de la ontología.

## 5.4 Implementación y desarrollo del método

### Paso 1: Configuración Inicial

La primera etapa de la implementación del método consiste en configurar el entorno necesario para trabajar con la ontología RDF. Esto incluye la importación de bibliotecas esenciales como `rdflib`, que permite la manipulación y consulta de gráficos RDF. En esta fase, también se definen varios conjuntos que almacenarán diferentes elementos de la ontología, tales como clases, propiedades, instancias, datatypes, y subclases, entre otros. Estos conjuntos serán utilizados para realizar un seguimiento tanto de los elementos presentes en la ontología como de aquellos que han sido cubiertos por los tests.

### Paso 2: Definición de Funciones de Búsqueda de Tripletas

El siguiente paso es definir funciones que busquen tripletas específicas en el gráfico RDF. Cada función se encarga de verificar la existencia de determinadas relaciones en la ontología. Por ejemplo, se define una función que busca tripletas del tipo **sujeto-predicado-objeto**, asegurando que ciertas instancias pertenecen a ciertas clases. Estas funciones actualizan los conjuntos de cobertura, registrando qué elementos han sido encontrados y cuáles no, permitiendo una evaluación detallada de la cobertura de la ontología.

### **Paso 3: Ejecución de Tests y Formateo de Resultados**

Una vez definidas las funciones de búsqueda, se procede a la ejecución de una serie de tests sobre la ontología. Estos tests están diseñados para verificar la presencia y la correcta estructuración de las tripletas RDF. Los resultados obtenidos de estos tests son luego formateados para facilitar su interpretación. Esta etapa es crucial para identificar tanto los elementos correctamente cubiertos como aquellos que no cumplen con los criterios establecidos.

### **Paso 4: Evaluación de Resultados y Cobertura**

En esta etapa, los resultados de los tests son evaluados y clasificados en dos categorías principales: coberturas y no coberturas. Esta clasificación permite identificar qué elementos de la ontología están correctamente definidos y cuáles requieren atención adicional. Se verifica la presencia de clases, propiedades, instancias, subclases y datatypes, y se registra su cobertura. Esta evaluación detallada proporciona una visión clara de las áreas de la ontología que están correctamente estructuradas y aquellas que necesitan mejoras.

### **Paso 5: Generación de Resultados en HTML**

Para facilitar la visualización y el análisis de los resultados, se genera un conjunto de archivos HTML. Estos archivos presentan de manera organizada y visualmente atractiva los resultados de la cobertura de la ontología. Cada archivo HTML contiene listas de los elementos cubiertos y no cubiertos, permitiendo a los usuarios identificar rápidamente las áreas de interés. La generación de estos archivos HTML es un paso crucial para la presentación clara y accesible de los resultados del análisis.

### **Paso 6: Vinculación de Clases con sus Tests**

Finalmente, para proporcionar una visión más detallada y específica, se generan archivos HTML adicionales que listan los tests relacionados con cada clase. Estos archivos están vinculados desde el archivo principal de clases, permitiendo a los usuarios navegar fácilmente entre las clases y sus respectivos tests. Esta vinculación es esencial para proporcionar un acceso detallado y contextual a la información, facilitando un análisis más profundo de la cobertura y la estructura de la ontología.

## 6 Desarrollo y Evaluación del Método

En este punto del trabajo se explica cómo hemos desarrollado y evaluado el método en distintos casos para distintas ontologías.

### 6.1 Creación de Tests en Themis.

Tal y como hemos visto en puntos anteriores, existen ya distintas herramientas para evaluar o “probar” ontologías, las cuales necesitaremos usar dado que el trabajo se centra en la cobertura y no en el testeo. Para este caso, usaremos Themis, que se encuentra en <https://themis.linkeddata.es/>.

Themis es una herramienta diseñada por Alba Fernández Izquierdo. Esta herramienta proporciona un entorno robusto y flexible para la validación de ontologías mediante tests. A través de Themis, los usuarios pueden cargar cualquier ontología de su elección y someterla a una serie de tests predefinidos o personalizados que permiten evaluar diversos aspectos de su estructura y contenido. La validación se realiza siguiendo un conjunto de reglas y criterios específicos que aseguran la calidad y la coherencia de la ontología evaluada.

