UNIVERSIDAD DE MÁLAGA

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

TESIS DOCTORAL

TECNOLOGÍAS INFORMÁTICAS

Semantics in Big Data Analytics

E.T.S.I. Informática
R.D. 99/2011

Autor

Antonio Benítez Hidalgo

Directores

Dra. María del Mar Roldán García
ITIS Software
Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga

Dr. Ismael Navas Delgado
ITIS Software
Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga

Enero 2024

AUTOR: Antonio Benítez Hidalgo

https://orcid.org/0000-0002-4396-8359

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD DE LA TESIS PRESENTADA PARA OBTENER EL TÍTULO DE DOCTOR

D./Dña ANTONIO BENÍTEZ HIDALGO
Estudiante del programa de doctorado TECNOLOGÍAS INFORMÁTICAS de la Universidad de Málaga, autor/a de la tesis, presentada para la obtención del título de doctor por la Universidad de Málaga, titulada: SEMANTICS IN BIG DATA ANALYTICS

Realizada bajo la tutorización de JOSÉ FRANCISCO ALDANA MONTES y dirección de MARÍA DEL MAR ROLDÁN GARCÍA E ISMAEL NAVAS DELGADO (si tuviera varios directores deberá hacer constar el nombre de todos)

DECLARO QUE:

La tesis presentada es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, conforme al ordenamiento jurídico vigente (Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia), modificado por la Ley 2/2019, de 1 de marzo.

Igualmente asumo, ante a la Universidad de Málaga y ante cualquier otra instancia, la responsabilidad que pudiera derivarse en caso de plagio de contenidos en la tesis presentada, conforme al ordenamiento jurídico vigente.

En Málaga, a 20 de NOVIEMBRE de 2023

Fdo.: ANTONIO BENÍTEZ HIDALGO Doctorando/a	Fdo.: JOSÉ FRANCISCO ALDANA MONTES Tutor/a
Fdo.: MARÍA DEL MAR ROLDÁN GARCÍA E ISMAEL NAVAS DELGADO Director/es de tesis	

Edificio Pabellón de Gobierno. Campus El Ejido. 29071
Tel.: 952 13 10 28 / 952 13 14 61 / 952 13 71 10
E-mail: doctorado@uma.es

DECLARACIÓN DE DIRECCIÓN Y ORIGINALIDAD DE LA TESIS PRESENTADA PARA OBTENER EL TÍTULO DE DOCTOR

Dña MARÍA DEL MAR ROLDÁN GARCÍA y D. ISMAEL NAVAS DELGADO, profesores doctores del Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga

DECLARAN QUE:

D. ANTONIO BENÍTEZ HIDALGO, estudiante del programa de doctorado TECNOLOGÍAS INFORMÁTICAS ha realizado bajo su dirección y tutorización la tesis presentada para la obtención del título de doctor por la Universidad de Málaga, titulada: SEMANTICS IN BIG DATA ANALYTICS.

La tesis presentada es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, conforme al ordenamiento jurídico vigente (Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia), modificado por la Ley 2/2019, de 1 de marzo. Así mismo, las publicaciones en coautoría que avalan dicha tesis no forman parte de otra tesis doctoral en la Universidad de Málaga ni en ninguna otra universidad.

En Málaga, a 20 de NOVIEMBRE de 2023

Fdo.: MARÍA DEL MAR ROLDÁN GARCÍA E ISMAEL NAVAS DELGADO
Director/es de tesis

Edificio Pabellón de Gobierno. Campus El Ejido. 29071
Tel.: 952 13 10 28 / 952 13 14 61 / 952 13 71 10
E-mail: doctorado@uma.es

DECLARACIÓN DE TUTORIZACIÓN Y ORIGINALIDAD DE LA TESIS PRESENTADA PARA OBTENER EL TÍTULO DE DOCTOR

D. JOSÉ FRANCISCO ALDANA MONTES, profesor doctor del Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga

DECLARA QUE:

D. ANTONIO BENÍTEZ HIDALGO, estudiante del programa de doctorado TECNOLOGÍAS INFORMÁTICAS ha realizado bajo su tutorización la tesis presentada para la obtención del título de doctor por la Universidad de Málaga, titulada: SEMANTICS IN BIG DATA ANALYTICS.

La tesis presentada es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, conforme al ordenamiento jurídico vigente (Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia), modificado por la Ley 2/2019, de 1 de marzo. Así mismo, las publicaciones en coautoría que avalan dicha tesis no forman parte de otra tesis doctoral en la Universidad de Málaga ni en ninguna otra universidad.

En Málaga, a 20 de NOVIEMBRE de 2023

Fdo.: JOSÉ FRANCISCO ALDANA MONTES
Tutor de tesis

Edificio Pabellón de Gobierno. Campus El Ejido. 29071
Tel.: 952 13 10 28 / 952 13 14 61 / 952 13 71 10
E-mail: doctorado@uma.es

Acknowledgements

First and foremost, I would like to express my profound gratitude to my tutor and Principal Investigator, José F. Aldana, and my supervisors, María del Mar and Ismael. Your commitment to excellence and ability to challenge my boundaries have been fundamental to my development as a researcher and individual. What this work represents is also a testament to your dedication and trust.

I also want to thank my research group family, Khaos, for sharing their knowledge and experiences with me and for the countless hours of discussion and teamwork.

My sincere appreciation to my friends and partner, who consistently remind me that I'm never alone in any challenge I face. Your unwavering support has been my strength.

To my beloved family: my parents, who ignited the spark of curiosity within me and setted me on this path; my siblings, Alejandro and Amanda, who have always stood by me unconditionally; and most profoundly, to my grandmothers, María and Charo. Every page of this work carries the weight of your love.

Lastly, to all who have been a part of this journey, whether directly or indirectly, I dedicate this work to you. Thank you for being a part of my story.

Table of Contents

Introducción

Los desafíos de procesamiento planteados por el Big Data provienen de la gran cantidad de datos, la diversidad de fuentes de información y la velocidad a la que se generan. Estos desafíos asociados con el Big Data se conceptualizan en varias dimensiones (las conocidas "V's"), y su avance ha facilitado numerosas soluciones a lo largo de toda la cadena de valor de los datos.

Las fuentes de conocimiento sobre un campo pueden ser de varios tipos. Se puede adquirir conocimiento de un área consultando a un experto o extrayéndolo automáticamente de un repositorio de datos, un corpus de documentos en diferentes formatos (XML, JSON, etc.), una página web, etc. Algunas técnicas facilitan esta extracción, como el aprendizaje de ontologías o la inferencia de esquemas (Buitelaar *et al.* 2005). Sin embargo, la representación de este conocimiento no está estandarizada, lo que hace desafiante su integración de diferentes fuentes.

En los últimos años, han surgido múltiples aproximaciones dirigidas a utilizar la semántica de dominio para el análisis de datos. Estas técnicas aprovechan la definición formal y su capacidad para inferir conocimientos y verificar la consistencia para guiar el proceso de análisis. La literatura actual incluye contribuciones que incorporan la semántica de dominio expresada a través de ontologías como parte de aplicaciones de minería de datos. En estas aplicaciones, la semántica se utiliza con diferentes objetivos: para reducir el espacio de búsqueda especificando restricciones, para filtrar resultados en la etapa de post-procesamiento, para anotar semánticamente los resultados del proceso de minería, para anotar diferentes clústeres, etc. Peis *et al.* 2009 describe una amplia gama de sistemas de recomendación semánticos. También podemos encontrar trabajos recientes que incluyen ontologías para guiar procesos de aprendizaje automático y aprendizaje profundo, como Pinto *et al.* 2015, que utiliza una ontología en el proceso de clasificación, aprovechando su capacidad para inferir inconsistencias entre conceptos. En Phan *et al.* 2015, se propone un modelo de aprendizaje profundo basado en ontologías para predecir el comportamiento humano.

De manera similar, surge una tendencia de investigación que se enfoca en diseñar ontologías para la anotación semántica de flujos de tareas de procesamiento y análisis de datos (flujos de trabajo), con el objetivo de incorporar conocimiento

en los procesos de análisis y descubrimiento de conocimiento en grandes conjuntos de datos. Entre los trabajos más recientes que utilizan ontologías OWL para modelar la entrada y salida de algoritmos involucrados en flujos de trabajo de minería de datos y descubrimiento de conocimiento para generar composiciones válidas, podemos encontrar KDDONTO (Diamantini *et al.* 2009), DMWF (Kietz *et al.* 2010), y KD Ontology (Žaková *et al.* 2011). Sin embargo, estos trabajos no consideran la descripción del dominio o los conceptos básicos en minería de datos (algoritmo, tipo de análisis, tarea, conjunto de datos, atributo, etc.), que pueden combinarse para definir entidades o restricciones.

Motivación

A pesar de los significativos avances en el uso de la semántica para el análisis de datos, todavía hay áreas considerables de oportunidad que, en parte, representan los desafíos de investigación presentados en esta tesis:

- Es crucial enfatizar las técnicas existentes para extraer o generar conocimiento directamente de las fuentes de datos e integrar este conocimiento con el conocimiento explícito declarado en las ontologías de dominio. Se debería desarrollar una ontología para los ítems de conocimiento para esta integración, permitiendo una representación formal y flexible. Este modelo semántico ofrece de manera consistente conocimiento de dominio a los algoritmos de análisis, permitiendo inferencias (razonamiento formal e hipotético) sobre el conocimiento recopilado para derivar conocimiento implícito del dominio.
- Se necesita un puente entre el conocimiento del dominio y el diseño algorítmico adaptado a este conocimiento. Desarrollar nuevos algoritmos que capturen experiencia y conocimientos externos a través de ontologías que representen el dominio de aplicación producirá nuevos operadores, mecanismos y restricciones (Parody *et al.* 2016). Estos aprovecharán la estructura semántica de los datos para inducir modelos de aprendizaje mejor adaptados al dominio de aplicación. Dichos algoritmos asimilarán naturalmente el conocimiento inyectado del dominio.
- Las ontologías actuales deberían ampliarse para abarcar nuevas familias de algoritmos de inteligencia artificial.
- Adaptar los modelos semánticos para operar en entornos de Big Data es esencial. Esto incluye generar flujos de trabajo y anotadores capaces de funcionar dentro de ecosistemas de Big Data e interoperar con modernas bibliotecas de procesamiento en tiempo real como Spark, Flink, etc.

Objetivos y fases

En esta Tesis, titulada *Semantics in Big Data*, abordamos algunos de los desafíos y soluciones a la gestión y procesamiento de grandes conjuntos de datos en entornos contextualmente ricos por medio de Tecnologías de la Web Semántica. Este enfoque permite una integración más eficiente y significativa de diversas fuentes de datos, lo que es especialmente relevante en la era del Big Data, donde la cantidad de información generada y almacenada crece exponencialmente. Por lo tanto, los principales objetivos abordados en esta tesis son como siguen:

O.1. Diseñar un framework para incorporar semántica en flujos de trabajo de Big Data.

- O.1.1. Desarrollar algoritmos que usen parametrizaciones y selección de operadores guiados por semántica a través del conocimiento (previamente anotado) de experiencias anteriores en estudios experimentales y literatura.
- O.1.2. Desarrollar componentes que encapsulen estos algoritmos para integrarlos en TITAN.

O.2. Implementar un razonador de ontologías escalable para su uso en algoritmos de Big Data.

- O.2.1. Analizar y diseñar un modelo de almacenamiento escalable.
- O.2.2. Diseñar e implementar un enfoque paralelo para procesar el conocimiento en el proceso de inferencia.

O.3. Diseñar y desarrollar ontologías y modelos semánticos para caracterizar las principales familias algorítmicas, sus operadores y su comportamiento.

- O.3.1. Desarrollar ontologías de dominio que puedan ser utilizadas para la inyección de conocimiento.
- O.3.2. Definir y desarrollar nuevas familias de algoritmos enriquecidas con el uso de la semántica en dominios específicos.

O.4. Definir una metodología para inyectar semántica en algoritmos de análisis de datos utilizando la experiencia adquirida en los objetivos anteriores.

- O.4.1. Analizar propuestas para la clasificación o extracción de conocimiento de grandes conjuntos de datos.

O.4.2. Establecer un mecanismo para el descubrimiento e integración de conocimiento necesario para el funcionamiento de los algoritmos de análisis de datos.

O.4.3. Diseñar una metodología para incorporar la semántica en los algoritmos.

Contribuciones de la tesis

Los principales retos y objetivos descritos en esta Tesis están asociados con las contribuciones incluidas en este documento de la siguiente manera:

- En el Capítulo 3, presentamos TITAN, un framework diseñado para gestionar el ciclo de vida de los flujos de trabajo en entornos de Big Data (**Objetivo O.1**). A través del desarrollo de la plataforma TITAN, intentamos proporcionar una herramienta para gestionar el ciclo de vida de flujos de trabajo en entornos distribuidos, cuya solución integra semántica de dominio (conocimiento experto) para facilitar la creación de flujos de trabajo más inteligentes y eficientes. TITAN se construyó con una arquitectura flexible, lo que permite la incorporación de nuevas funcionalidades.

- El Capítulo 4 está dedicado a abordar el **Objetivo O.2**, presentando el desarrollo de una arquitectura escalable optimizada para tareas de razonamiento en OWL (Lenguaje de Ontologías Web). Esta herramienta, denominada NORA, está diseñada para proporcionar razonamiento sobre grandes ontologías. Al utilizar NORA en conjunto con TITAN, se puede realizar un razonamiento eficiente y escalable sobre flujos de trabajo semánticamente ricos aprovechando tecnologías de bases de datos NoSQL para garantizar la escalabilidad y la fiabilidad de la solución. NORA hace uso de Apache Spark como motor computacional para implementar reglas de inferencia, permitiendo que el proceso de razonamiento se evalue de manera iterativa hasta que no se derive nuevo conocimiento inferido.

- En el Capítulo 5, presentamos SALON, una ontología específica de dominio para describir alineamientos múltiples de secuencias en el campo de la bioinformática (**Objetivo O.3**). SALON posibilita el desarrollo de repositorios de datos enlazados (Linked Data) para ofrecer un acceso uniforme a información diversa, siendo esencial para los investigadores en bioinformática. Esta ontología también puede funcionar como un esquema mediador para la integración de datos de diversas fuentes y validar alineamientos de secuencias mediante la definición de reglas SWRL.

- El Capítulo 6 se dedica a abordar el **Objetivo O.4** mediante el desarrollo de una metodología para inyectar conocimiento de dominio (definido por medio de ontologías) en algoritmos de Big Data. Esta ontología permite enriquecer los algoritmos con información específica del dominio, para generar decisiones más informadas y precisas sobre los datos. Varios casos prácticos demuestran la efectividad de META a la hora de mejorar el proceso de análisis, incluido su uso para mapear conocimiento y restricciones

de dominio en modelos de aprendizaje automático. A través de META, los algoritmos pueden ser guiados por conocimiento experto y consideraciones específicas del dominio.

Por último, identificamos varias direcciones prometedoras para trabajos futuros. Estas incluyen mejorar las capacidades semánticas de TITAN, extender las funcionalidades de NORA, y desarrollar interfaces intuitivas para META, con el objetivo de hacer la semántica en Big Data más accesible y eficiente para una amplia gama de usuarios.

Conclusiones y trabajos futuros

Los avances en el campo del Big Data han cambiado el panorama del procesamiento de datos e impuesto desafíos para unificar conocimientos de fuentes diversas. Tal y como se aborda en esta Tesis, la semántica de dominio ha logrado avances significativos en el análisis de datos. Sin embargo, aún existen desafíos, particularmente en la integración fluida de conocimiento a partir de fuentes de datos heterogéneas y en la adaptación de los propios algoritmos basándose en este conocimiento.

El tema central de esta Tesis gira en torno a entender el papel de la semántica a la hora de facilitar la integración de conocimientos en procesos de Big Data. El objetivo ha sido explorar formas de extraer, representar e inyectar conocimientos de manera fluida, asegurando que los procesos sean más inteligentes. Como resultado, hemos abordado los Objetivos establecidos en el Capítulo 1 tal y como se indica en las siguientes secciones.

Flujos de trabajo en Big Data

Con el crecimiento exponencial de Big Data, es imperativo crear soluciones basadas en semántica que sean inherentemente escalables. Además, los sistemas basados en conocimientos subrayan la importancia de entrelazar la semántica de dominio con el procesamiento de datos, proporcionando soluciones escalables y perspectivas significativas a partir de grandes conjuntos de datos. En base a esto, presentamos TITAN (**Ch3**), una plataforma de Sistema de Gestión de Flujos de Trabajo (WMS) diseñada explícitamente para abordar estas complejidades. La plataforma TITAN aprovecha los principios de la semántica de dominio para guiar diseños y validaciones de flujos de trabajo inteligentes (**O.1**). En este contexto, la ontología BIGOWL (Barba-González, García-Nieto, del Mar Roldán-García, *et al.* 2019) surgió de un enfoque impulsado por ontologías diseñado para facilitar la gestión del conocimiento dentro de los flujos de trabajo de análisis de Big Data. BIGOWL incluye un conjunto completo de conceptos, atributos y relaciones derivados del ecosistema de Big Data y optimización, tales como *Componente*, *Tarea*, *Algoritmo*, *Datos* y *Flujo de Trabajo*. La arquitectura de TITAN está diseñada para usar BIGOWL como elemento principal y ser inherentemente escalable, convirtiéndola en una solución robusta para los diversos desafíos planteados por Big Data. Así, las tareas dentro de un flujo de trabajo representan instancias específicas de componentes, cada uno de ellos definido por distintos valores de entradas, salidas y parámetros asociados. Esto nos permite la reutilización de componentes en distintos flujos de trabajo.

Razonamiento escalable

En paralelo, desarrollamos NORA (**Ch4**), un sistema de razonamiento escalable basado en tecnologías de Big Data como Apache Spark y Cassandra. NORA utiliza una estrategia de materialización para razonar. Este método produce nuevos datos, o *inferencias*, y los devuelve a la Knowledge Base. La implementación de las reglas de inferencia sigue un algoritmo de punto fijo utilizando Apache Spark, por lo que el proceso de razonamiento se evalúa iterativamente hasta que no se deriva nuevo conocimiento inferido.

La arquitectura central de NORA consta de dos etapas: un extractor de jearquías de OWL y un motor de razonamiento. Estos servicios trabajan juntos de manera efectiva para lograr un razonamiento escalable. En el primer paso del proceso, los datos obtenidos de cualquier Knowledge Base (KB), representados como una ontología OWL, se materializan en una base de datos Cassandra. Se utiliza un razonador de lógica descriptiva (DL) para pre-calcular la jearquía completa de la TBox como punto de partida. Usamos la API de OWL, una interfaz de Java y una implementación de referencia para el lenguaje OWL, para leer las ontologías OWL de entrada. Pellet, un razonador OWL 2 DL, se utiliza para clasificar en la TBox. Al combinar estas herramientas, obtenemos varias taxonomías, como las jerarquías de clase/subclase y propiedad/subpropiedad, pero no se infiere ningún conocimiento nuevo derivado de la ABox. Posteriormente, se crea un esquema de base de datos para materializar toda esta información (Roldan-Garcia and J. F. Aldana-Montes 2008). Como resultado, la TBox razonada se almacena en Cassandra junto con las instancias explícitas descritas en la Base de Conocimientos.