Themis permite a los usuarios definir y ejecutar tests sobre sus ontologías. Estos tests pueden cubrir una amplia gama de aspectos, desde la correcta definición de clases y propiedades hasta la verificación de instancias y subclase

Los tests pueden ser tanto predefinidos como personalizados, ofreciendo flexibilidad para adaptarse a las necesidades específicas de cada proyecto.

La herramienta proporciona una interfaz gráfica intuitiva que facilita la carga de ontologías y la ejecución de tests. Los resultados de los tests se presentan de manera clara y estructurada, permitiendo una fácil interpretación.

Además de su interfaz gráfica, Themis ofrece una API que permite la automatización del proceso de validación. Esta API es particularmente útil para integraciones en flujos de trabajo automatizados o para su uso en scripts y programas externos

Para nuestro proyecto, utilizaremos Themis para lanzar los tests necesarios sobre la ontología y obtener los resultados, los cuales serán luego utilizados para evaluar la cobertura.

Para ello únicamente precisamos de rellenar el json con el llamar a la API de themis y obtener los resultados.

## 6.2 Ejecución de Tests

Como se dijo previamente utilizaremos la api de themis para la ejecución de los test en concreto

<https://themis.linkeddata.es/swagger-ui/index.html#/Themis%20evaluator/getResults>

que recibe un json con los test y devuelve los resultados en json también

```
{
  "got": "{\\\"core\\\": [{\\\"Term\\\": \\\"Actuator\\\", \\\"URI\\\": \\\"https://saref.etsi.org/core/Actuator\\\"}]",
  "ontologies": [
    "https://saref.etsi.org/core/"
  ],
  "ontologiesCode": [],
  "testfile": "",
  "documentationFile": "",
  "tests": [
    "Actuator type Class",
    "Actuation type Class",
    "Device subclassOf FeatureOfInterest"
  ],
  "format": "json"
}
```

La estructura del json como se observa en la imagen es la siguiente

**got (glosario de términos)** , se deben añadir todos aquellos términos de los que se pretenda hacer un test.

**ontologies**, URI a la ontología de la que se va a testear

**tests**, siguiendo el formato de la imagen se crean tripletas como test.

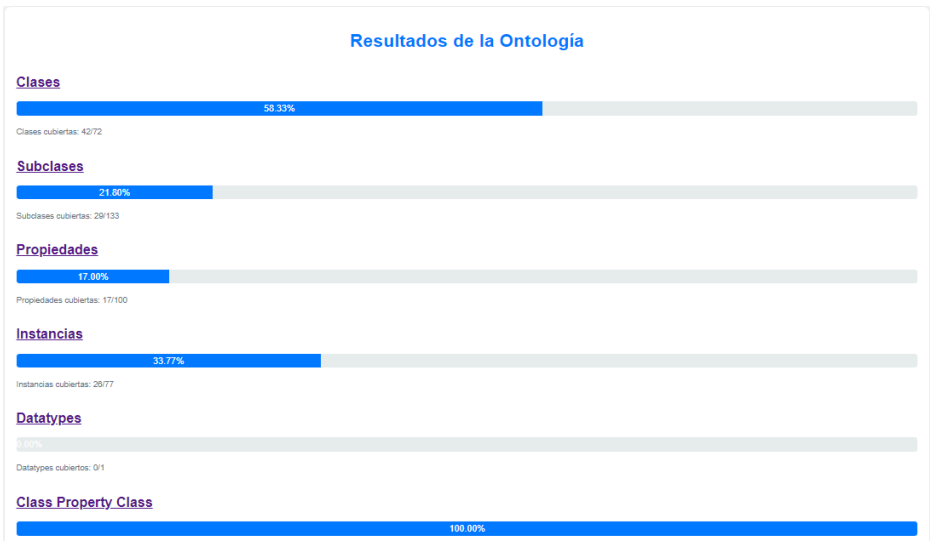
En cuanto a los resultados los obtenemos de la siguiente manera en json

```
Response body
{
  {
    "Test": "Sensor type Class",
    "Results": [
      {
        "Ontology": "http://iot.linkeddata.es/def/core",
        "Result": "Passed"
      }
    ]
  },
  {
    "Test": "Sensor subclassOf Device",
    "Results": [
      {
        "Undefined": [
          "Device"
        ],
        "Ontology": "http://iot.linkeddata.es/def/core",
        "Result": "Undefined"
      }
    ]
  }
}
```

## 6.3 Casos de Estudio y Validación del Método

Para validar el método, hemos utilizado diversas ontologías, entre ellas saref4watr. Esta ontología, disponible en el repositorio de ETSI (European Telecommunications Standards Institute), proporciona un modelo común para representar dispositivos y servicios relacionados con la gestión del agua. El enlace a la ontología es el siguiente: [saref4watr](#).