En el segundo paso, razonamos sobre la ABox traduciendo cada regla de inferencia como una expresión de Apache Spark, que son evaluadas iterativamente hasta que no se derive nuevo conocimiento inferido. Para ello, el conector de Spark para Cassandra puede leer datos (columnas y filas) de un espacio de claves y ejecutar métodos estándar de Resilient Distributed Dataset (conocido como RDD) para interactuar directamente con Cassandra. La transformación RDD incluye operaciones *map*, *sort* y *join* que no están disponibles de forma nativa en Cassandra. Esta abstracción nos permite manipular los datos distribuidos en muchas máquinas de forma distribuida. El motor de razonamiento evalúa las reglas de inferencia, expresadas como trabajos de Spark, accediendo a la base de datos con datos materializados. Las reglas de inferencia se aplican repetidamente hasta que no se produzcan más declaraciones inferidas. En nuestro caos, usamos una caché externa[1] para acelerar esta operación cargando información

[1]En URL: `https://redis.io`

de la base de datos en la caché para operaciones de búsqueda rápidas.

Nuestra elección de NoSQL para el almacenamiento fue intencional para proporcionar escalabilidad y fiabilidad al sistema. Sin embargo, vale la pena señalar que tuvimos que compensar la ausencia de algunas características relacionales encontradas en otros sistemas SQL, como las operaciones de unión, integrando Apache Spark, a expensas de un mayor tiempo de cómputo.

Sin embargo, NORA puede manejar conjuntos de datos masivos sin comprometer el rendimiento. También complementa a TITAN al mejorar sus capacidades (**O.2**). TITAN simplifica la gestión de flujos de trabajo e integra datos semánticamente ricos. En contraste, NORA proporciona el poder de cómputo para realizar un razonamiento profundo y escalable sobre estos flujos de trabajo cuando sea necesario.

Hemos validado nuestro enfoque utilizando el Lehigh University Benchmark (LUBM) y el University Ontology Benchmark (UOBM). La experimentación inicial muestra resultados prometedores en términos de escalabilidad, tiempo de ejecución y completitud. Nuestro trabajo es de código abierto, por lo que está abierto a mejoras por parte de la comunidad.

Ontologías biológicas

También hemos presentado SALON (**Ch5**), una ontología diseñada para asegurar que los alineamientos de secuencias biológicas se interpreten y utilicen de manera consistente (**O.3**). Identificar secuencias biológicas es probablemente una de las tareas más críticas en bioinformática. Los alineamientos de secuencias tienen muchas aplicaciones, como la búsqueda de genes, la predicción de funciones de genes o el ensamblaje de secuencias genómicas. Por lo tanto, la disponibilidad de bases de datos biológicas con información sobre alineamientos de secuencias correctos y completos es particularmente interesante para científicos e investigadores en esta área. Anotar el alineamiento de secuencias biológicas (y su información asociada) con una ontología permite el desarrollo de repositorios de datos vinculados, proporcionando acceso homogéneo a información heterogénea y habilitando aplicaciones inteligentes y avanzadas.

La ontología propuesta, disponible en `https://w3id.org/salon`, consta de 30 clases, 14 propiedades de objeto, incluyendo 4 propiedades de objeto inversas para asegurar la vinculación con recursos relacionados, 27 propiedades de datos y 144 axiomas lógicos. Las clases de nivel superior de la ontología son *Alignment*, *AlignmentColumn*, *AlignmentScore*, *AlignmentScoreFunction*, *AlignmentSequence*, *ColumnScore*, *ColumnScoreFunction*, *ConstructionMethod*, *Feature*, *Protein* y *SubAlignment*.

La clase *Alignment* representa un alineamiento de secuencias. Un *Alignment* contiene al menos un *SubAlignment*, representando un subconjunto de secuencias con alguna propiedad en común. Esto puede ser útil para representar clusters de secuencias dentro del alineamiento, por ejemplo, árboles filogenéticos.

Los atributos de *AlignmentColumn* incluyen *ColumnScore*, *ColumnWeight* y *ConsensusCharacter*, entre otros. Algunos atributos de *AlignmentSequence* son *organism* o *keyword*.

Para validar alineamientos de secuencias, SALON se puede explotar definiendo reglas SWRL, que determinan automáticamente si un alineamiento es plausible basándose en su puntuación global asignada. Este valor numérico se usa para diferenciar los buenos alineamientos de los pobres. La puntuación del alineamiento cuando contamos con dos secuencia se calcula en base a coincidencias/desajustes y huecos entre ambas secuencias. En alineamientos múltiples, con tres o más secuencias, esta puntuación puede determinarse en base a funciones de puntuación, que se calcula de forma independiente para cada columna del alineamiento. La puntuación total será la suma de estos valores. SALON incluye varias reglas SWRL para determinar si algún valor de puntuación está fuera de rango (basándose en la función de puntuación subyacente), en base a sus límites superiores e inferiores. Estas reglas pueden ayudar a los científicos a asegurar automáticamente la corrección de la alineación.

Además, la ontología podría actuar como un esquema mediador para la integración virtual de datos de diferentes fuentes utilizando consultas SPARQL federadas. La naturaleza distribuida de los datos biológicos (recogidos en distintas fuentes de datos en línea) hace que SALON sea particularmente útil en la integración de información. Por ejemplo, mediante el uso de consultas SPARQL federadas, SALON puede utilizar la información almacenada en el repositorio de SALON sobre secuencias de proteínas para integrar datos de otras fuentes de datos como UniProtKB, una de las bases de datos más conocidas de estructuras y información funcional de proteínas, para recuperar información sobre proteínas, incluyendo organismo fuente, nombre, identificador NCBI y gen.

Así, SALON no solo proporciona una ontología robusta para la anotación y validación de alineamientos de secuencias biológicas, sino que también facilita la integración y el análisis de datos biológicos de múltiples fuentes, potenciando la investigación en bioinformática y otras áreas relacionadas.

Inyección de semántica en algoritmos

Es importante destacar que ninguno de los sistemas mencionados anteriormente considera inherentemente la implementación algorítmica. Adaptar algoritmos

basados en semántica de dominio asegura un procesamiento de datos eficiente y garantiza que los resultados se alineen con el conocimiento y requisitos específicos del dominio.

En vista de esto, hemos trabajado en una metodología para inyectar conocimiento semántico en algoritmos de análisis de Big Data (**O.4**). Nuestro enfoque utiliza la ontología META. Dentro de este framework, modelamos conceptos derivados del Web Ontology Language (OWL), como clases y propiedades, restricciones y preferencias del usuario, como elementos OWL. Al hacerlo, buscamos ofrecer un mecanismo para integrar conocimientos de varias fuentes de información y habilitar su reutilización en diferentes aplicaciones.

Abstract

In this thesis, entitled "Semantics in Big Data Analytics", we address challenges and solutions for managing and processing large-scale datasets in semantic environments.

Through the development of the TITAN platform, we aim to provide a tool for managing the lifecycle of workflows, integrating semantics to facilitate more intelligent and efficient workflows. TITAN was built with a flexible architecture, allowing for the implementation of new functionalities.

In this regard, we developed NORA, a tool designed to provide reasoning over large ontologies. Using NORA with TITAN, efficient and scalable reasoning can be performed on semantically rich workflows, leveraging NoSQL database technologies to ensure scalability and reliability. NORA uses Apache Spark as its computational engine to implement inference rules, allowing the reasoning process to be evaluated iteratively until no new inferred knowledge is derived.

In the biological domain, we introduce SALON, an ontology that provides a consistent understanding and use of multiple sequence alignments. SALON eases the development of Linked Data repositories to offer uniform access to diverse information essential for bioinformatics researchers. This ontology can also serve as a mediator schema for integrating data from various sources and validating sequence alignments by defining SWRL rules.

Furthermore, we explore a methodology to inject semantic knowledge (expressed via ontologies) into analysis algorithms using the META ontology. This ontology allows algorithms to be enriched with domain-specific information, resulting in more informed and accurate decisions. Several use cases demonstrate META's effectiveness in enhancing the analysis process, including its use for mapping domain knowledge and constraints into machine learning models. Through META, algorithms can be guided by expert knowledge and domain-specific considerations.

Lastly, we identify several promising directions for future work. These include enhancing the semantic capabilities of TITAN, extending NORA's functionalities, and developing intuitive interfaces for META to make semantics in Big Data more accessible and efficient for a broad range of users.

Chapter 1

Introduction

The processing challenges posed by Big Data stem from the vast amount of data, the diversity of information sources, and the speed at which they are generated. These challenges associated with Big Data are conceptualized in several dimensions (the well-known "V's"), and their advancement has facilitated numerous solutions throughout the entire data value chain (Al-Sai *et al.* 2019).

Knowledge sources about a field can be of various types. One can gain knowledge of an area by consulting an expert or by automatically extracting it from a data repository, a corpus of documents in different formats (XML, JSON, etc.), a webpage, and so forth. Some techniques facilitate such extraction, like ontology learning or schema inference (Buitelaar *et al.* 2005). However, the representation of this knowledge is not standardized, making its integration from different sources challenging.

In recent years, many efforts have emerged aiming to use domain semantics for data analysis (Dou *et al.* 2015). These techniques leverage formal definition and their ability to infer knowledge and check consistency to guide the analysis process. Current literature includes contributions incorporating domain semantics expressed through ontologies as part of data mining applications (Sinha *et al.* 2021). In these applications, semantics are used with different objectives: to reduce the search space by specifying restrictions, to filter results in the post-processing stage, to semantically annotate the results of the mining process, to annotate different clusters, etc. We can also find recent works that include ontologies to guide machine learning and deep learning processes, such as Pinto *et al.* 2015, which uses an ontology in the classification process, leveraging their ability to infer inconsistencies between concepts. In Phan *et al.* 2015, a deep learning model based on ontologies is proposed to predict human behavior.

Similarly, a research trend that focuses on designing ontologies for the semantic annotation of data processing and analysis task flows (workflows), aiming to incorporate knowledge into the analysis and knowledge discovery processes in large datasets, is emerging. Among the most recent works using OWL ontologies

to model the input and output of algorithms involved in data mining and knowledge discovery workflows to generate valid compositions, we can find KDDONTO (Diamantini *et al.* 2009), DMWF (Kietz *et al.* 2010), and KD Ontology (Žaková *et al.* 2011). However, these works do not consider the description of the domain or the basic concepts in data mining (algorithm, type of analysis, task, dataset, attribute, etc.), which can be combined to define entities or restrictions.

1.1 Motivation

Ontologies are the backbone of various data mining processes, providing the much-needed structure and semantics to data. Yet, a gap remains in their application for performance optimization of data mining algorithms. Existing ontologies lack the detailed granularity necessary to support advanced concepts such as meta-mining and meta-learning. Meta-learning (Lemke *et al.* 2015) is defined as the process using machine learning techniques to analyze metadata related to experiments using these techniques, meaning leveraging meta-knowledge extracted from experience. In Nguyen *et al.* 2014, meta-mining is described as the KDD (Knowledge Discovery in Databases) procedure to enhance performance in data mining processes. Consequently, there is a growing interest in using semantics to improve or refine the data mining processes. In this context, the European FP7 e-LICO initiative [1] gave rise to the DMOP (Data Mining OPtimization Ontology) (Keet *et al.* 2015). This ontology characterizes analysis workflows and delves into the description of algorithms, operators, parameters, inputs/outputs, and a significant portion of the metadata involved in data mining processes. Despite significant progress in using semantics for data analysis, there are still considerable areas of opportunity that, in part, represent the research challenges presented in this thesis:

- It is crucial to emphasize existing techniques for extracting or generating knowledge directly from data sources and integrating this knowledge with the explicit knowledge declared in domain ontologies. An ontology should be developed for knowledge items for this integration, allowing formal and flexible representation. This semantic model consistently offers domain knowledge to analysis algorithms, enabling inferences (formal and hypothetical reasoning) about the gathered knowledge to derive implicit domain knowledge.
- A bridge between domain knowledge and algorithmic design tailored to this knowledge is needed. Developing new algorithms that capture external expertise and experiences through ontologies representing the application domain will produce novel operators, mechanisms, and constraints (Parody *et al.* 2016). These will harness the semantic structure of data to induce learning models better adapted to the application domain. Such algorithms will naturally assimilate the domain's injected knowledge.
- Current ontologies should be expanded to encompass new families of artificial intelligence algorithms.

[1] In URL: `http://www.e-lico.eu/`

- Adapting semantic models to operate in Big Data environments is essential. This includes generating workflows and annotators capable of functioning within Big Data ecosystems and inter-operating with modern streaming processing libraries such as Spark, Flink, etc.

1.2 Objectives and Phases

This thesis is framed within the "SENSACIÓN–Semántica de dominio en algoritmos de análisis del (Big) Data)" project (TIN2017-86049-R, funded by the Spanish Ministry of Science and Innovation), as its objectives are strongly aligned with it. As a result, the primary objectives of this thesis are articulated as follows:

O.1. Design a framework to deal with the use of semantics in Big Data workflows.

- O.1.1. Develop algorithms that use parameterizations and operator selection guided by semantics through knowledge (previously annotated) of prior experience in experimental studies and literature.
- O.1.2. Develop components that encapsulate these algorithms to integrate them into TITAN (a result of the SENSACIÓN project).

O.2. Implement a scalable ontology reasoner for its use in Big Data algorithms.

- O.2.1. Analyse and design a scalable storage model.
- O.2.2. Design and implement a parallel approach for processing the knowledge in the inference process.

O.3. Design and develop ontologies and semantic models to characterize the main algorithmic families, their operators, and their behavior.

- O.3.1. Develop domain ontologies that can be used for knowledge injection.
- O.3.2. Define and develop new families of algorithms enriched with the use of semantics tailored for domain-specific environments.

O.4. Define a methodology to inject semantics into data analysis algorithms using the experience acquired in the previous goals.

- O.4.1. Analyze proposals for classification or knowledge extraction from large datasets, such as Ontology Learning and Schema Inference.
- O.4.2. Establish a mechanism for the discovery and integration of knowledge necessary for the operation of data analysis algorithms.
- O.4.3. Design a methodology to incorporate semantics into the algorithms.

1.3 Thesis Contributions

The primary challenges and objectives outlined in this thesis are associated with the contributions included in this document as follows:

- In Chapter 3, we introduce TITAN, a metadata framework designed to manage the lifecycle of workflows in Big Data environments (**Objective O.1**). This software tool leverages Semantic Web technologies to enrich data with user-specific domain annotations. Moreover, TITAN provides an interactive interface that actively guides users in developing and optimizing Big Data workflows.
- Chapter 4 is dedicated to addressing **Objective O.2**, featuring the development of a scalable architecture optimized for reasoning tasks in OWL (Web Ontology Language). This tool employs Big Data technologies such as Apache Spark and Cassandra to materialize reasonings and distribute them across a computing node network (cluster). This design markedly decreases the time required for data processing and substantially increases the data volume that can be accommodated for reasoning tasks. In doing so, it outperforms existing solutions in terms of both speed and scalability.
- In Chapter 5, we present SALON, a domain-specific ontology for describing multiple sequence alignments in the field of bioinformatics (**Objective O.3**). SALON can serve as a mediating schema, facilitating the seamless integration of heterogeneous data related to sequence alignments by standardizing the representation of such data.
- Chapter 6 is devoted to tackling **Objetive O.4** by developing a methodology for injecting domain knowledge into Big Data algorithms.

Finally, Chapter 7 summarizes the main outcomes of this thesis and the significance of each contribution in the broader context of Big Data and domain semantics.

The development of these objectives has led to a series of publications in high-impact journals. Specifically, three articles have been published in journals indexed in the Journal Citation Reports (JCR), with one more in the process of being published, as detailed below:

- A. Benítez-Hidalgo, C. Barba-González, J. García-Nieto, P. Gutiérrez-Moncayo, M. Paneque, A. J. Nebro, M. del Mar Roldán-García, J. F. Aldana-Montes, I. Navas-Delgado, **TITAN: A knowledge-based platform for Big Data workflow management** (Benítez-Hidalgo, Barba-González, *et al.* 2021).

This article focuses on TITAN, a knowledge-based platform for managing Big Data workflows. The main contributions of the platform include the improvement and extension of BIGOWL to enable the automatic deployment of workflows in Big Data environments, thus providing validation mechanisms to ensure the quality of workflows from the design stages before their execution. The platform also includes developing tools to support the domain contextualization of data processed and Big Data workflows and specific tooling (i.e., Graphical User Interface) designed for creating and executing Big Data analytic workflows in distributed environments.

- A. Benítez-Hidalgo, I. Navas-Delgado, M. del Mar Roldán-García, **NORA: Scalable OWL reasoner based on NoSQL databases and Apache Spark** (Benítez-Hidalgo, Navas-Delgado, *et al.* 2023).

  This article presents NORA, a scalable OWL (Web Ontology Language) reasoner designed to handle large knowledge bases. Built on Apache Spark and utilizing NoSQL databases, NORA addresses the limitations of traditional reasoning techniques, which often struggle with extensive knowledge bases. The system materializes OWL data in the Apache Cassandra database to facilitate scalable reasoning. NORA's main contributions lie in its scalability, enhanced performance, and thorough evaluation against various benchmarking ontologies, offering a more efficient way to conduct reasoning over large-scale ontologies.

- A. Benítez-Hidalgo, J. F. Aldana-Montes, I. Navas-Delgado, M. del Mar Roldán-García, **SALON ontology for the formal description of sequence alignments** (Benítez-Hidalgo, J. F. Aldana-Montes, *et al.* 2023).

  Sequence alignment is a critical process in bioinformatics for identifying regions of similarity in biological sequences like DNA, RNA, or proteins. However, the field lacks a standardized vocabulary and representation for these alignments. This gap makes it challenging to correlate existing biological knowledge with newly generated alignment data automatically. To address this issue, we introduce the Sequence Alignment Ontology (SALON). SALON aims to provide a comprehensive, controlled vocabulary for representing and semantically annotating pairwise and multiple sequence alignments. The ontology is built on OWL 2 and supports automated reasoning, which facilitates the validation of alignments and easier retrieval of related information from public databases. One of the features of SALON is its compatibility with the Open Linked Data approach. SALON allows for seamless data integration from various sources, making it easier for scientists and researchers to interpret alignment results and validate acquired

knowledge. By serving as a mediated schema, SALON integrates data and ensures its validity, thereby reducing the effort needed for manual interpretation.

In addition to these direct contributions, I have had the opportunity to actively participate in two national conferences where I disseminated the results of my research. In these conferences, I specifically presented academic papers that stem from this thesis. These presentations took place at the SISTEDES conference, a reputable forum for experts and academics in the field.

Chapter 2

Background and Preliminaries

This chapter provides an overview of the theoretical foundations and the state of the art related to Big Data, workflow management systems, and Semantic Web technologies.

2.1 Big Data

Big Data refers to datasets that are so large and complex that they require specialized technologies and methods for their handling, storage, analysis, and visualization. Big Data challenges include data capture, storage, search, sharing, analysis, and visualization. Many technologies (most of them open-source projects, like those from the Apache Foundation[1]) have emerged in recent years around Big Data to propose generic solutions that can address any of these challenges.

In terms of large-scale data handling, for instance, there are various proposals to efficiently harness distributed computing, which ultimately speeds up data processing time. However, one of the main focuses today is to derive value from all available data. Extracting this value is not an easy task that can be solved with a single technology: there are many approaches related to Big Data analysis (i.e., the use of advanced analysis techniques applied to large and diverse datasets) coming from heterogeneous vertical domains, such as transportation (Neilson *et al.* 2019), healthcare (W. Raghupathi and V. Raghupathi 2014), e-science (Taylor *et al.* 2007), and agriculture (Wolfert *et al.* 2017).

With the data explosion and expectations to utilize it, organizations begin to automate data extraction processes, resulting in data repositories from which extracting value is a challenge. In this sense, using semantics, including contextual information, is a promising approach to handling data, enhancing data analysis processes, and ensuring efficient reuse of software components (Barba-González, García-Nieto, Roldán-García, *et al.* 2019; Gil, Ratnakar, *et al.* 2011).

2.1.1 Challenges

As we mentioned, Big Data presents unique challenges that require specialized processing solutions:

1. **Storage**: With the sheer volume of Big Data, the primary challenge lies in storing it efficiently. Traditional storage solutions may not suffice, especially considering the need for redundancy, distribution, and fault tolerance. Moreover, the dynamic nature of data generation means that storage solutions must be scalable, adapting to ever-growing datasets without significant downtimes or overhauls.

2. **Search**: As datasets grow exponentially, pinpointing specific pieces of information becomes akin to finding a needle in a haystack. Efficient search mechanisms that can rapidly comb through terabytes or even petabytes of

[1] In URL: `https://www.apache.org`

data to retrieve pertinent details are crucial. The challenge is magnified when the data is unstructured or semi-structured, making conventional querying techniques less effective.

3. **Sharing**: In an interconnected digital ecosystem, data often needs to be accessed by multiple stakeholders, be it departments within an organization or different entities in a supply chain. Facilitating seamless data sharing while ensuring data integrity, consistency, and security becomes a complex endeavor, especially when dealing with large volumes of concurrent access requests.

4. **Analysis**: The true value of Big Data lies in the insights it can provide. However, sifting through vast datasets to extract meaningful patterns, trends, or anomalies requires sophisticated analytical tools and algorithms. The diverse nature of Big Data, encompassing structured and unstructured formats, adds an extra layer of complexity to the analysis.

5. **Visualization**: Conveying the insights derived from Big Data in a manner understandable to stakeholders, regardless of their technical expertise, is pivotal. Crafting intuitive visualizations, dashboards, or reports that encapsulate the core findings while dealing with the granularity and volume of the data, poses its own set of challenges.

6. **Real-time Processing**: In an age of instant gratification, many applications, from stock trading to health monitoring, require instantaneous insights. Processing vast incoming data streams in real-time or near-real-time and delivering actionable results without lags necessitates highly optimized and responsive solutions.

2.1.2 NoSQL Databases

The rapid growth of data-intensive applications and the need to manage large volumes of structured data has driven the emergence of NoSQL (Not Only SQL) databases as alternatives to traditional Relational Database Management Systems (RDBMSs). Unlike relational systems, NoSQL databases are designed to handle large volumes of unstructured or semi-structured data, such as documents, graphs, key-value, or columns, at the expense of sacrificing query capabilities and consistency guarantees (Gessert *et al.* 2017).

The most common distinction between NoSQL databases is how they store and allow access to data, although there are other essential considerations to remember (see Figure 2.1). In particular, scalability is a fundamental aspect in the design of database systems, and there are two main approaches to achieve

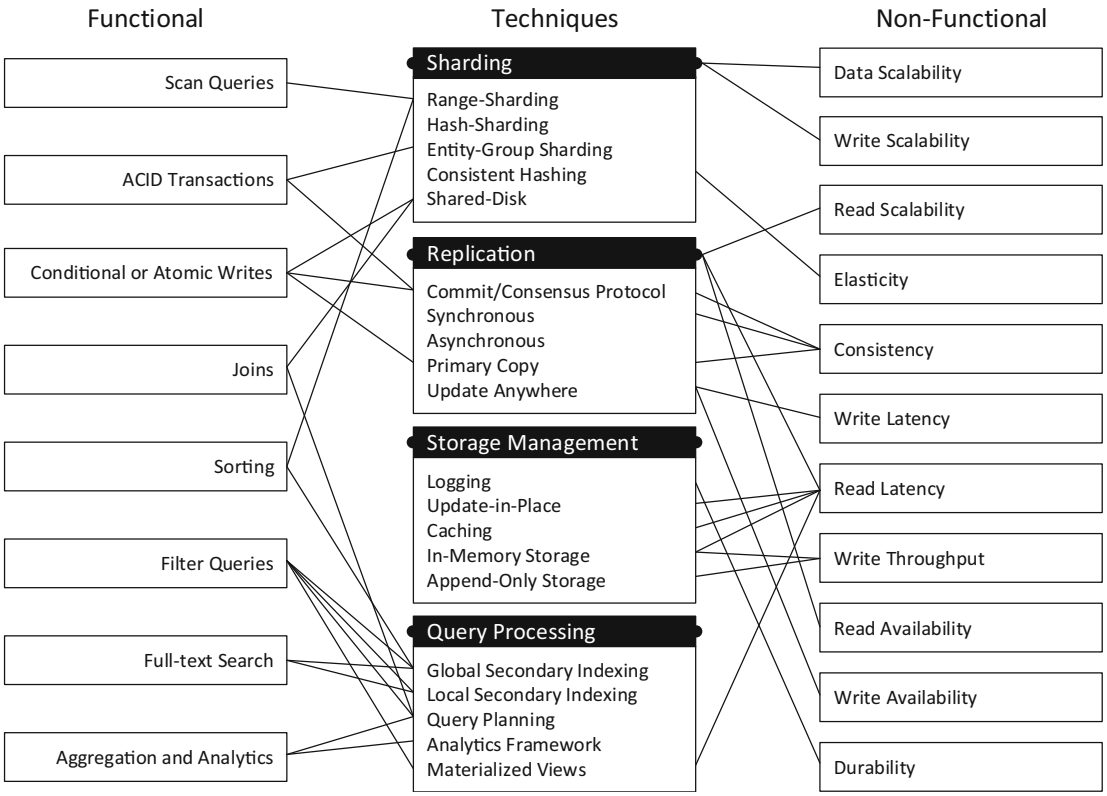

Figure 2.1: Main techniques available used by NoSQL databases and related to different functional and non-functional properties they allow. Source: Gessert *et al.* 2017.

it: horizontal scalability and vertical scalability:

- **Vertical scalability** involves increasing the processing and storage capacity of an existing machine or node, usually by enhancing its resources, such as adding more memory, a faster CPU, or larger capacity disks. This approach is suitable when the workload and storage requirements can be managed within the limits of a single machine. However, there are limitations to how much a machine can be vertically scaled, and it can be costly or complex when significant performance increases are required. Relational databases are generally better suited in environments where vertical scalability is required.
- **Horizontal scalability** refers to a system's ability to handle an increased data volume and a larger workload by adding more machines or nodes to the system. Instead of relying on a single powerful machine, the load is distributed among multiple nodes, allowing parallel processing and better performance. This approach is beneficial when exponential data growth is

expected or when high availability and fault tolerance are needed. Additionally, horizontal scalability makes it easier to expand the infrastructure as storage and performance requirements increase. NoSQL databases are designed to scale horizontally efficiently, distributing the data across multiple servers or nodes in a cluster.

Apache Cassandra

Apache Cassandra (Lakshman and Malik 2010) is a highly scalable and distributed NoSQL database designed to handle large volumes of structured data. Cassandra is designed to be fault-tolerant and highly available. Data is replicated across multiple nodes in a cluster, ensuring they are available even in the event of hardware failures or individual node downtimes. Moreover, Cassandra provides consistency tolerance, meaning data is asynchronously replicated and can be read and written from any node, ensuring high availability and fast performance.

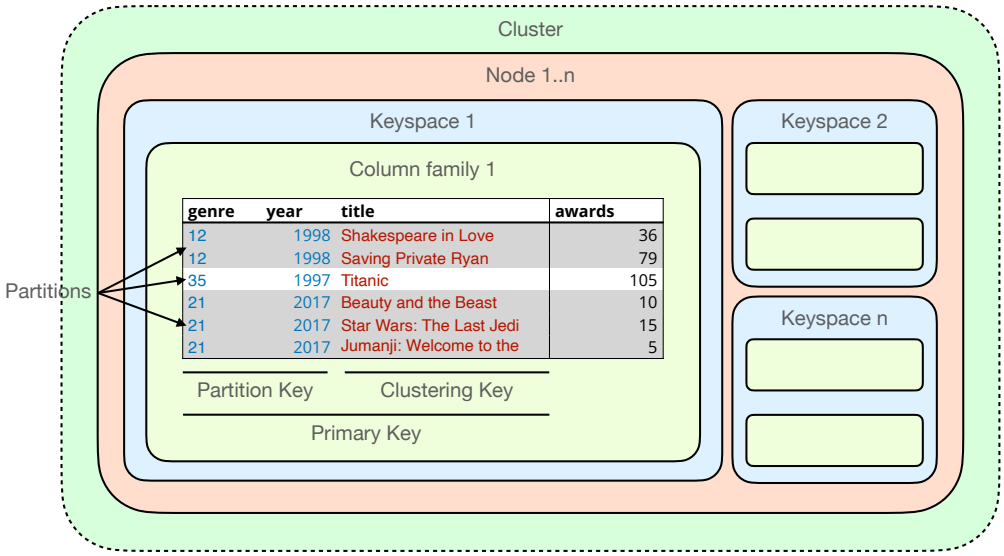

Figure 2.2: Apache Cassandra data model. Source: author's own.

Figure 2.2 shows the primary elements of the Cassandra data model. At its core, a keyspace governs how data is replicated across nodes. Typically, a Cassandra cluster will have a distinct keyspace for each application. Data within this system is arranged in rows or columns; thus, the primary identifier for a table in Cassandra might be a combined key of multiple columns. When employing such keys, the first column is referred to as the partition key, responsible for uniquely identifying a record in the table and for data distribution across the

nodes. Subsequent columns are denoted as clustering keys, which dictate the arrangement of data within that partition.

2.1.3 Large-Scale Data Processing

With the exponential growth of data generated by various sources such as social networks, sensors, business transactions, and mobile devices, the need arises to develop techniques and tools that allow for managing, analyzing, and extracting knowledge from these massive volumes of information.

Large-scale data processing relies on the distribution and parallelization of computational tasks to achieve optimal performance in highly scalable hardware environments. A key technology in this field is distributed processing, which allows the division of work into multiple processing nodes and simultaneous execution. Distributed processing leverages the computing power of server clusters or distributed systems, thus achieving a higher processing capacity and reducing the time needed to complete complex tasks. Over the past decade, many tools and frameworks have emerged, each designed to tackle specific facets of Big Data processing:

Apache Hadoop and MapReduce

One of the most well-known frameworks for Big Data processing is *Apache Hadoop* (Bhandarkar 2010). It provides a distributed storage system, the Hadoop Distributed File System (HDFS), and MapReduce processing framework. Hadoop allows data to be stored in its native format and processes it in a distributed manner, enabling scalability and fault tolerance.

Apache Spark

While Hadoop's MapReduce requires data to be stored on disk between operations, *Apache Spark* (Zaharia, Chowdhury, Franklin, *et al.* 2010) allows for in-memory data storage and faster processing times for specific applications. Spark is particularly suited for iterative algorithms like machine learning and streaming solutions. In this regard, Apache Spark has become one of the most popular and widely used tools for efficient data processing. Spark's architecture is based on the concept of RDD (Resilient Distributed Datasets; Zaharia, Chowdhury, Das, *et al.* 2012, see Figure 2.3), an immutable and distributed collection of objects. RDDs allow data to be split into partitions and processed in parallel across a cluster of nodes. This data structure provides fault tolerance and automatic recovery during errors. RDDs represent collections of segments, known as partitions, spread out over various nodes within a cluster. In Apache Spark, these

partitions are the foundational elements for parallel processing.

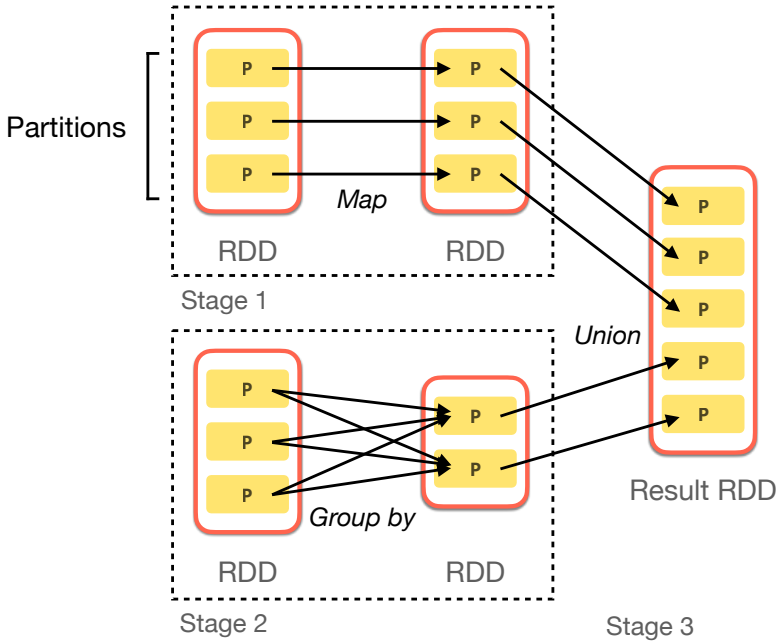

Figure 2.3: Apache Spark's operations on RDDs. Source: author's own.

Apache Kafka, Storm and Flink

For real-time data processing, tools like *Apache Kafka*, *Storm*, and *Flink* are essential. Apache Kafka is primarily known as a distributed event streaming platform but is often used with other Big Data tools to facilitate real-time processing. It provides high-throughput, fault-tolerant, and scalable message brokering, making it a go-to choice for many companies that need to process large volumes of real-time data. Kafka's ability to seamlessly integrate with various data processing frameworks and databases amplifies its utility in Big Data ecosystems. Apache Storm is specifically designed for real-time stream processing. Unlike batch processing systems like Hadoop, Storm processes data as it arrives, making it highly suitable for applications that require immediate insights or actions, such as fraud detection or monitoring platforms. Its distributed nature ensures scalability, and its ability to guarantee data processing, even in the event of node failures, makes it a robust choice for mission-critical applications.

While Storm and Kafka provide capabilities for stream processing, Apache Flink elevates it by offering stream and batch processing within a single platform. Its core is a streaming dataflow engine, which means it processes data

as it comes in but can revert to batch processing when required. Flink's standout feature is its ability to manage stateful computations over data streams. This provides enhanced capabilities for more sophisticated analytics and event-driven applications. Furthermore, its high-throughput and low-latency processing capabilities have positioned Flink as a powerful tool in the Big Data processing toolbox.

2.2 Workflow Management Systems

Over the past few decades, Workflow Management Systems (WMS) have gained popularity in the scientific community, mainly due to the rapid expansion of data volume. These systems can be classified into two distinct categories: task-oriented WMS and data-oriented WMS (Mitchell *et al.* 2019). On one hand, in the task-oriented approach, completing a task in the workflow triggers the execution of dependent tasks. On the other hand, in the data-oriented approach, functions in the workflow are initiated when input data is available rather than depending on task completion.

Various studies have defined analytical (macro)data tools, both task and data-oriented, to design and execute analytical data workflows. These studies not only automate repetitive tasks but also manage complex analysis processes at different levels of detail and systematically capture provenance information for derived data products. Some approaches even leverage metadata to assist users in designing and implementing workflows.

2.2.1 Task-driven workflow management systems

VisTrails (Scheidegger *et al.* 2008), developed at the University of Utah, aims to facilitate and streamline scientific data exploration. It integrates with various tools, libraries, and visualization systems while capturing provenance data for data products and workflow designs. VisTrails also employs its engine to manage execution, allowing workflows to run online or locally (Freire *et al.* 2006).

In this sense, YesWorkflow (McPhillips *et al.* 2015) emerged in 2015 as a suite of software tools. It enables scientists to annotate Python, R, or Perl scripts with special comments. These annotations improve the provenance of scientific outcomes, which can be queried to enhance the trustworthiness and reusability of workflows.

KNIME (Konstanz Information Miner) (Berthold *et al.* 2009) stands out in the realm of open-source analytics tools. It offers a robust, modular, and highly scalable platform for designing and executing workflows. KNIME includes data loading, transformation, analysis, and visual exploration components. Although it is not well-suited for streaming data, KNIME addresses this issue with a streaming executor allowing concurrent node execution.

Airflow (*Apache Airflow Documentation* 2019) is another open-source workflow manager that allows users to design and execute workflows as Directed Acyclic Graph (DAG) tasks. It has an architecture capable of distributing tasks

across many workers and servers. The user interface aids in visualizing pipelines, monitoring progress, and troubleshooting.

Taverna (Wolstencroft *et al.* 2013) is an open-source workflow tool suite widely used in the e-science community. It integrates distributed Web Services and local tools into complex analysis pipelines. However, it does not focus on Big Data analytics. Taverna operates in various execution environments and offers a desktop GUI for more accessible workflow design.

Workflow Instance Generation and Specialisation (WINGS) (Gil, Ratnakar, *et al.* 2011) exists in the sphere of semantic-aware tools. It enables scientists to design computational experiments via semantic representations that articulate data and computational constraints. WINGS ensures compatibility by checking semantic rules and provides web-based access.

Expanding on semantic assistance in predictive Big data analysis, Kumara *et al.* 2015 proposed using Automatic Service Composition (ASC). This framework produces an abstract workflow and supports semi-automated model selection via analytics ontology.

MINT (Gil, Cobourn, *et al.* 2018) is a framework that employs semantic representations based on the Geoscience Standard Names ontology (S. D. Peckham 2014) to describe datasets and models. It guides users in discovering appropriate model and data combinations and visualizing results.

Similarly, Makeflow (Albrecht *et al.* 2012) is a command-line workflow engine that represents data-intensive workflows. It requires explicit input data specification and supports various execution environments, including local and HTCondor (Tannenbaum *et al.* 2001).

Lastly, BioMOBY (Wilkinson and Links 2002) uses semantic approaches to validate scientific workflows by defining data types for the inputs and outputs of components.