Utilizando saref4watr, hemos llevado a cabo una serie de pruebas que, aunque originalmente en formato .csv, se adaptaron para ser procesadas en formato JSON. Estos tests nos permitieron evaluar la precisión y eficiencia de nuestro método en un entorno realista.



Podemos observar como los test cubren gran parte de las clases , dentro de clases encontramos,

| Clases                   |  |
|--------------------------|--|
| Manhole                  |  |
| Aquifer                  |  |
| SourceAsset              |  |
| WaterMeterProperty       |  |
| StorageAsset             |  |
| Interval                 |  |
| TemporalEntity           |  |
| Lagoon                   |  |
| Person                   |  |
| Point                    |  |
| HydroelectricPowerPlant  |  |
| River                    |  |
| Intake                   |  |
| Tariff                   |  |
| MonitoringInfrastructure |  |
| WaterInfrastructure      |  |

Para evaluar la cobertura de los tests realizados utilizando la ontología saref4watr, hemos categorizado las clases según hayan sido cubiertas por algún test o no. Las clases que no han sido cubiertas por un test están marcadas en rojo, mientras que aquellas que sí lo han sido están en verde. Esto proporciona una visión clara de la profundidad y amplitud de los tests y su cobertura en términos de las clases de la ontología.

Es importante destacar que la cobertura óptima no se basa en tener un test por clase individualmente, sino en probar aspectos importantes que, debido a la redundancia, abarcan múltiples clases y propiedades. Por ejemplo, si tenemos un test del tipo "ClaseA subclassof ClaseB" y este test es verdadero, no solo se cubre la tripleta de la subclase, sino que también se da por sentado que ClaseA y ClaseB son instancias de sus respectivas clases. De esta manera, se logra una cobertura más amplia y eficiente.

Por ejemplo usando la clase "Device" podemos desglosarla de la siguiente manera.

|                 |           |
|-----------------|-----------|
| StepDownCommand | Ver tests |
| Device          | Ver tests |
| SensingFunction | Ver tests |
| State           | Ver tests |

|                                            |  |
|--------------------------------------------|--|
| Tests para Device                          |  |
| Device subClassOf FeatureOfInterest Passed |  |
| Device type Class Passed                   |  |

Comprobamos cómo "Device" aparece cubierto por dos test , además también se indica si el test fue pasado o no , tambien podria ser "undefined" en caso de no haber encontrado los términos a los que se refiere

## 7 Conclusiones

Este trabajo, aunque simple, representa una primera toma de contacto en el intento de elaborar una forma de visualizar la cobertura de una ontología. Hemos desarrollado y aplicado un método que permite identificar claramente las áreas de la ontología que están cubiertas por los tests y aquellas que requieren mayor atención. Al marcar en rojo los elementos no cubiertos y en verde los que sí lo están, hemos proporcionado una herramienta visual efectiva para evaluar la cobertura individual de cada clase y las interrelaciones entre ellas.

A través de este enfoque, hemos demostrado que es posible obtener una visión detallada y precisa de la efectividad y profundidad de los tests aplicados. Este método no solo facilita la identificación de áreas no cubiertas, sino que también ayuda a entender cómo las relaciones y redundancias entre clases pueden influir en la cobertura total de la ontología. La visualización de la cobertura es un paso fundamental para asegurar que las ontologías sean robustas y completas. Al proporcionar una representación gráfica de la cobertura, los desarrolladores e investigadores pueden identificar rápidamente las áreas que necesitan más tests y ajustar sus estrategias de validación en consecuencia.

Sin embargo, es importante notar que este trabajo es solo una primera toma de contacto. En el futuro, otros investigadores podrán construir sobre esta base para realizar trabajos más completos, profundos y efectivos. Podrán desarrollar herramientas de automatización más avanzadas, ampliar la cobertura de tests a todas las clases y propiedades de la ontología, y crear sistemas de visualización interactivos que permitan una evaluación más dinámica y detallada. Además, la validación del método en escenarios del mundo real y la integración de múltiples ontologías pueden proporcionar una comprensión más holística y práctica de la cobertura de ontologías.

A pesar de las limitaciones y desafíos enfrentados, como la complejidad de las ontologías y la necesidad de adaptar formatos de tests, este trabajo sienta las bases para futuras investigaciones y mejoras. En resumen, este estudio inicial ha mostrado el potencial de una herramienta visual para evaluar la cobertura de una ontología, proporcionando una base sólida para que futuras investigaciones desarrollen métodos más robustos y efectivos en esta área crucial.

## **8 Trabajo Futuro**

### **8.1 Resumen de Resultados**

El análisis de cobertura de ontologías mediante nuestro método ha proporcionado una visión detallada y precisa de la efectividad y profundidad de los tests aplicados. Al marcar en rojo los elementos no cubiertos y en verde los que sí lo están, hemos podido identificar claramente las áreas de la ontología que requieren mayor atención. Este enfoque ha demostrado ser eficaz para evaluar no solo la cobertura individual de cada clase, sino también las interrelaciones y redundancias que existen entre ellas, permitiendo una validación más completa y eficiente.

### **8.2 Limitaciones y Desafíos Enfrentados**

La complejidad de las ontologías se manifiesta en la estructura y la interconexión intrincada de clases y propiedades, lo que dificulta asegurar una cobertura completa sin redundancia excesiva. La adaptación de formatos de los tests, por ejemplo, de .csv a JSON, añade otra capa de dificultad, ya que es una tarea laboriosa y propensa a errores. Esta adaptación puede consumir tiempo valioso y generar inconsistencias que afecten la calidad del análisis. Además, la automatización insuficiente representa un desafío significativo. La ausencia de herramientas completamente automatizadas para la generación y evaluación de tests limita la eficiencia del proceso, haciendo que la tarea sea más manual e intensiva en recursos, lo cual puede ralentizar el progreso y aumentar la posibilidad de errores humanos.

### **8.3 Propuestas para Futuras Investigaciones**

El desarrollo de herramientas de automatización es una propuesta clave para avanzar en este campo. El objetivo principal es crear herramientas avanzadas que automaticen el proceso de generación, conversión y evaluación de tests. Esto permitiría reducir significativamente el esfuerzo manual y minimizar los errores asociados a la adaptación de formatos y ejecución de tests. Para implementar esta solución, se podrían desarrollar scripts y aplicaciones capaces de tomar archivos de tests en diversos formatos y convertirlos automáticamente a los formatos necesarios. Además, estas herramientas deberían poder ejecutar los tests de manera automatizada y reportar los resultados de forma clara y concisa, mejorando así la eficiencia y precisión del proceso de análisis de cobertura de ontologías.

Otra propuesta importante es la integración multi-ontológica. El objetivo aquí es explorar la integración de múltiples ontologías en el análisis para evaluar la interoperabilidad y la coherencia entre diferentes dominios y sistemas. Al lograr esto, se aumentaría la interoperabilidad y la consistencia de datos entre

distintos sistemas y dominios, lo que es esencial en un entorno donde la información debe ser compartida y comprendida entre diversas plataformas. La implementación de esta propuesta implicaría diseñar estudios de caso que incluyan múltiples ontologías, evaluando cómo el método de análisis de cobertura puede ser adaptado y aplicado para validar la consistencia y la interconexión entre ellas. Esto no solo enriquecería el análisis de cobertura, sino que también contribuiría a una mejor integración y uso de las ontologías en aplicaciones del mundo real.