2.2.2 Data-driven workflow management systems

Various general purpose, data oriented workflow management systems have emerged in recent years (Atkinson *et al.* 2017), in addition to MapReduce-based solutions like Apache Hadoop, Apache Yarn (Vavilapalli *et al.* 2013), and Apache Spark.

Pachyderm (Novella *et al.* 2019) is an open-source workflow and data management platform. In Pachyderm, a workflow is termed a 'pipeline' centered around data repositories, which serve as nodes in a Directed Acyclic Graph

(DAG). The system runs a pipeline on its data and asynchronously waits for new data to process. Pachyderm is agnostic to data format and programming languages, and it adopts the principle of a single conceptual store instead of separate storage buckets. It operates on numerous software layers and is compatible with major commercial cloud providers like Amazon S3 (Palankar *et al.* 2008), Microsoft Azure (Wilder 2012), and Google Cloud (Mishra *et al.* 2010).

Similarly, Nextflow (Di Tommaso *et al.* 2017) allows workflows to be executed locally or in various cluster environments. It supports software execution through Docker (Merkel 2014) or Singularity (Kurtzer 2016) clients, depending on the operating system. Like Pachyderm, Nextflow runs pipelines on its data and is fully integrated with version control platforms such as GitHub, allowing workflows to pull updates and data from specified repositories.

VLAM-G (Korkhov *et al.* 2007) is another tool designed as an interactive, data-driven workflow engine focused on the e-science community. Built on core Grid services like resource management and data access, VLAM-G features a Run-Time Environment for workflow components and a Run-Time System Manager to oversee execution. Workflows are assembled by linking components through data dependencies, which can be custom software for VLAM-G, web services, or legacy software interfaces. The tool also enables real-time monitoring and interactive parameter adjustments of the connected modules.

Karma2 (Simmhan *et al.* 2008) is geared towards provenance management in data-driven workflows. In this setup, individual services within a workflow publish their provenance data to reduce performance overhead at the workflow engine level. Karma collects this metadata as a workflow trace and stores it in a centralized database. This metadata includes information about the task nodes in workflows and operates independently of the specific workflow management system. The gathered information can be used to generate workflow execution traces that illuminate producer/consumer relationships in data exchange between services.

2.3 Semantic Web Technologies

The Semantic Web, as proposed by Tim Berners-Lee, the inventor of the original web, is an extension of the traditional web in which information has a well-defined meaning, allowing computers and people to work together (Hitzler *et al.* 2012). Unlike the conventional web, primarily designed for human use, the Semantic Web is intended for computers to understand and interpret information. This allows for tighter data integration, providing a framework for sharing and reusing data across various applications, businesses, and user communities.

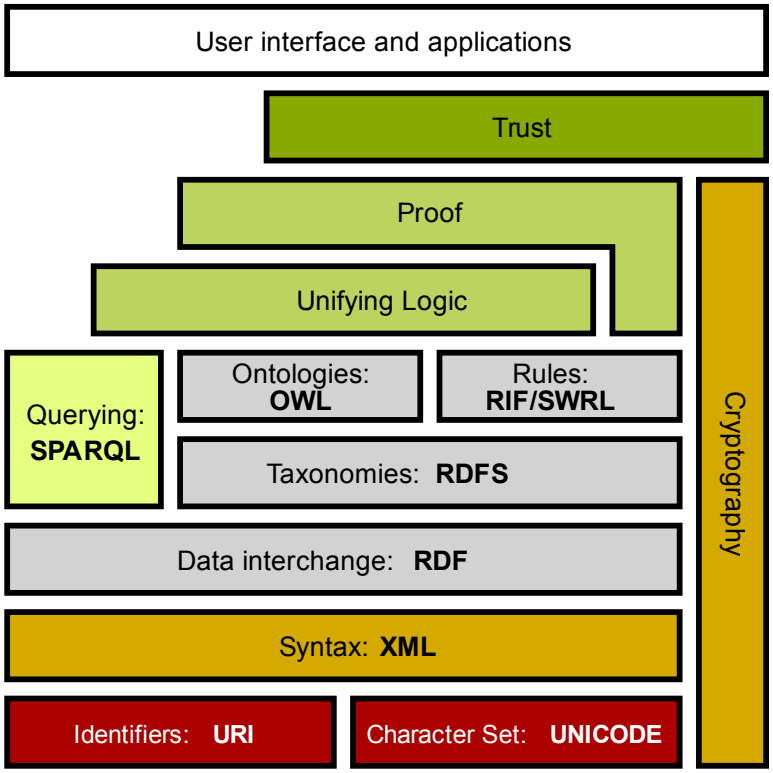

Figure 2.4: Overview of Semantic Web Technologies. Source: author's own.

Semantic Web technologies are the tools and methods that allow this concept to work in practice. These technologies enable the creation, publication, and linking of data on the web so that machines can process, interpret, and utilize them. Among the most important and commonly used Semantic Web technologies, we find (see Figure 2.4):

- **RDF (Resource Description Framework)**: RDF is a W3C specification that provides a standard model for data exchange on the web. Data in RDF is

represented as triples, which consist of a subject, a predicate, and an object. RDF allows for the creation of interconnected data networks, facilitating data integration.

- **RDFS (RDF Schema)**: RDFS is a set of classes and properties that extend the basic RDF vocabulary, providing mechanisms for the description and classification of resources.
- **SPARQL (SPARQL Protocol and RDF Query Language)**: SPARQL is the standard language for querying RDF-based databases. It allows users and developers to search for and manipulate data stored in RDF.
- **OWL (Web Ontology Language)**: OWL (Bechhofer *et al.* 2004) is an ontology language developed by the W3C to represent rich and complex information about things, groups of things and the relations between them. OWL builds upon RDF and provides structure and constraints to RDF. The semantics of RDF are primarily found in the semantics of OWL (in fact, most RDF constructs are included in OWL), but not all RDF features are included in all OWL variants.
- **JSON-LD (JSON for Linking Data)**: JSON-LD is a way to encode linked data using JSON. It allows developers to utilize the full capabilities of the Semantic Web (autodiscovery, data linking, etc.) in a format that is familiar and popular among web developers.
- **Ontologies**: They provide a formal representation of knowledge and a shared vocabulary to describe concepts. An ontology defines classes, properties, and relationships between them, creating a structured framework for understanding information. Commonly used ontology languages include OWL and RDFS.

Ontologies play a fundamental role in the Semantic Web by providing a formal structure and a shared vocabulary to describe concepts and relationships. This is one of the most common objectives in ontology development. For example, suppose multiple different websites contain medical information. Computer agents can extract and aggregate data from these sites if these websites share and publish the same underlying ontology of terms they use.

Moreover, ontologies facilitate the reuse of domain knowledge. Suppose a group of researchers develops an ontology in detail. In that case, others can reuse it for their domains, or if we need to construct a large ontology, we can integrate several existing ontologies that describe parts of the broad domain.

Key concepts and relevant relationships within the domain must be identified

to describe a particular domain using ontologies. The **classes** play a fundamental role in defining sets of individuals that share common characteristics. Additionally, we can construct class hierarchies using **subclasses**. **Data type properties** allow us to relate individuals to specific values, while object properties establish relationships between individuals. Individuals help us identify and differentiate elements within the domain, whether physical or conceptual.

By combining individuals, classes, and properties, we can create complex descriptions of sets of individuals, sets of literals, and relations between them. These descriptions take the form of **axioms**, statements that precisely and formally describe the domain we are modeling.

2.3.1 Reasoning in the Semantic Web

In the Semantic Web context, reasoning involves using rules and algorithms to infer new information from existing data. It is a way to discover new insights and connections between data not explicitly present in the original datasets. As seen, ontologies can represent complex relationships between entities and allow for the definition of constraints and logical rules. From a logical standpoint, classes in OWL are unary predicates (for instance, a "Cat" class represents the property of being a cat). In contrast, properties are binary predicates (for example, a "hasChild" property defines the relationship between a person and their child). Constraints are logical formulas established as axioms in these predicates, asserted as true in the domain of interest.

This representation is directly based on Description Logic. Description Logics are a family of knowledge representation languages that can be used to depict knowledge in a domain of application in a structured and formally comprehensible manner. Description Logics are decidable fragments of first-order logic, allowing reasoning over complex logical axioms about unary and binary predicates.

A reasoner can use these rules and constraints to infer new information. For example, if it's known that "all cats are mammals" and that "Raven is a cat," a reasoner can deduce that "Raven is a mammal," even if this information wasn't explicitly specified. Some examples of reasoners include Pellet (Sirin *et al.* 2007), HermiT (Shearer *et al.* 2008), and Racer (Haarslev *et al.* 2012). These reasoners can handle different subsets of OWL and have varying performance and scalability features.

Inference can also be carried out using the SWRL (Semantic Web Rule Language; Horrocks *et al.* 2005), combining logical rules with OWL ontologies. SWRL extends OWL by enabling the definition of rules using a syntax similar

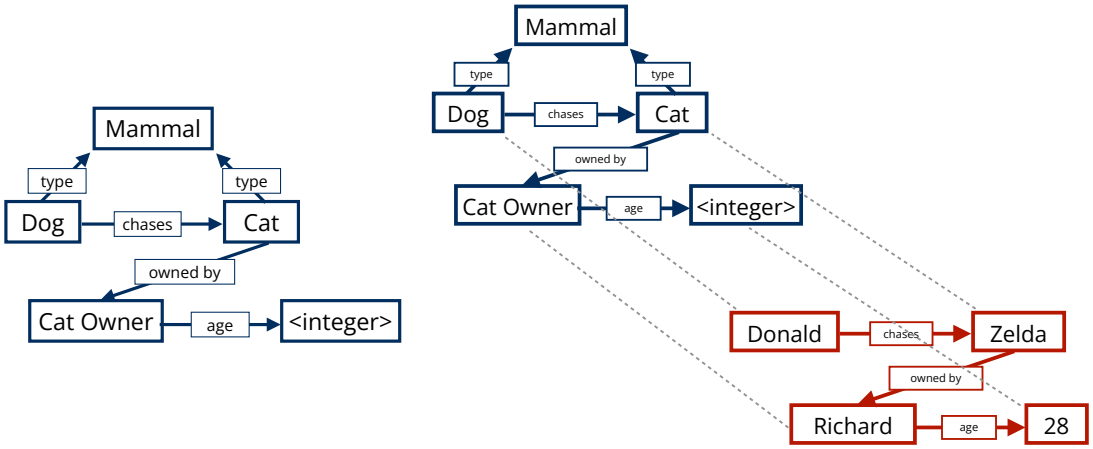

Figure 2.5: Ontologies (left) are reusable and indexed repositories of digital knowledge that define the types of elements in a domain of knowledge and their interrelationships and properties. Knowledge graphs (right) are indexed knowledge bases that can contain ontologies and real entities and their data. Source: own elaboration.

to production rules in programming. Rules in SWRL consist of a head and a body. The head specifies the new information that can be inferred, while the body sets the conditions under which the rule can be applied. For instance, one can define a SWRL rule: "If a person has a child, and that child is a student, then the person is a parent." This rule allows the inference of the parent-child relationship from the student-person relationship. Using rules in SWRL enhances the reasoning capabilities of the Semantic Web, allowing for more complex and powerful inferences. Reasoners that support SWRL can use these rules to derive new information and discover data connections more efficiently.

2.3.2 Knowledge Graphs

Knowledge Graphs represent an advanced way of organizing and structuring information, allowing a more profound and contextualized understanding of data. A knowledge graph can be described as an interconnected network of entities (nodes) and relationships (edges) that organize information intuitively and semantically coherently.

Building a knowledge graph typically involves the following: 1) gathering data from various sources, 2) processing this data to extract entities and relationships, and 3) representing these elements in a graph. Semantic Web technologies, such as RDF and OWL, play a crucial role in this process, providing the means to express and reason about data in the knowledge graph.

Although ontologies and knowledge graphs are terms often used in the same context and used to represent information in a structured form, they have some fundamental differences in purpose, structure, and use. The main difference between ontologies and knowledge graphs is their focus and application. Ontologies focus on defining a conceptual framework and a shared vocabulary for a domain, also called *TBox*, and are ideal for capturing and structuring high-level knowledge. Knowledge graphs, on the other hand, focus on representing and linking concrete data and facts about specific entities, or *ABox*, as shown in Figure 2.5.

In practice, ontologies and knowledge graphs are often used together. For instance, an ontology can be the foundation for constructing a knowledge graph, providing the classes and relationships used to represent data. In turn, knowledge graphs can enrich and expand ontologies with real-world data and facts. In this sense, ontologies and knowledge graphs complement each other, and both are fundamental elements in the management and use of semantic data.

2.3.3 Applications and Use of Semantic Web Technologies

Semantic Web technologies have found applications in various domains and sectors. These applications range from enriching web content and enhancing search capabilities to integrating enterprise data and building knowledge management systems. Some of the areas in which these technologies are making a significant impact include:

- **Open Data**: Open data are freely available to everyone. Semantic Web technologies, like RDF and SPARQL, represent and query these data, allowing its interconnection and reuse in various contexts.
- **Knowledge Management**: In knowledge management, Semantic Web technologies are employed to represent, organize, and access knowledge effectively. This might involve building ontologies to represent the knowledge of a particular domain or using SKOS to collect and share controlled vocabularies.
- **Data Integration**: Data integration involves combining data from different sources and transforming it into a coherent format. Semantic Web technologies, like RDF and OWL, facilitate this process by providing a common framework to represent and relate data.

Chapter 3

TITAN: A knowledge-based platform for Big Data workflow management

The adoption of semantics introduces a range of intriguing research challenges. These include improving the quality of data and analytical outcomes, ensuring the robust validation of analytical processes during both their design and implementation stages, and enhancing the traceability of results. Beyond merely providing explanations for results, there is also the imperative to furnish a "confidence level" that hinges on factors like data provenance and domain semantics (McClatchey *et al.* 2015). Specifically, Semantic Web technologies offer avenues to enrich data with domain-specific annotations and provide detailed descriptions of features and transformative actions executed by data analysis algorithms. This encompasses aspects like algorithmic parameters, input-output variables, and various behavioral dynamics.

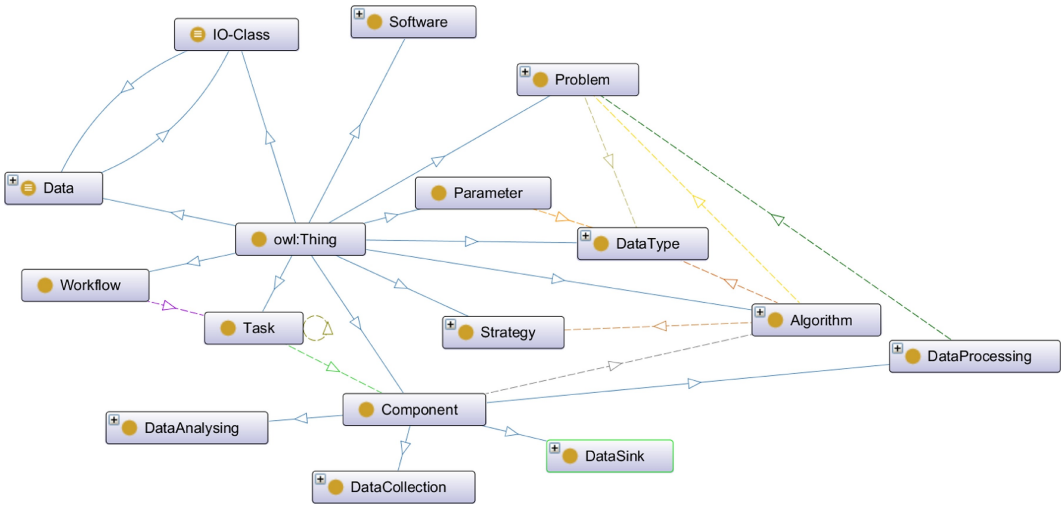

Figure 3.1: Overview of the BIGOWL ontology.

In this context, the BIGOWL ontology (Barba-González, García-Nieto, del Mar Roldán-García, *et al.* 2019) emerged from an ontology-driven approach de-

signed to facilitate knowledge management within Big Data analytic workflows. BIGOWL aligns with the prevailing trend in Big Data analysis, which emphasizes automating the selection of specific modeling techniques, such as deep learning or multi-objective methods through service composition (Siriweera *et al.* 2017; Akila *et al.* 2018). It encompasses a comprehensive set of concepts, attributes, and relationships derived from the Big Data and optimization ecosystem (see Figure 3.1), such as *Component*, *Task*, *Algorithm*, *Data*, and *Workflow*. Nonetheless, effectively harnessing BIGOWL in practical scenarios demanded significant effort from developers, including manual annotation of information and deploying workflows in the execution environment.

In this scenario, we present TITAN (SemanTics in bIg daTa ANalytics), a software platform enriched with semantic capabilities that streamline the design, annotation, development, testing, deployment, and execution of Big Data analytic workflows. TITAN is designed around the BIGOWL framework, providing the ontological framework that directs the semantic annotation of algorithmic components, data sources, operational constraints, and execution blueprints. As a result, TITAN utilizes semantics not only for workflow design, recommendation, and validation but also to give contextual relevance to the analyses conducted. TITAN distinguishes developer users from business users. Developers would design and semantically annotate new components and leverage these components for their analytics, allowing users to tap into its comprehensive semantic capabilities without delving into the complexities of defining data or component semantics.

3.1 Related Work

In this section, we summarise the main features of the related work with regards to the knowledge-based approach proposed here (see Table 3.1). These features consist of specifying whether the existing systems focus on their orientation to Big Data analysis streaming processing support, provide online version, programming language, use of semantics in different steps of the study, or the license of the framework. The main features taken into consideration are:

- ***Approach***. This column indicates wherever the approach is data driven (D) or task (T) driven.
- ***Big Data support***. The support for Big Data technologies has been taken into account by discriminating three possible cases: (N) Big Data technologies are not included, (Y*) Big Data technologies are included and (Y) focused only in Big Data technologies.
- ***Data Stream support***. The analysis of streaming data is a key topic in many Big Data workflows, which can be either supported (Y) or not supported (N).
- ***Semantic-Based approach***. The use of semantics can be present in several aspects: (D) design support, (V) workflow validation, (P) deployment support, (L) Linked Data provision. In other cases, (N) when there is no support.
- ***Open Source***. Open Source approaches enable long term maintenance of the systems, avoiding the limitation of a specific funding frame. There are three options: closed source (N), open source (Y) or unknown (U).
- ***Cost***. The cost of use can be: (F) free, (C) free community version and enterprise version underpayment, (E) only available after any payment or subscription method, or unknown (U).
- ***Languages Supported***. The programming languages supported for the development of new components is relevant to be able to engage as many developers as possible in the case of Open Source solutions.
- ***Platform***. This feature makes reference to the way the proposal can be used: (O) online version available, (S) stand-alone version available, (C) cloud support.

Table 3.1: Comparison of the main features of the platform with regards to other solutions.

Framework	Approach	Big Data	Data Stream	Semantic-Based approach	Open Source	Cost	Languages supported	Platform
VisTrails	T	N	N	N	Y	F	Python	S
YesWorkflow	T	N	N	D	Y	F	Java & Python	S
KNIME	T	Y*	Y	L	N	E	Java	SC
Airflow	T	Y*	N	N	Y	F	Python	SC
Taverna	T	N	Y	N	Y	F	Java	S
WINGS	T	N	N	DV	Y	F	Java	S
Pegasus	T	Y*	N	DV	Y	F	Java	S
MINT	T	N	N	DV	Y	F	Python	S
Makeflow	T	Y	N	D	Y	F	None	S
BioMOBY	T	N	N	V	Y	F	Java	S
ASC	T	Y	N	D	U	U	U	U
Pachyderm	D	Y*	Y	N	N	E	Go	SC
Nextflow	D	Y*	Y	N	Y	F	Python	S
VLAM-G	D	Y*	N	N	U	U	U	U
Karma2	D	N	N	D	U	U	U	U
TITAN	TD	Y*	Y	DVPL	Y	F	Multi-language	SOC

3.2 Metadata Framework

Our primary objective is to create a component-based platform where semantics are integral in representing various facets of Big Data analytic workflows. These facets encompass data, tasks, components, algorithms, and domain specific knowledge. Fundamentally, our platform facilitates the construction of intricate workflows through a sequential order of operations applied to input data. TITAN does not solely consist of Big Data components; it also incorporates conventional elements vital for constructing data analysis workflows.

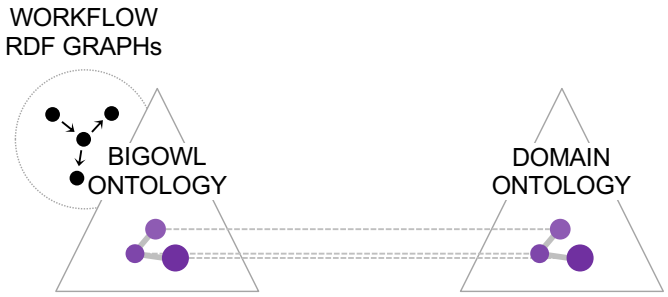

Figure 3.2: TITAN operates at three levels of abstraction: BIGOWL ontology, RDF graphs (i.e., individuals describing the workflows), and domain ontologies.

Explicit domain knowledge, coupled with understanding how each component transforms data, can lead to better workflow designs. We ensure this by formally describing each task using semantics. This guarantees component compatibility via semantic reasoning. Typically, this domain knowledge is extracted during problem analysis and manually embedded during algorithm design. Ideally, this knowledge manifests as rules that adapt the overall behavior of analytic algorithms. BIGOWL aligns with domain ontologies to further this integration, enhancing the workflow design process. This alignment equips TITAN with a structured representation of components segmented into four categories: data sources, data processing, data analysis, and data sinks. Semantics is meticulously articulated within TITAN through three layers: the BIGOWL ontology, RDF graphs (describing workflows), and domain ontologies, as depicted in Figure 3.2.

3.3 Software Architecture

TITAN conceptualizes a workflow as an interconnected ensemble of tasks, adhering to a modular and open-ended design framework. In this context, *tasks* represent specific instances of *components*, each defined by distinct values for their inputs, outputs, and associated parameters. This inherently flexible design strategy promotes component re-usability, enabling their incorporation into diverse workflows. This streamlines the process and enhances efficiency by avoiding the redundant creation of similar components for different workflows.

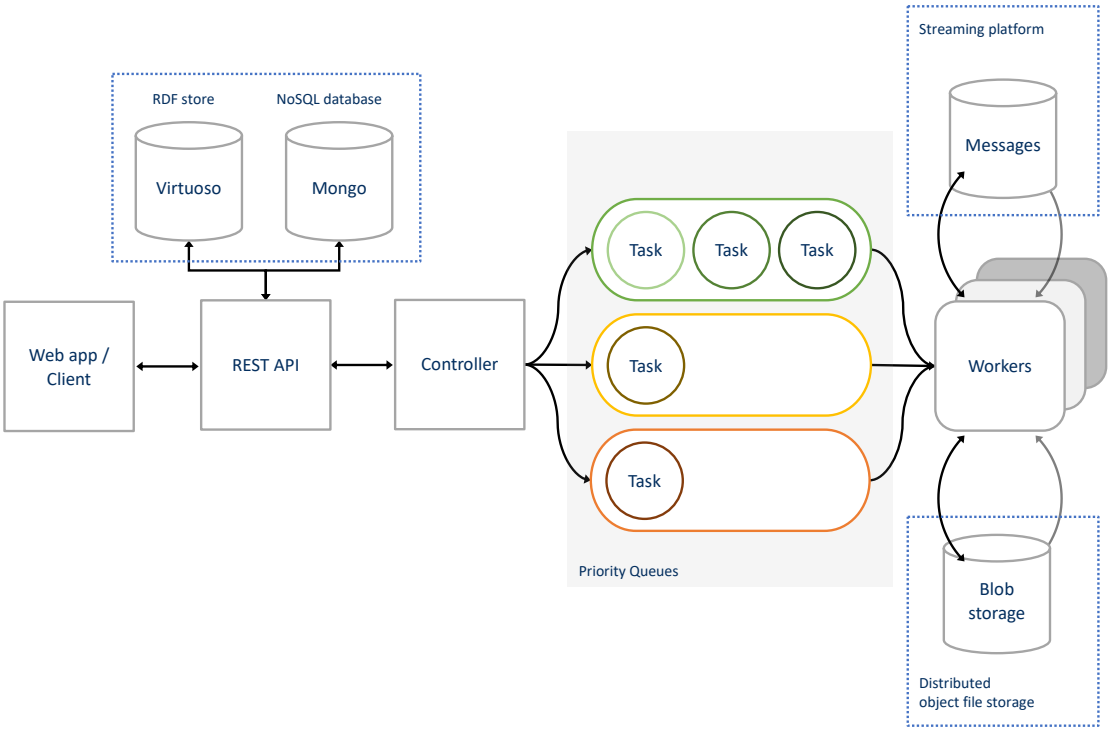

Figure 3.3: Core TITAN platform's architecture.

The current architecture of TITAN platform is depicted in Figure 3.3 and encompasses several key components:

- **Web application / Client**: This provides an interactive visual platform for users to navigate, design, and manage workflows. The components catalog is dynamically sourced from BIGOWL (maintained and stored in an RDF triple store, commonly referred to as RDF storage).
- **REST APIs**: Acts as an intermediary that facilitates communication between the front-end interface and the platform's backend infrastructure.

The primary purpose of these components is to provide the platform with essential functionalities. Examples of such functionalities include user authentication, data persistence, semantic annotation of components, and more. This also encompasses the visualization of the workflow's execution process and all functionalities related to data management.

- **Controller**: The orchestrator is responsible for streamlining and executing the workflows. In this setup, workflows initiated for execution are broken down or "flattened" into individual tasks. These tasks are then dispatched to a highly scalable and lightweight message broker.
- **Remote Object File Storage**: This storage solution houses the outputs generated by various components. It ensures that results are retained and can be accessed and used in subsequent steps or future workflows.
- **Streaming platform**: As the platform's communication backbone, the broker distributes messages among components. It ensures that tasks and processes are well-coordinated, and data flow is seamless.

3.4 Use case: Supervised Classification

As mentioned earlier, TITAN workflows operate as Directed Acyclic Graph (DAG) representations. In these representations, tasks are nodes, while the edges symbolize data and control flow dependencies between these tasks. TITAN workflows exist in two distinct versions: the *abstract workflow*, the user's conceptualization of the workflow, and the *concrete workflow*, the version that gets executed. In TITAN, the abstract workflow consists of various components and their interconnections. Once all these components and their parameters are set up, they become instantiated as independent, executable tasks via Kafka, linked together through Kafka topics.

Let us explore an example of a machine learning workflow constructed using the TITAN platform. The objective is to forecast a response variable based on exploratory observations. Specifically, the workflow aims to classify different species of the Iris flower using a dataset that includes 50 observations for each of the three species: *Setosa*, *Versicolour*, and *Virginica*.

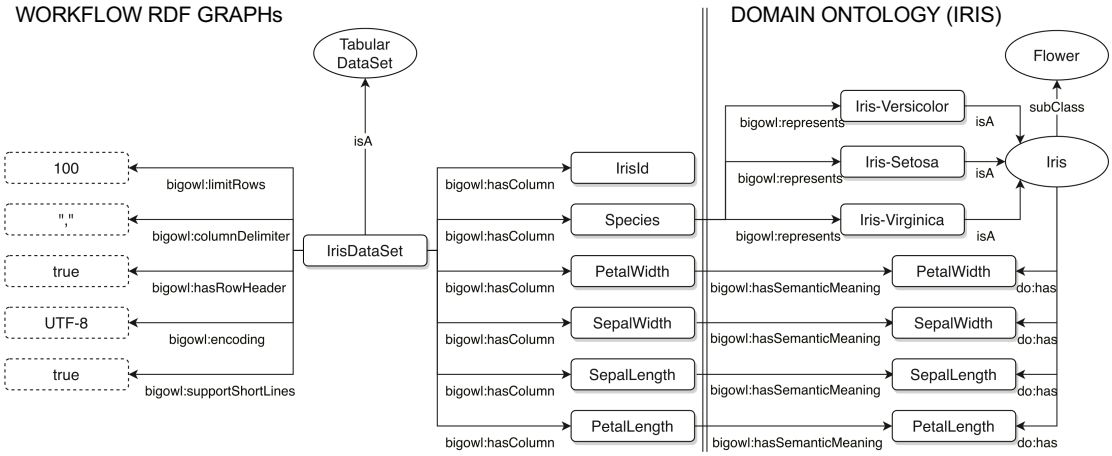

Figure 3.4: Definition of the Iris dataset using RDF graphs, incorporating both BIGOWL and Iris Domain ontologies. The prefixes `bigowl` and `do` denote properties originating from BIGOWL and the Domain ontology, respectively. Dotted boxes implies data properties within the ontology.

Figure 3.4 illustrates the definition of the Iris dataset via an RDF graph, leveraging two ontologies: BIGOWL and a domain-specific ontology. Properties of the Iris dataset are annotated using BIGOWL, and its columns are semantically linked through BIGOWL properties such as "represents" and "hasSemanticMeaning".

This workflow is generated with TITAN platform, as depicted in Figure 3.5,

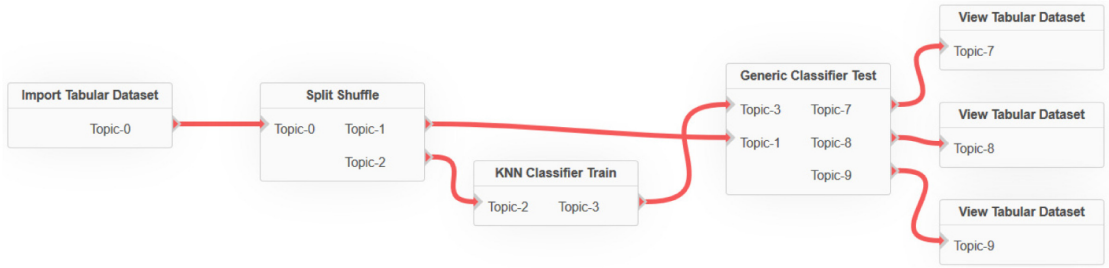

Figure 3.5: The Iris dataset classification workflow as visualized in the graphical tool. The initial step involves loading the dataset, followed by the second step, partitioning the data into training and testing sets. The training data is used to build a machine-learning model in the third step. The fourth step employs the trained model and the testing data to evaluate the classifier's predictive performance. Ultimately, key metrics generated from the model are highlighted.

and comprises the following components: (1) the dataset is loaded. Then, (2) it splits into train and test data. After that, (3) train data are used to train a machine learning model, and (4) test data and the model are used to test the classifier's prediction. Finally, (5) some useful metrics derived from the model are highlighted. We can annotate each of these steps semantically as distinct tasks, which are connected in a specific sequence:

- The first task involves an "Import Tabular Dataset" component that loads the Iris dataset. The output class for this component is `GenericDataSet`.
- The second task employs a "Split Shuffle" component to divide the dataset into training and testing sets. This component includes a parameter to set the ratio for the split, and takes and outputs a `GenericDataSet` class.
- The third task uses a "KNN Classifier Train" component, which accepts an input of type `GenericDataSet` and outputs a trained model of type `TrainModelKNN`. This component allows for several parameters to be set for tuning the model.
- The fourth task inputs a trained KNN model and a testing dataset (of type `GenericDataSet`) to validate the classifier. The output is validation results of type `ValidationResults`, achieved through the "Validate Model" component.
- The fifth task utilizes the `ValidationResults` from the prior task to visualize the metrics through a "Visualize Results" component.

```
SELECT DISTINCT
    (COUNT(?parameterTask) = COUNT(?parameterComp) as ?result)
WHERE {
    titan:WorkflowIris rdf:type dmop:Workflow .
    titan:WorkflowIris bigowl:hasTask ?task .
    ?task rdf:type bigowl:Task .
    ?task bigowl:hasComponent ?comp .
    OPTIONAL {
        ?task bigowl:hasParameter ?parameterTask .
        ?comp bigowl:hasParameter ?parameterComp .
        ?parameterTask bigowl:instanceParameterOf ?parameterComp .
    }
}
GROUP BY ?parameterTask ?parameterComp
```

Listing 1: SPARQL query evaluates whether all task parameters within the Iris workflow have been explicitly declared to ensure that each task has all its required parameters specified, thus guaranteeing that no critical information is missing before workflow execution.

```
SELECT DISTINCT ?task1 ?task2
WHERE {
    titan:WorkflowIris rdf:type dmop:Workflow .
    titan:WorkflowIris bigowl:hasTask ?task1 .
    titan:WorkflowIris bigowl:hasTask ?task2 .
    ?task1 rdf:type bigowl:Task .
    ?task2 rdf:type bigowl:Task .
    ?task1 bigowl:connectedTo ?task2 .
    ?task1 bigowl:hasComponent ?comp1 .
    ?task1 bigowl:hasComponent ?comp2 .
    ?comp1 bigowl:specifiesOutputClass ?outputC .
    ?comp2 bigowl:specifiesInputClass ?inputC .
    ?task1 bigowl:specifiesOutputClass ?outputTask .
    ?task2 bigowl:specifiesOutputClass ?inputTask .
    ?inputC rdf:type ?class .
    ?outputC rdf:type ?classOC .
    ?outputTask rdf:type ?classOT .
    ?inputTask rdf:type ?classIT .
    ?classOC rdfs:subClassOf* ?class .
    ?classOT rdfs:subClassOf* ?class .
    ?classIT rdfs:subClassOf* ?class .
    FILTER (?class!=owl:NamedIndividual)
}
GROUP BY ?task1 ?task2
```

Listing 2: This SPARQL query aims to validate that the tasks and their sequence are semantically compatible and correctly defined.

For implementing this workflow, Listing 1 and Listing 2 showcase the SPARQL queries that the semantic model uses to verify the workflow's coherence and task compatibility. These queries are auto-generated by the TITAN platform. The first query evaluates whether all task parameters in the Iris workflow have been declared. The second query assesses the overall correctness of the workflow.

Chapter 4

NORA: A Scalable Reasoning System

This chapter presents NORA, our proposed system designed to address the challenges associated with scalable reasoning in modern Semantic Web applications (Objective O.2). As the volume of data on the web grows exponentially, there is an increasing need for reasoning systems that are not only scalable but also robust and efficient:

Robustness refers to the system's ability to manage failure scenarios effectively, ensuring that no data is lost and availability is not compromised even in adverse conditions. This aspect of robustness is critical for maintaining the integrity and reliability of the system under various operational stresses.

Efficiency, particularly in terms of performance, is another key attribute of NORA. It denotes the system's ability to process large volumes of data utilizing resources optimally.

Scalability, the third pivotal characteristic of NORA, pertains to its capability to handle vast and growing amounts of structured data without a decline in performance. This means that as the dataset grows - a common occurrence in the Big Data landscape of today's web - the system can adapt and continue to operate effectively without being overwhelmed. This is in contrast to traditional reasoning systems, which, while effective in their own domain, face scalability issues when confronted with the Big Data landscape that characterizes today's web.

In this context, NORA emerges as a novel approach, leveraging state-of-the-art technologies and architectures to ensure that reasoning remains feasible even as data scales. The underlying principle of NORA is combining the strengths of distributed computing platforms, like Apache Spark, with the flexibility and scalability of NoSQL databases, such as Apache Cassandra. By doing so, NORA aims to bridge the gap between traditional reasoning systems and the evolving needs of modern web applications.

This chapter delves deep into NORA's design principles, architecture, and

implementation details. We will also present empirical results from our experiments with NORA, demonstrating its performance capabilities and scalability features when handling real-world datasets. Through these experiments, we aim to establish NORA's credentials as a viable solution for scalable reasoning in the Semantic Web age.

4.1 Related Work

Reasoning in the Semantic Web can be extremely valuable for tasks like data validation (checking if data meets certain constraints), classification (grouping similar entities), and data integration (finding connections between data from different sources). However, it is also computationally intensive, especially for large datasets and intricate ontologies. Managing vast amounts of data in OWL is an increasingly essential challenge in many real-world application domains. There is also a need to contextualize, integrate, and make these data publicly available in a canonical representation for the scientific community. This results in not only more complex but also much larger ontologies.

Most existing reasoners have memory and storage limitations, making them less efficient for scalable reasoning over extensive datasets. Another aspect to consider is the scalability of reasoners in distributed environments. Since scalable reasoning involves processing large amounts of data and performing complex computations, distributing the workload across multiple processing nodes is necessary. This presents additional challenges, like coordinating distributed computing and efficient communication between nodes.

To overcome these issues, numerous proposals emerged that used relational databases for implementing OWL reasoning. These systems offered ontology, data persistence, and some level of scalability but often compromised on completeness. A comprehensive overview of these reasoners can be found in del Mar Roldan-Garcia and J. Aldana-Montes 2006. Noteworthy contributions in this space include DBOWL (Roldan-Garcia and J. F. Aldana-Montes 2008) and Stardog[1], which were significant strides toward creating an OWL-DL complete reasoner using relational database technology.

However, centralized systems struggle to manage the vast expanse of Web data, Big Data, and the ever-increasing volume of published Linked Data. There is a noticeable gap in research exploring alternative database paradigms, such as NoSQL databases, for constructing scalable RDF triple stores or OWL reasoners. Jena-Hbase and H2RDF employ Apache HBase, a NoSQL column family store, for RDF data storage. CumulusRDF, meanwhile, adopts a cloud-based design, leveraging Apache Cassandra for RDF triple storage. As detailed in Roldan-Garcia and J. F. Aldana-Montes 2008, there are efforts to materialize OWL ontologies in NoSQL databases, with Cassandra hosting both ontologies and their respective class and property hierarchies.