## 9 Referencias

1. Smith, J. *"Una ontología mal formada puede llevar a errores de interpretación y fallos en los sistemas que la utilizan."*
2. Jones, R. *"El análisis semántico de una ontología es esencial para asegurar que las relaciones definidas son coherentes y no conflictivas."*
3. Taylor, A. *"La consistencia es un requisito mínimo para cualquier ontología que pretenda ser útil en aplicaciones prácticas."*
4. Brown, L. *"Una ontología debe ser suficientemente rica para representar todos los aspectos relevantes del dominio que modela."*
5. White, K. *"Las pruebas de competencia son una forma efectiva de validar la utilidad práctica de una ontología."*
6. Beck, K. *"Las pruebas unitarias permiten identificar y solucionar errores en componentes específicos, mejorando la calidad del software desde sus primeras fases de desarrollo."*
7. Myers, G. *"Las pruebas de integración son cruciales para garantizar que los diferentes componentes del sistema colaboran de manera efectiva y sin conflictos."*
8. Sommerville, I. *"Las pruebas de sistema proporcionan una visión integral de la calidad y el comportamiento del software en un entorno realista."*
9. Kaner, C. *"Las pruebas de aceptación son el último paso antes de la implementación, asegurando que el software cumple con las expectativas del cliente y es adecuado para su propósito."*
10. Crispin, L. *"Las pruebas de regresión son esenciales para mantener la estabilidad del software a lo largo de su ciclo de vida, especialmente en entornos de desarrollo ágil."*
11. Kaner, C. *"La cobertura de declaraciones es un indicador básico de qué tan bien se ha probado el código, pero no garantiza que todas las posibles condiciones se hayan evaluado."*
12. Beizer, B. *"La cobertura de decisiones es más rigurosa que la cobertura de declaraciones, ya que verifica la ejecución de todos los caminos lógicos en el código."*
13. Myers, G. *"La cobertura de condiciones proporciona una visión más detallada del comportamiento del código bajo diferentes condiciones lógicas."*
14. Chilenski, J.J. *"MC/DC es un estándar exigente utilizado en la industria aeroespacial y de automoción debido a su capacidad para asegurar una alta fiabilidad del software."*
15. McCabe, T.J. *"La cobertura de caminos es la más exhaustiva de todas las métricas de cobertura, pero también la más difícil de alcanzar debido a la combinatoria explosiva de caminos posibles."*
16. Beck, K. *"Una alta cobertura de código es indicativa de un buen esfuerzo de pruebas, pero no debe ser la única medida de la calidad del software."*

17. Gómez-Pérez, A., Fernández-López, M., & Corcho, O. (2004). *Ontological Engineering: With Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer.
18. Uschold, M., & Gruninger, M. (1996). *Ontologies: Principles, methods and applications*. *Knowledge Engineering Review*.
19. Noy, N. F., & McGuinness, D. L. (2001). *Ontology Development 101: A Guide to Creating Your First Ontology*. Stanford Knowledge Systems Laboratory.
20. Obrst, L. (2003). *Ontologies for semantically interoperable systems*. *Proceedings of the 12th international conference on Information and knowledge management*.
21. Horridge, M., & Bechhofer, S. (2004). *The OWL API: A Java API for Working with OWL Ontologies*. *Proceedings of the 5th International Conference on The Semantic Web*.
22. Rector, A. L., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., ... & Wroe, C. (2004). *OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns*. *Proceedings of the 5th International Conference on The Semantic Web*.
23. Arp, R., Smith, B., & Spear, A. D. (2015). *Building Ontologies with Basic Formal Ontology*. MIT Press.
24. Horrocks, I., Patel-Schneider, P. F., & van Harmelen, F. (2003). "From SHIQ and RDF to OWL: The Making of a Web Ontology Language". *Journal of Web Semantics*.
25. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., & Patel-Schneider, P. (2007). *"The Description Logic Handbook: Theory, Implementation, and Applications"*. Cambridge University Press.
26. Hitzler, P., Krötzsch, M., Rudolph, S., & Sure, Y. (2009). *"Semantic Web: Grundlagen"*. Springer.
27. Kalyanpur, A., Parsia, B., Sirin, E., & Hendler, J. (2006). "Debugging unsatisfiable classes in OWL ontologies". *Journal of Web Semantics*.

Este documento esta firmado por



|                               |                                                                                                                     |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------|
| <b>Firmante</b>               | CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES                                               |
| <b>Fecha/Hora</b>             | Mon Jun 03 18:37:47 CEST 2024                                                                                       |
| <b>Emisor del Certificado</b> | EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES |
| <b>Numero de Serie</b>        | 561                                                                                                                 |
| <b>Metodo</b>                 | urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)                                                       |