More recent literature, such as Antoniou *et al.* 2018, provides insight into

[1] In URL: `https://www.stardog.com/`

large-scale reasoning on the Web of data. The trajectory of OWL reasoners aligns with the proliferation of Big Data technologies, showcased by platforms like the Apache Hadoop Framework[2] and Apache Spark.

WebPIE employs "map and reduce" functions on RDF data, capitalizing on the Hadoop Distributed File System (HDFS). Yet, it overlooks data partitioning, treating datasets holistically, leading to prolonged data read and process times. This methodology also poses scaling challenges with vast datasets. NORA, on the other hand, uses Spark to leverage data partitioning and access, making it more efficient with large datasets.

Cichlid (Gu *et al.* 2015) is an RDFS and OWL reasoning system that uses Spark for data partitioning and access. Similarly, SPOWL (Liu and McBrien 2017) is a Spark-driven OWL 2 reasoner that employs HDFS for data retention. Although the synergy between HDFS and Spark is well-matched for this objective, the absence of record-level indexing often leads to diminished query response times.

Likewise, SPOWL (Liu and McBrien 2017) is a Spark-based OWL 2 reasoner. SPOWL uses HDFS to persist data. While HDFS, together with Spark, is well suited for this task, it lacks record-level indexing, which is often bound to give lower query response times. Our proposal, by using Apache Cassandra, which provides tunable consistency and allows for record-level indexing, provides a higher query response time.

In Mohamed. *et al.* 2021, the authors proposed the SANSA framework. This system employs several scalable reasoning strategies over large OWL datasets using Spark. However, it relies on in-memory data storage, which can be a bottleneck when data size surpasses available memory. Additionally, the transient nature of in-memory data raises data loss concerns, necessitating recovery efforts. On the other hand, NORA uses Apache Cassandra for data storage, a NoSQL database renowned for its capability to handle large datasets. Apache Cassandra distributes data across many nodes, thereby mitigating the memory limitation. Furthermore, data stored in memory is transient and can be lost in the event of a failure. This necessitates additional steps for data recovery. NORA's use of Cassandra offers data persistence, ensuring data is safely stored and easily recoverable, which provides an extra layer of security and robustness.

Previous works highlight Spark's popularity in distributed reasoning systems. However, there's still a need to research and develop approaches and techniques that address these memory, storage, and scalability limitations to enable more

[2]Available at `https://hadoop.apache.org/`

efficient reasoning. However, to the best of our knowledge, the approach presented in this chapter is the first to successfully combine a NoSQL database with Spark in order to implement a scalable OWL reasoner (Table 4.1). The benefits of using a NoSQL database, such as Apache Cassandra, include improved performance, horizontal scalability, and flexibility in handling large datasets. These advantages, coupled with the power of the Spark framework, allow our system to efficiently process and reason over massive knowledge bases, addressing some of the limitations faced by existing solutions.

Table 4.1: Popular reasoning systems with Big Data support.

Name	Version	Language	Fragment	Storage layer	Processing	Parallel	Distributed
WebPIE	1.1.1	Java	RDFS/OWL	HDFS	MapReduce	Y	Y
Cichlid	-	Scala	RDFS/OWL	In Memory	Apache Spark	Y	-
SPOWL	-	Java	OWL 2 RL	HDFS	Apache Spark	-	-
SANSA framework	0.7.1	Scala	RDF/OWL	HDFS	Apache Spark	Y	-
DBOWL	0.0.1	Java	OWL-DL	Oracle	SQL views	Y	Y
NORA	1.0.0	Java	OWL 2	Cassandra	Apache Spark	Y	Y

4.2 System Architecture

The core NORA's architecture consists of two steps: an OWL parser (including a bulk loader for ontology instances) and an OWL reasoning engine. These services work together effectively to achieve scalable reasoning. The process is summarised in Figure 4.1. In the first step, data obtained from any Knowledge Base (KB), represented as an OWL ontology, is materialised in a Cassandra database. A description logic (DL) reasoner is used to pre-compute complete subsumption relations regarding the TBox from an OWL file (the starting point). We use the OWL API, a Java interface and reference implementation for the OWL language, to read the input OWL ontologies. Pellet, a well-known OWL 2 DL reasoner, is used to classify on the TBox. By combining these tools, we obtain several taxonomies, such as the class/subclass and property/subproperty hierarchies, but no newly derived knowledge is inferred from the ABox. Subsequently, a database schema is created to materialise all this information (Roldan-Garcia and J. F. Aldana-Montes 2008). As a result, the reasoned TBox is stored in Cassandra together with the explicit instances described in the Knowledge Base.

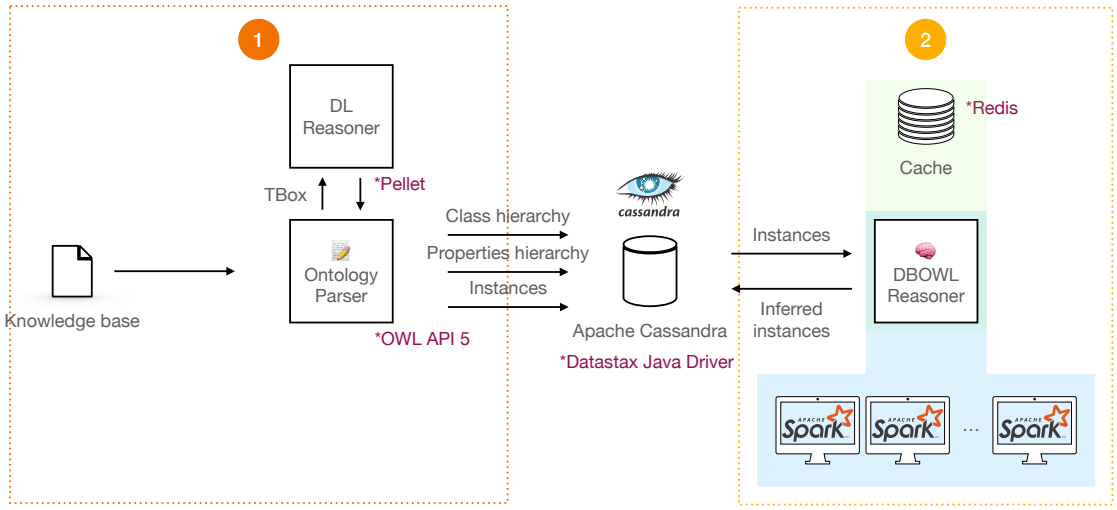

Figure 4.1: The high-level architecture of NORA.

In the second step, the reasoning in the ABox occurs. We create an RDD from a specific table to read data from the database. This abstraction allows us to manipulate the data spread across many machines in a distributed fashion. The NORA reasoner engine powered by Apache Spark evaluates the inference rules, expressed as Spark jobs, by accessing the database with materialised data. The inference rules are applied repeatedly until no further inferred statements

are produced. Under the hood, an external cache, Redis[3] in our case, is used to speed up this operation by loading information from the database into the cache for fast lookup operations.

[3]In URL: `https://redis.io`

4.3 Data materialisation

NORA persists OWL data in a NoSQL database, Apache Cassandra. The idea of working with NoSQL databases for storing large volumes of RDF-like data has been explored in the past (Ladwig and Harth 2011; Curé *et al.* 2012; Michel *et al.* 2019; Reyes-Álvarez *et al.* 2018; Roldan-Garcia and J. F. Aldana-Montes 2008). We have chosen Apache Cassandra over other software implementations of NoSQL databases for several reasons. None of these reasons is unique to Cassandra, but it is the only system that incorporates many of them. For example, like most NoSQL databases, Cassandra is massively scalable by simply increasing the number of nodes. Cassandra goes beyond and has a masterless architecture, where all nodes are equally important. This fact directly contributes to the cluster's robustness, which is of particular interest in Big Data applications, and implies that multiple nodes can fail without impacting the global availability of the system. Furthermore, Cassandra is also designed for heavy writes, which are unaffected by large volumes of data. Performance-wise, the Cassandra Query Language (CQL) also defines prepared statements, i.e., statements that are repeated multiple times can be pre-parsed, validated once and executed with a minimal impact on performance.

All of these benefits are not without concessions. NoSQL databases, in general, and Cassandra, in particular, lack relational features such as joins and other join-like operators. Dropping these features allows Cassandra to scale across multiple nodes more easily. Some solutions exist to overcome this barrier by enabling joins efficiently over such databases, which will be highlighted later.

Many concepts in Cassandra are closely related to its relational counterpart. In Cassandra, a database is known as a keyspace. Keyspaces contain one or more column families (tables), which organise data into rows. Our proposed optimised schema is based on our previous work as described in Roldan-Garcia and J. F. Aldana-Montes 2008, but with some refinements to ease data management. We use a compound primary key consisting of a partition key (responsible for determining which node stores the row) and a clustering key (accountable for data sorting within the partition) to allow multiple rows with the same identifier. The combination of partition and clustering key(s) enables performant reads/writes by distributing data efficiently across the cluster.

In our implementation, we have addressed a comprehensive OWL fragment, including many essential OWL constructs, e.g., intersectionOf, unionOf, and complementOf, and support for cardinality restrictions (such as minCardinality and maxCardinality). These constructs are stored in their corresponding column families within Cassandra. We defined 29 column families in Cassandra to store

all the OWL data. The column families are:

ClassIndividuals	*IsComplementOf*	*IsDisjointWith*
IsEquivalentToAll	*IsEquivalentToClass*	*IsEquivalentToIntersection*
IsEquivalentToMaxCardinality	*IsEquivalentToMinCardinality*	*IsEquivalentToSome*
IsEquivalentToUnion	*IsFunctionalProperty*	*IsInverseFunctionalProperty*
IsObjectPropertyDomain	*IsObjectPropertyEquivalentTo*	*IsObjectPropertyRange*
IsSameAs	*IsSubclassOfAll*	*IsSubclassOfClass*
IsSubclassOfComplementTo	*IsSubclassOfIntersection*	*IsSubclassOfMaxCardinality*
IsSubclassOfMinCardinality	*IsSubclassOfSome*	*IsSubclassOfUnion*
IsSubPropertyOf	*IsSuperClassOf*	*PropIndividuals*

For example, the *ClassIndividuals* column family materialises the individuals w.r.t. its class in the ontology (see Table 4.2). In this case, the partition key is *cls* (the class to which the instance belongs). Thus, all the instances of the previous class are stored in the cluster node, making the operation of retrieving instances for each class more efficient. The clustering key is *num* (the order in which this instance has been found in storing it). This order is critical in the process of discovering new instances in the reasoning process. For nested class descriptions defined as the intersection or union of two or more descriptions, we generate a random internal identifier and use it as a partition key in the column family. To illustrate the whole process, let us describe the following set of axioms:

$$\text{Tbox} = \text{WomanCollege} \equiv \text{College} \sqcap \forall \text{hasStudent}.\neg\text{Man}$$

$$\text{Abox} = \text{WommanCollege}(\text{college1}), \text{College}(\text{college2}), \text{hasStudent}(\text{college2,professor1})$$

cls	num	individual
WomanCollege	1	college1
College	1	college2

ClassIndividuals

cls	num	domain	range
hasStudent	1	college2	professor1

PropIndividuals

cls	num	ind1	ind2
WomanCollege	1	college	*_uid1*

IsEquivalentToIntersection

cls	num	prop	range
_uid1	1	hasStudent	*_uid2*

IsEquivalentToAll

cls	num	complement
_uid2	1	Man

IsComplementOf

Figure 4.2: Materialised axioms from Example 1. All column families are represented.

Figure 4.2 shows how the former Tbox and Abox axioms are materialised in the database. We have two instances of class WomanCollege and College, namely college1 and college2. The property hasStudent is also stored in the

Table 4.2: Proposed materialisation schema for storing Tbox and Abox axioms. For the sake of simplicity, only some axioms are shown.

Axiom	**Column family name**	**Schema**		
		column name	**type**	**datatype**
Class Axioms				
$A \sqsubseteq B$	IsSubclassOfClass	*cls*	partition key	*text*
		num	clustering key	*int*
		supclass	regular	*text*
$A \equiv B$	IsEquivalentToClass	*cls*	partition key	*text*
		num	clustering key	*int*
		equiv	regular	*text*
$A \sqsubseteq \neg B$	IsDisjointWith	*cls*	partition key	*text*
		num	clustering key	*int*
		ind1	regular	*text*
$A \equiv \neg B$	IsComplementOf	*cls*	partition key	*text*
		num	clustering key	*int*
		complement	regular	*text*
Abox Axioms				
$A(a)$	ClassIndividuals	*cls*	partition key	*text*
		num	clustering key	*int*
		individual	regular	*text*
$P(a, b)$	PropIndividuals	*prop*	partition key	*text*
		num	clustering key	*int*
		domain	regular	*text*
		range	regular	*text*
$a_1 \equiv a_2$	IsSameAs	*ind*	partition key	*text*
		num	clustering key	*int*
		same	regular	*text*

proper column family. To keep the class description in the Tbox, we generate random identifiers (_uid1 and _uid2) and use them as partition keys in their corresponding column family.

4.4 Reasoning strategy

Roldan-Garcia and J. F. Aldana-Montes 2008 proposed the storage of KBs in Cassandra, but the reasoning was not provided. Thus, there was a limitation as the NoSQL database offers efficient access to the KB, but only for explicit knowledge. NORA's reasoning engine implements ABox reasoning by translating each inference rule as an Apache Spark expression. The reasoning engine evaluates these Spark programmes iteratively until no new inferred knowledge is derived. The Spark connector for Cassandra can read data (columns and rows) from a keyspace and run standard Resilient Distributed Dataset (referred to as an RDD) methods to interact with Cassandra directly. RDD transformation includes *map*, *sort* and *join* operations which are not available in Cassandra natively. Listing 3 shows how a view of a column family can be loaded by using Spark and the DataStax Spark Cassandra Connector. Each row is converted to a serializable object in Java, and a representation of key-row pairs is returned.

```
JavaPairRDD<String, ClassIndividuals.Row> classIndividualsRDD = javaFunctions(spark)
    .cassandraTable(connection.getDatabaseName(), "classindividuals",
            CassandraJavaUtil.mapRowTo(ClassIndividuals.Row.class))
    .keyBy((Function<ClassIndividuals.Row, String>) ClassIndividuals.Row::getCls);

JavaPairRDD<String, IsSubclassOfClass.Row> isSubclassOfClassRDD = javaFunctions(spark)
    .cassandraTable(connection.getDatabaseName(), "issubclassofclass",
            CassandraJavaUtil.mapRowTo(IsSubclassOfClass.Row.class))
    .keyBy((Function<IsSubclassOfClass.Row, String>) IsSubclassOfClass.Row::getCls);
```

Listing 3: Read Cassandra column family from Java with Apache Spark.

TBox axioms can be rewritten in terms of join expressions which Spark can handle. We can join multiple column families using Spark's RDD API by selecting rows matching keys in both relations. We can apply multiple joins over the same column family and return only the final result to the driver program (lazy evaluation). NORA includes over 29 Spark programmes to perform the reasoning, available at the GitHub repository[4]. In order to illustrate how they operate, we will describe and show an example of two of them.

In Listing 4 we use a join operation to combine *ClassIndividuals* and *IsSubclassOfClass* column families to infer new sub-classes for the individuals in the keyspace. Figure 4.3 shows an example of this process. In this example, our Knowledge Base declares that Airport is a sub-class of Infrastructure. We also know that Madrid Barajas, Paris City Airport and New York Airport are instances of Airport and that New York Airport is an Infrastructure. If we join both column families, our reasoning algorithm will infer that Madrid Barajas and Paris City

[4]In URL: `https://github.com/benhid/nora`

```
JavaPairRDD<String, String> firstJoinRDD = isSubclassOfClassRDD
    .join(classIndividualsRDD)
    .mapToPair(tuple -> {
        Tuple2<IsSubclassOfClass.Row, ClassIndividuals.Row> secondOperand = tuple._2();

        IsSubclassOfClass.Row isSubclassOfClassRow = secondOperand._1();
        ClassIndividuals.Row classIndividualsRow = secondOperand._2();

        return new Tuple2<>(isSubclassOfClassRow.getSupclass(), classIndividualsRow.getIndividual());
    });
```

Listing 4: Reasoning algorithm for inferring sub-classes of classes.

```
JavaPairRDD<String, Tuple3<IsSubclassOfAll.Row, String, String>> firstJoinRDD = isSubclassOfAllRDD
    .join(classIndividualsRDD)
    .mapToPair(tuple -> {
        String cls = tuple._1();

        Tuple2<IsSubclassOfAll.Row, String> secondOperand = tuple._2();
        String individual = secondOperand._2();
        IsSubclassOfAll.Row isSubclassOfAll = secondOperand._1();

        return new Tuple2<>(isSubclassOfAll.getProp(), new Tuple3<>(isSubclassOfAll, individual, cls));
    });

JavaPairRDD<String, String> secondJoinRDD = firstJoinRDD
    .join(propIndividualsRDD)
    .mapToPair(Tuple2::_2)
    // Filters out rows where individual == domain
    .filter(tuple -> {
        Tuple3<IsSubclassOfAll.Row, String, String> firstOperand = tuple._1();
        String individual = firstOperand._2();

        PropIndividuals.Row propIndividualsRow = tuple._2();

        return individual.equals(propIndividualsRow.getDomain());
    })
    // Transforms to <range, range>
    .mapToPair(tuple -> {
        Tuple3<IsSubclassOfAll.Row, String, String> firstOperand = tuple._1();
        IsSubclassOfAll.Row isSubclassOfAllRow = firstOperand._1();

        PropIndividuals.Row propIndividualsRow = tuple._2();

        return new Tuple2<>(isSubclassOfAllRow.getRange(), propIndividualsRow.getRange());
    });
```

Listing 5: Reasoning algorithm for inferring subclasses.

Airport are sub-classes of Infrastructure as well. New York Airport being sub-class of Infrastructure is also inferred, but it is discharged as no new knowledge is produced in this step.

In many cases, reasoning is not as straightforward as simply loading and joining various column families. For example, the reasoning of *owl:subClassOf* property involves several column families, namely *IsSubclassOfAll*, *ClassIndividuals* and *PropIndividuals*. This process is shown in Listing 5. First, RDDs are

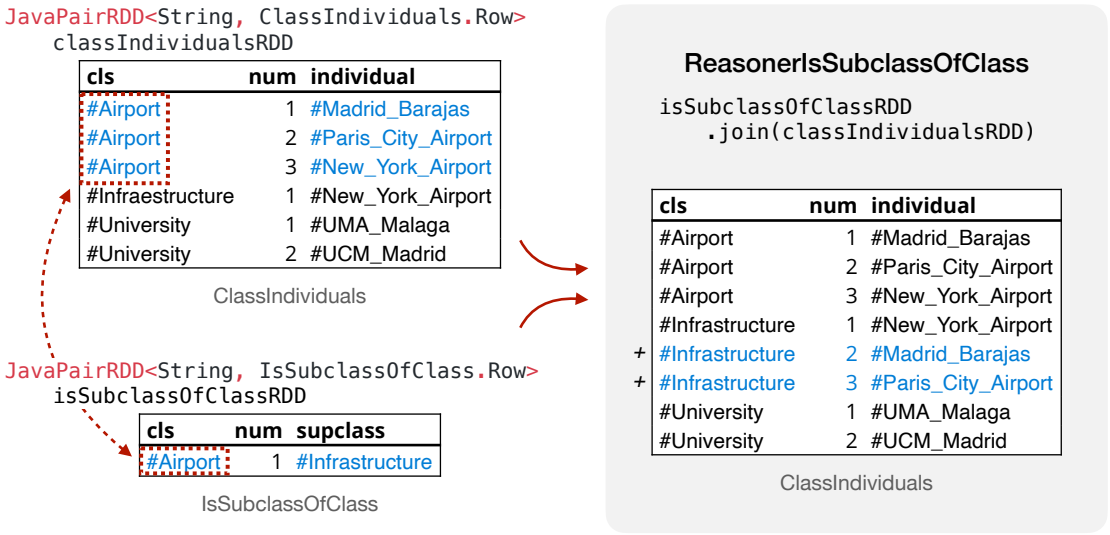

Figure 4.3: Join operation between *ClassIndividuals* and *IsSubclassOfClass* column families. Matching rows are classified as new individuals in the Knowledge Base and inserted in the *ClassIndividuals* column family if they do not exist yet.

created for each of these column families. Then, *IsSubclassOfAll* and *ClassIndividuals* are joined (first join), and an RDD is returned. This RDD, together with the *PropIndividuals* RDD, is joined (second join) using the property (from *IsSubclassOfAll* and *PropIndividuals* column families) as matching key. After that, the RDD is filtered, as we are only interested in those rows in which the individual (from *ClassIndividuals* column family) is the same as the domain (from *PropIndividuals* column family). Finally, the RDD is transformed, and a tuple of ranges (from *IsSubclassOfAll* and *PropIndividuals* column families) is returned. Figure 4.4 shows an example of this process. In this case, the Knowledge Base describes Humans, where a Human's parent is also a Human. For example, Fran, a Human, has a parent, María. Using this explicit knowledge, we can infer that María is a Human by joining the three column families above and filtering out the results.

To avoid redundancy in the database (i.e. duplicate entries), when a new inference is found, we need to determine whether the individual is already stored in the column family. The lookup is performed by querying a fast in-memory key-value store, which is pre-populated with the individuals from the KB at the beginning of the reasoning process. This lookup helps to reduce the number of database read operations, potentially avoiding network timeouts or other latency-related issues. If the individual is found in the cache, the inference is already present in the database, and we can skip the insertion: no further actions are required. Otherwise, it is added to the cache and a counter of the number of

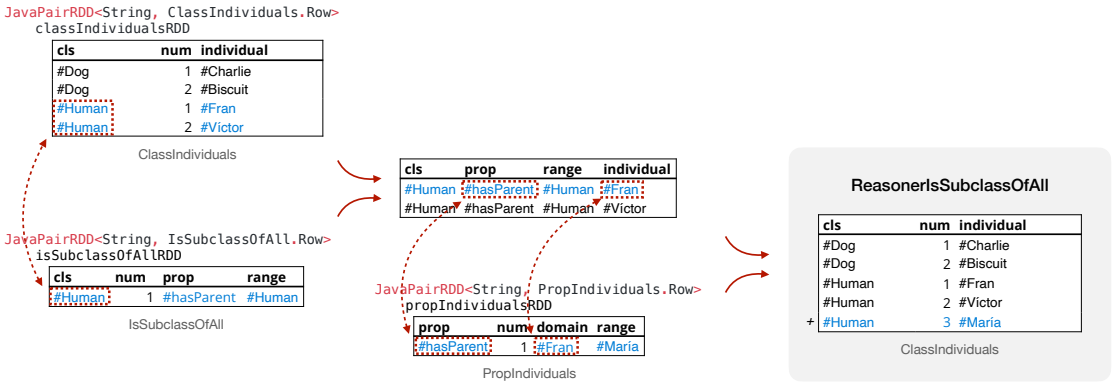

Figure 4.4: Join operation between *ClassIndividuals*, *IsSubclassOfAll*, and *PropIndividuals* column families to infer new individuals belonging to the Human class.

individuals belonging to the class is incremented. This value is used to perform the insertion.

4.5 Use cases

In this section, we present several use cases to illustrate the use of our proposal and show examples of the ABox reasoning involved. For evaluating the reasoner, we focus on assessing its performance and ability to provide correct inferences by employing two benchmarks. The first one is the well-known Lehigh University Benchmark for benchmark tests (Guo *et al.* 2005). The second one entails a more expressive scenario using the University Ontology Benchmark (Ma *et al.* 2006). These benchmarks help us to better understand the system's efficiency and its potential to generate accurate results under varying conditions. We performed our experiments 25 times and reported the mean value. To this end, we used a computing Spark cluster of 7 nodes (a master node and 6 slave nodes), each equipped with 100 GB RAM and 8 cores, running Ubuntu 16 and Apache Spark 3.3.0. Apache Cassandra database has been deployed in a cluster composed of 6 nodes, using a virtualized environment with Docker. Each node has 32 GB of RAM and runs Cassandra 4.0.4.

4.5.1 The Univ-bench benchmark

This first use case has been developed to show the scalability of the reasoning process using NORA. The Lehigh University Benchmark (LUBM) is a benchmark ontology describing an academic domain (universities, departments, and activities) that facilitates the evaluation of reasoning algorithms. A companion tool generates synthetic OWL data with different ABox sizes. Using this tool, we generated 80 universities, yielding approximately 8 million triples, to evaluate NORA's performance and efficiency in reasoning, using a range of different cluster configurations. NORA was able to generate over 4 million new *ClassIndividuals* and more than 1.5 million *PropIndividuals* as a result of the reasoning process.

Figure 4.5 (A) depicts the time used by NORA for reasoning materialisation with caching within the context of the LUBM ontology. Our initial findings suggest that that the running time obtained decreases along with the number of cores used. We find this a compelling indication of NORA's capacity to handle increasing workloads efficiently. It is important to note that our efforts in optimising the system's code are ongoing, and hence, we anticipate achieving even more impressive results in subsequent evaluations.

In Figure 4.5 (B), we have taken a detailed look at the time taken for inference and the time NORA requires to carry out CQL operations (database interactions) with an increasing number of universities, while maintaining a fixed clus-

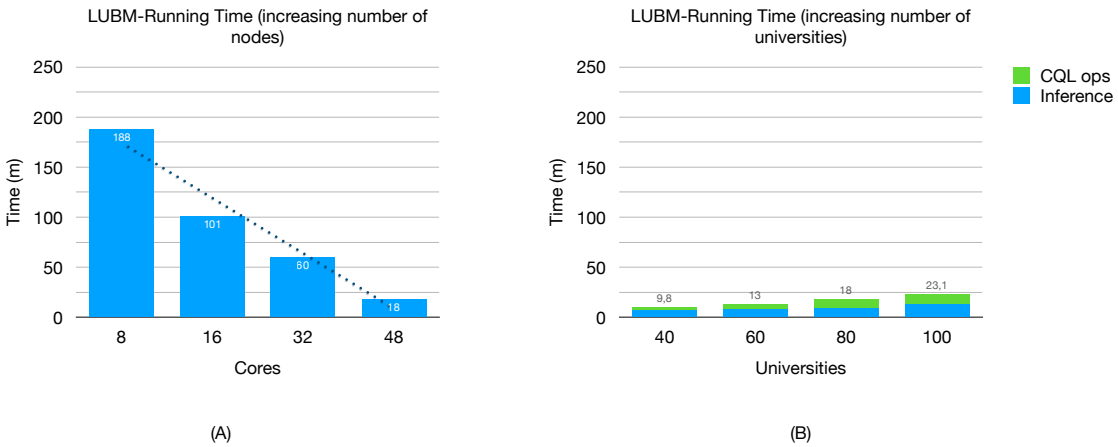

Figure 4.5: Running time (in minutes) spent in reasoning with different configurations.

ter configuration of 48 cores. As the volume of data, in this case represented by an increasing number of universities, expands, NORA demonstrates a robust ability to effectively manage and process this data without a significant increase in execution time. To maintain system stability and prevent potential issues arising from overload, we deliberately limit the number of concurrent write operations directed towards the database. A careful tuning of the database can streamline these interactions, thereby reducing the execution time. Furthermore, we anticipate that the removal of the limit on concurrent write operations would facilitate more rapid data processing, albeit potentially necessitating higher hardware specifications.

4.5.2 The University Ontology Benchmark

The second use case has been performed to show how NORA can perform the most common reasoning tasks. The University Ontology Benchmark (UOBM) is a benchmark ontology derived from LUBM that comes in two versions: a more expressive OWL DL (UOBM DL) and a less expressive OWL Lite (UOBM Lite) version. We used the latter, covering all language constructs of OWL Lite, for evaluation. Randomly generated test data with pre-computed correct query results were used to evaluate the completeness and soundness of the inference. The queries are tailored for this experiment and respond to different purposes. For example, Q_1 involves a simple conjunction (all undergraduate students who take a specific course), whereas Q_6 involves an inverse property (see below). The final data set contains over 210.000 statements broken up into 20 files, each representing a department in the university:

$\boldsymbol{Q_1}$ Find all undergraduate students who take course http://www.Department0.University0.edu/Course0

Q_2 Find out all employees
Domain(worksFor,Employee), <a worksFor b> → <a rdf:type Employee>
Domain(worksFor,Employee), researchAssistant ⊑ ∃worksFor.ResearchGroup → researchAssistant ⊑ Employee

Q_3 Find out all students of http://www.Department0.University0.edu
Range(takeCourse,Student), GraduateStudent ⊑ ≥1 takeCourse → GraduteStudent ⊑ Student

Q_4 All the publications by faculty of http://www.Department0.University0.edu
SubClass: Faculty = FullProfessor ⊔ AssociateProfessor ⊔ ... ⊔ ClericStaff, Publication = Article ⊔ ... ⊔ Journal

Q_6 All alumni of http://www.University0.edu
Inverse(hasAlumni,hasDegreeFrom), <a hasDegreeFrom b> → <b hasAlumnus a>

Q_{12} All students who take course taught by http://www.Department0.University0.edu/FullProfessor0

GraduateStudent ≡ ∀takesCourse.GraduateCourse ⊓ ≥1.takesCourse, Domain(takesCourse,Student) → Student ⊒ GraduteStudent

Experimental results show that our proposal is sound, i.e., the precision is 1. Furthermore, we could also answer all queries correctly. Figure 4.6 shows the execution time for running the queries in a single node using different numbers of universities. The data points in the graph showcases the trend in execution time as the volume of the data increases.

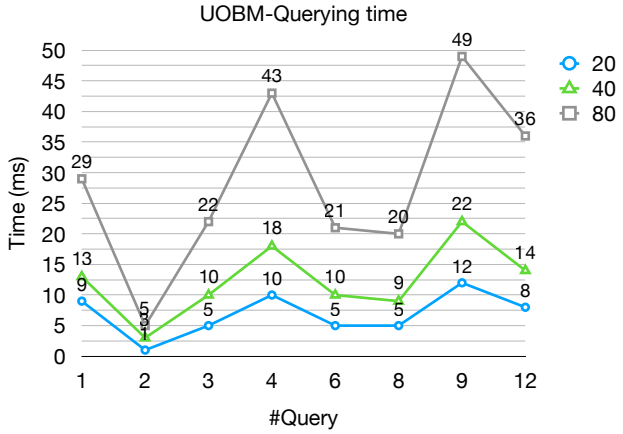

Figure 4.6: Querying time (in milliseconds) spent in answering queries using 20, 40 and 80 universities.

Chapter 5

SALON: Sequence Alignment Ontology

Sequence alignment is a fundamental process in bioinformatics that involves arranging two or more biological sequences (DNA, RNA, or proteins) in a way that identifies regions of similarity between them. They are pivotal for various applications, including phylogenetic tree construction, gene annotation, protein structure prediction, and identifying conserved motifs or domains. Over the years, many algorithms and tools have been developed to address the challenges in sequence alignment, each offering unique advantages depending on the biological problem at hand.

This chapter introduces SALON, an OWL 2 ontology that formally represents pairwise and multiple-sequence alignments. SALON is grounded in the FAIR principles and offers a formal structure for sequence alignments by leveraging concepts from the Multiple Alignment Ontology (MAO) (and extending it with additional OWL properties), facilitates the integration of diverse biological databases via federated SPARQL queries and ensures data validity through SWRL rules.

5.1 Related Work

Thompson et al. (J. Thompson *et al.* 2005) describe a Multiple Alignment Ontology (MAO), an OBO ontology for data retrieval and exchange in the fields of multiple sequence alignments (DNA/RNA/protein), and 3D structures. The MAO ontology contains multiple cross-references to related ontologies and other datasets, e.g., UniProtKB (The UniProt Consortium 2018) and Genbank (Clark *et al.* 2015). It provides a task-oriented design to facilitate communication between different methods for constructing, analysing and annotating sequence alignments. To this end, the top-level concept of MAO defines a subset of sequences, which in turn defines alignment sequences and columns. Hierarchical relationships between concepts are defined by *is_a* (specialisation relation) and *part_of* (partitive relation). Each of these concepts can then be characterised according to attributes. For example, MAO defines an associative relation, *is_name*, which allows specifying a user-defined name for a given sequence to provide a further understanding of such sequence. It also allows the integration of 3D structural information about proteins. However, as of 2015, the ontology is tagged as deprecated/inactive in the OBO Foundry website[1].

Furthermore, the MAO ontology is only available in OBO format, which can not take advantage of automated reasoning for ontology validation and the search for subsumption hierarchies. However, it can be translated to OWL (as OBO can be considered a subset of OWL). Due to the nature of OBO, most knowledge is represented as labels, lacking OWL entailments related to reasoning. Therefore, this ontology is a simple approach to representing this kind of knowledge, limiting its use to the presentation, interoperability, and data sharing between multiple alignment protocols.

KinView (McSkimming *et al.* 2016) is a visual comparative sequence analysis tool. It includes an ontology MSAOnt, a simple schema for relating sequences to a profile or consensus sequence. MSAOnt is compared to MAO, asserting that MSAOnt is a minimalist representation of a profile or consensus alignment components, while MAO is a more generic ontology. The paper does not provide details about the ontology, and it appears as not available for download.

This chapter presents an approach that reuses MAO conceptualisation and extends it with additional OWL, mainly object and datatype properties and SWRL axioms, to enable data validation.

[1] In URL: `https://obofoundry.org/`

5.2 Methods

The proposed ontology has been developed using the standard ontology 101 development process (Noy and Mcguinness 2001), consisting of seven steps. Below we describe how these steps have been approached in this work:

1. *Determine the domain and scope of the ontology*. SALON is focused on the description and formalisation of biological sequence alignments.

2. *Consider reusing existing ontologies*. We have reviewed the OBO Foundry ontologies seeking existing ontologies that match SALON scope and intended application. We did not find any other related ontology apart from MAO.

3. *Enumerate important terms in the ontology*. Essential terms in the ontology are those needed to describe biological sequence alignments. These terms have been extracted from MAO and manually extended with other concepts defined, taking information from the scientific literature in the field. Examples of such terms are Alignment, Alignment Sequence, Alignment Column, Feature, and Sub Alignment. Furthermore, terms for describing alignment construction methods are selected, for example, Deterministic Approach and Stochastic Approach. Finally, we included terms regarding scores, such as Alignment Score, Column Score, and Column Score Function.

4. *Define classes and the class hierarchy*. Classes in the ontology correspond to important terms. In contrast with MAO, complete covering of 3D structural information is not included in SALON. We are focused on merely describing sequence alignments, ranging from a single residue to a set of sequences (i.e., multiple sequence alignment) and its construction method and/or scoring. Examples of classes in the ontology are: *Alignment*, *AlignmentSequence*, *AlignmentColumn*, *ConstructionMethod* and *AlignmentScore*. We have defined *DNAAlignmentSequence* and *ProteinAlignmentSequence* as subclasses of *AlignmentSequence*. *DeterministicApproach* and *StochasticApproach* have been defined as subclasses of *ConstructionMethod*.

5. *Define the properties of classes and slots*. Object properties and data properties (slots) are defined to describe Alignments. For example, an alignment has columns and contains sub-alignments. A sub-alignment has sequences, an alignment sequence has features, an alignment has a score, and a sub-alignment has a construction method. On the other hand, alignment sequences are described by the data properties character, identifier, accession number, keyword, and score value.

Table 5.1: SALON object properties.

Object Property	Domain	Range	Inverse
hasConstructionMethod	SubAlignment	ConstructionMethod	-
hasAlignmentScore	Alignment	AlignmentScore	-
hasAlignmentScoreFunction	AlignmentScore	AlignmentScoreFunction	-
hasColumn	Alignment	AlignmentColumn	isColumnOf
hasColumnScore	AlignmentColumn	ColumnScore	-
hasColumnScoreFunction	ColumnScore	ColumnScoreFunction	-
hasSubAlignment	Alignment	SubAlignment	isSubalignmentOf
hasFeature	AlignmentSequence	Feature	isFeatureOf
hasSequence	SubAlignment	AlignmentSequence	isSequenceOf
hasAssociationWith	ProteinSequenceAlignment	Protein	-

6. *Define the facets of the slots*. This step includes the definition of cardinality constraints and value restrictions. An example of cardinality constraint in our ontology is that a Sub-alignment has at least two Alignment sequences.

7. *Create instances*. Ontology Individuals are obtained by mapping the data from different data sources to RDF according to the ontology specification. One of the data sources used in the proposed use cases is the benchmark alignment database BAliBASE (J. D. Thompson, Koehl, *et al.* 2005).

The SALON ontology has been developed using the Prótegé OWL editor version 5 in OWL 2 format. To assess the compliance of the ontology against the FAIR principles, we have used the FOOPS! validator[2], a web service for detecting best practices according to each FAIR principle, scoring the highest mark in 19 out of 24 tests.

5.2.1 Concepts and relationships

The proposed ontology[3] consists of 30 classes, 14 object properties (see Table 5.1) including 4 inverse object properties to ensure linking to related resources, 27 data properties and 144 logical axioms. The top-level classes of the ontology are *Alignment*, *AlignmentColumn*, *AlignmentScore*, *AlignmentScoreFunction*, *AlignmentSequence*, *ColumnScore*, *ColumnScoreFunction*, *ConstructionMethod*, *Feature*, *Protein*, and *SubAlignment* (see Figure 5.1).

The *Alignment* class represents a pairwise or multiple sequence alignment. An *Alignment* contains at least one *SubAlignment*, representing a subset of sequences with some property in common. This can be useful to represent clusters of sequences within the alignment, e.g., phylogenetic trees for multiple sequence alignments.

[2]In URL: `https://foops.linkeddata.es/FAIR_validator.html`
[3]Ontology available at `https://w3id.org/salon`

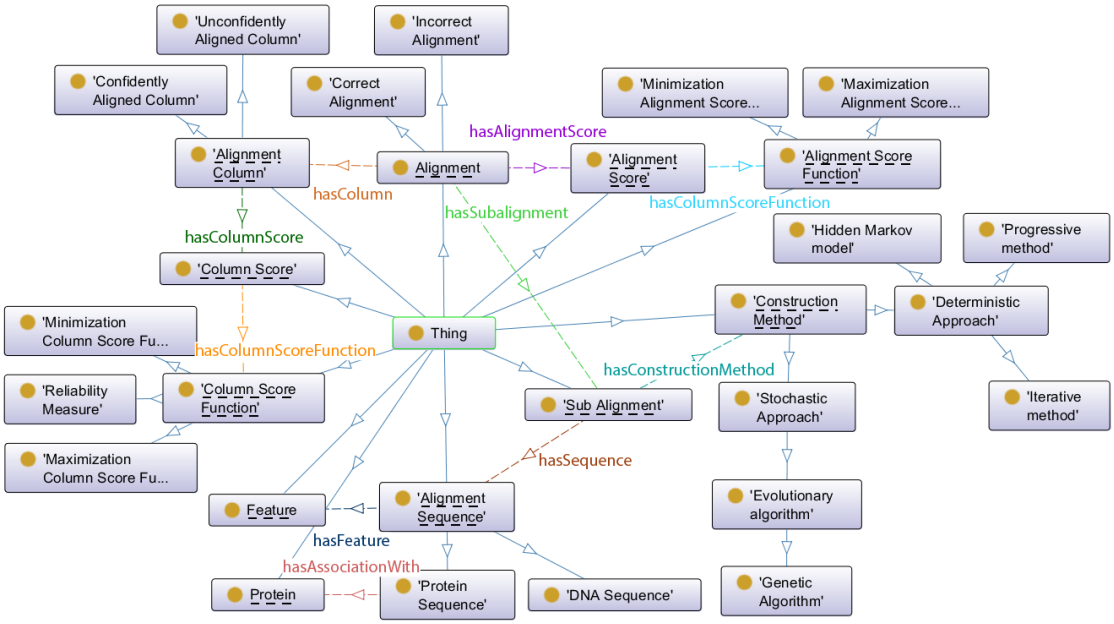

Figure 5.1: Classes in SALON. Underline classes represent top-level classes in the ontology. The *is-a* relationship is depicted as blue arrows, with object properties depicted as dashed arrows. Boxes represent classes.

Table 5.2: Part of SALON data properties.

Data Property	Domain	Range
atSequenceIndex	AlignmentColumn	integer
column	AlignmentColumn	string
score	AlignmentScore or ColumnScore	integer or float
scoreMax	AlignmentScore or ColumnScoreFunction	integer or float
scoreMin	AlignmentScore or ColumnScoreFunction	integer or float
scoreCutoff	ReliabilityMeasure	integer or float
FScore	Feature	integer or float
columnWeight	AlignmentColumn	integer
consensusCharacter	AlignmentColumn	short
gapCharacter	SubAlignment	short
accessionNumber	AlignmentSequence or Protein	string
identifier	AlignmentSequence	string
length	AlignmentSequence	integer
organism	AlignmentSequence	string

Attributes of *AlignmentColumn* are *ColumnScore*, *ColumnWeight*, *ConsensusCharacter*, among others. Some attributes of *AlignmentSequence* are *organism* or *keyword*. Table 5.2 shows SALON data properties. Range in data properties correspond to a datatype, i.e. string or integer.

Table 5.3: XML mappings from MACSIMXML to SALON classes. Some SALON data properties, such as *gapCharacter*, are derived from the sequence itself and are not shown in the table.

XML	Direct mappings
`<aln-name></aln-name>`	salon:SubAlignment class, salon:subAlignmentName
`<sequence seq-type="Protein"></sequence>`	salon:ProteinAlignmentSequence class
`<seq-name></seq-name>`	salon:identifier
`<accession></accession>`	salon:accessionNumber
`<sequence></sequence>`	salon:sequence
`<fitem></fitem>`	salon:hasFeature
`<ftype></ftype>`	salon:FType
`<fstart></fstart>`	salon:FStart
`<fstop></fstop>`	salon:FStop
`<fnote></fnote>`	salon:FNote

5.2.2 SALON instance generation

This section focuses on the automatic generation of SALON instances from sequence alignment data. This transformation process is done from an input file in MACSIM XML format (J. D. Thompson, Muller, *et al.* 2006), and produces the corresponding RDF triples (SALON instances) as seen in Table 5.3. To this end, we have developed a Python client package available at GitHub[4] with several utilities to work alongside an RDF repository. For RDF manipulation, we use the rdflib library[5], which generates an RDF graph that can be serialised into a plethora of semantic-aware formats such as RDF/XML, NTriples, Turtle, and JSON-LD. The RDF file can be loaded in the Prótegé OWL editor to visually inspect the sequence alignment data. It is worth noting that new mapping functions can be applied to alignments in other file formats, producing SALON instances.

5.2.3 Ensuring alignment correctness

Scoring methods for sequence alignments assign numerical values to differentiate good alignments from poor ones. The alignment score in pairwise alignments is computed based on matches/mismatches and gaps in both sequences. In multiple alignments, scores can be determined based on scoring functions. The score function is independently computed for every column in the alignment. The alignment score will be the sum of these values. It is important to note that there is no consensus about the best metric to score a multiple sequence

[4]In URL: `https://github.com/benhid/SALON`
[5]In URL: `https://rdflib.readthedocs.io/en/stable/index.html`

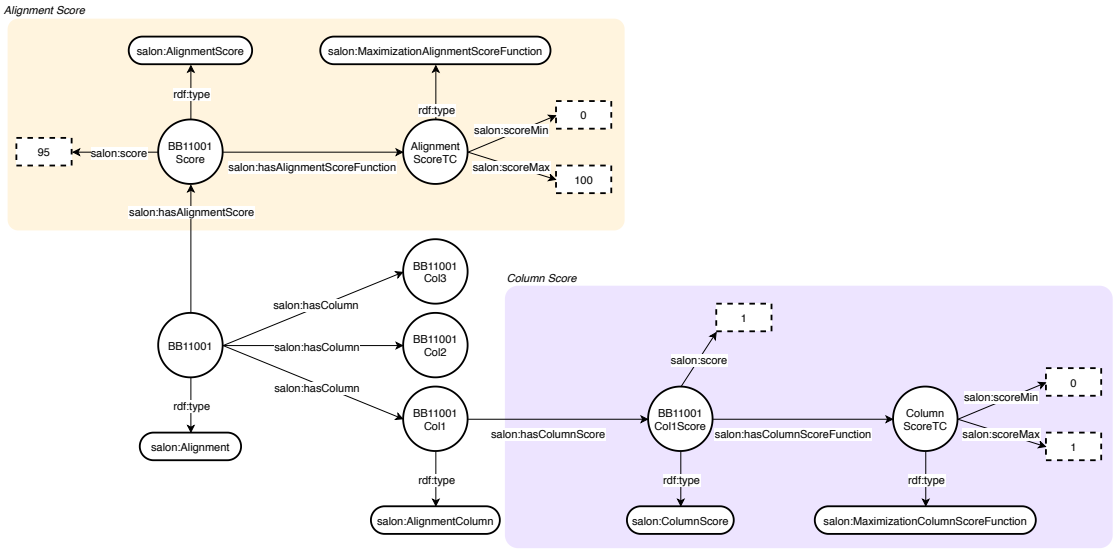

Figure 5.2: Alignment and column score functions represented according to SALON. The *Alignment* contains one *AlignmentScore* which scores 95. The underlying *MaximizationAlignmentScoreFunction* ranges from 0 to 100. Additionally, one *ColumnScore* scoring 1 is included in the figure. It contains one *MaximizationColumnScoreFunction* ranging from 0 to 1.

alignment, so several functions exist to this end.

Some examples of score functions include the percentage of totally conserved columns, percentage of non-gaps, and entropy (Shannon 1948) scores. For example, the percentage of totally conserved columns and percentage of non-gaps scores can be either 0 or 1 for a given column (i.e., *ColumnScoreFunction*), and ranges from 0 to 100 (representing a percentage) for the whole alignment (*AlignmentScoreFunction*). The entropy value of any given column in a multiple sequence alignment ranges from 0 (only one residue is represented in that position of the sequence) to 4.322 (all 20 residues are represented). Thus the alignment scores a maximum of number of sequences $* 4.322$. On the other hand, reliability score functions such as GUIDANCE (Penn *et al.* 2010) and Heads-or-Tails (Landan and Graur 2007) help us quantify the robustness of the sequence alignment. For example, GUIDANCE assigns a confidence score between 0 and 1 for each column in the alignment, which can be used to filter unreliably aligned regions before subsequent analysis.

SALON includes several SWRL rules to determine if any score value is out of range based on the underlying score function, which typically fits into an upper and lower bound. These rules can help scientists to ensure alignment correctness automatically. For example, to check if the alignment in Figure 5.2

is correct, we have defined the SWRL rule in Listing 6. This rule is generic and is valid to check the correctness of any alignment modelled according to SALON. To check if an alignment is incorrect, we have defined four SWRL rules to consider all the incorrectness possibilities, i.e. $columnscore > columnscoreMax$, $columnscore < columnscoreMin$, $alignmentscore > alignmentscoreMax$ and $alignmentscore < alignmentscoreMin$.

```
Alignment(?a)
^ hasColumn(?a, ?c)
^ hasColumnScore(?c, ?cs)
^ score(?cs, ?csc)
^ hasColumnScoreFunction(?cs, ?csf)
^ scoreMax(?csf, ?cmax)
^ scoreMax(?csf, ?cmin)
^ swrlb:lessThanOrEqual(?csc, ?cmax)
^ swrlb:greaterThanOrEqual(?csc, ?cmin)
^ hasAlignmentScore(?a, ?as)
^ score(?as, ?asc)
^ hasAlignmentScoreFunction(?as, ?asf)
^ scoreMin(?asf, ?amin)
^ scoreMax(?asf, ?amax)
^ swrlb:lessThanOrEqual(?asc, ?amax)
^ swrlb:greaterThanOrEqual(?asc, ?amin)

    -> CorrectAlignment(?a)
```

Listing 6: SWRL rule to check Alignment correctness.

Listing 7 shows the rule to check the latest case. The SWRL rule in Listing 8 checks for unreliable columns in any alignment modelled in SALON according to a reliability score function.

SWRL rules require a reasoner that supports them. The rules engine uses alignments and rules as inputs for inferring new facts, e.g., if the alignment is correct or not. Furthermore, as SWRL rules are built in the ontology, no external validation system is required to ensure alignment correctness.

RDF repository

To test how the ontology instances can be used in real scenarios, we have deployed an on-premises instance of Stardog Knowledge-graph platform. Stardog supports the SPARQL query language and thus provides access to the data via the SPARQL protocol, a declarative language for performing operations over SPARQL endpoints, capable of receiving and processing those requests. Unlike other triple stores, Stardog supports SWRL rules (which this ontology uses). Therefore, SALON requires a running Stardog server instance to unlock all features at the

```
Alignment(?a)
^ Alignment(?a)
^ hasAlignmentScore(?a, ?as)
^ score(?as, ?asc)
^ hasAlignmentScoreFunction(?as, ?asf)
^ scoreMin(?asf, ?amin)
^ swrlb:lessThanOrEqual(?asc, ?amin)

    -> IncorrectAlignment(?a)
```

Listing 7: An example of SWRL Rule to check Alignment incorrectness.

```
Alignment(?a)
^ hasColumn(?a, ?c)
^ hasColumnScore(?c, ?cs)
^ score(?cs, ?csc)
^ hasColumnScoreFunction(?cs, ?csf)
^ ReliabilityMeasure(?csf)
^ scoreCutoff(?csf, ?csfc)
^ swrlb:lessThan(?csc, ?csfc)

    -> UnconfidentlyAlignedColumn(?c)
```

Listing 8: An example of SWRL Rule to check unreliable columns in the alignment.

server side (to provide additional functionalities such as testing user interfaces on top of it). Virtuoso or any other triple store can be used, but in such a case, SWRL will not be evaluated, and the validation will be limited to the inference support provided.

RDF instance files can be inserted into Stardog using a SPARQL update query. Automatic scripts for RDF repository population using Python can be found at GitHub[6].

5.2.4 Semantic enrichment of protein sequences

Due to the increasing use and development of high-throughput platforms, many biological data are available in public databases and repositories. Usually, these databases provide different information regarding the same biological structure. Bringing together all this information into a single and unified repository might be very time-consuming. SALON can help to address this issue. The SPARQL protocol defines a federated form of the SPARQL query language that provides access to remote endpoints. Those queries (which can be issued to a SPARQL endpoint) are not constrained to working with a single database, but federated

[6]In URL: `https://github.com/benhid/SALON`

queries are supported across multiple endpoints. This powerful feature enables the online query of several related SPARQL Endpoints. However, the resolution of such queries relies on the availability of the individual SPARQL Endpoints to provide a complete answer.

This way, complete information from several Open Linked Data resources can be integrated using a single query through the federation. For example, we can use the information already stored in our repository about protein sequences to integrate data from other data sources. When a new alignment record is inserted in our RDF repository, a federated query is automatically triggered against UniProt Knowledgebase (UniProtKB), the most well-known databases of protein structures and functional information. UniProtKB stores data about proteins, e.g., source organism, name, NCBI identifier, and gene, and provides a public accessible SPARQL endpoint[7].

However, the Linked Data sources' distributed nature makes these federated queries fail due to the system failure in one of its parts. For this reason, in the example described in this section, the additional information returned with the federated query is inserted into the local database.

The sequence's accession number can be used, for example, to retrieve 3D protein structures and compare alternative multiple sequence alignments in terms of their relative accuracy (Kemena *et al.* 2011). SALON can use this identifier (represented by the *accessionNumber* data property) to find matched records from UniProtKB and retrieve information that would aid subsequent analyses. Listing 9 shows the corresponding federated SPARQL query to aggregate information about the protein 1D2N. The range of the SALON *accessionNumber* data property is a string containing the UniProtKB entry of the protein. Using this value, we obtain all the information related to such a protein and insert it in our RDF repository.

Note that this information about each protein involved in the alignment process is unavailable in the alignment data. Thus, a user obtaining the alignment will need to retrieve such information from UniProtKB. Traditionally this could be done by manually exploring the UniProt portal with a vast human effort (for alignments with many proteins). An alternative would be to develop a software application using the UniProt services to retrieve such data, with the required programming skills from the user. Federated SPARQL queries simplify this process to retrieve updated information related to this data.

[7] In URL: `https://sparql.uniprot.org/`

```
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX up:<http://purl.uniprot.org/core/>
PREFIX pdb:<http://rdf.wwpdb.org/pdb/>
PREFIX salon:<https://w3id.org/salon#>
INSERT {
    <seq_uri> salon:organism ?ncbi ;
              salon:hasAssociationWith ?protein_uri .
    ?protein_uri a salon:Protein .
    ?protein_uri salon:description ?protfullname .
    ?protein_uri salon:keyword ?protmnemonic .
    ?protein_uri salon:proteinName ?protfullname .
    ?protein_uri rdfs:seeAlso ?organism .
    ?protein_uri rdfs:seeAlso ?protein .
    ?protein_uri rdfs:seeAlso ?pdb .
}
WHERE {
    <seq_uri> rdf:type salon:ProteinAlignmentSequence .
    SERVICE <http://sparql.uniprot.org/sparql> {
        BIND(pdb:1D2N AS ?pdb) .
        ?protein a up:Protein ;
                 rdfs:seeAlso ?pdb ;
                 up:recommendedName ?protname ;
                 up:mnemonic ?protmnemonic ;
                 up:organism ?organism ;
                 up:encodedBy ?gene .
        ?gene skos:prefLabel ?gename .
        ?protname up:fullName ?protfullname .
        ?organism up:mnemonic ?orgmnemonic ;
                  up:scientificName ?orgscientific .
    BIND(STRAFTER(STR(?protein),
       "http://purl.uniprot.org/uniprot/") AS ?ac)
    BIND(STRAFTER(STR(?organism),
       "http://purl.uniprot.org/taxonomy/") AS ?ncbi)
    BIND(URI(CONCAT("salon:", STR(?ac))) AS?protein_uri)
    }
}
```

Listing 9: Example of a SPARQL query with a specific protein (1D2N) after using the template.

Generation of FASTA description lines

The FASTA format is used to represent biological sequences. Still, it lacks a strict standard specification, translating into consistency issues as bioinformatics tools fail to parse description lines correctly. This problem can cause malfunctioning of third-party tools and unexpected results when parsing those files. There are cases in which FASTA files' description lines are poorly described, or some infor-

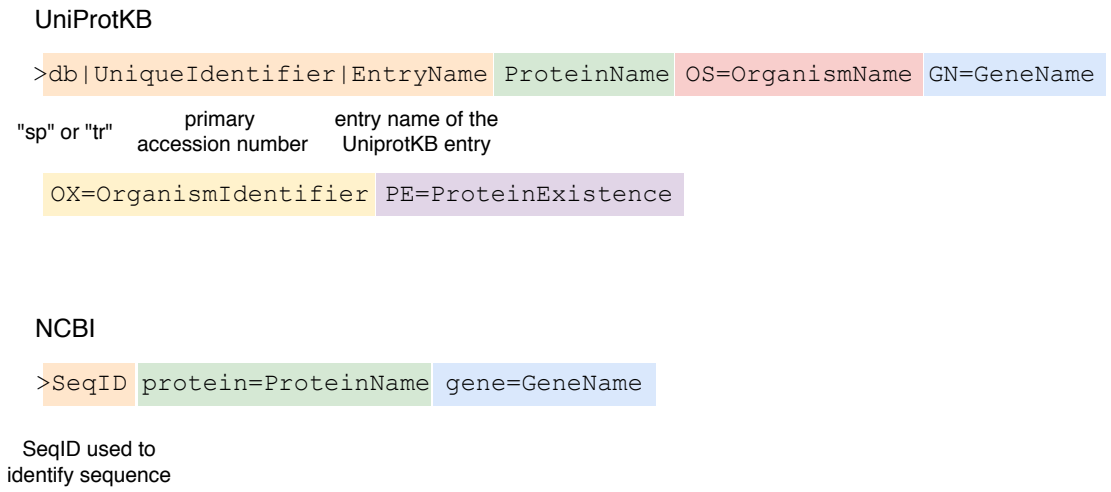

Figure 5.3: Formatting of FASTA header/description line (including optional modifiers) for a protein sequence in UnitprotKB and NCBI. For the sake of simplicity, some modifiers have been omitted.

```
>sp|Q01853|TERA_MOUSE Transitional endoplasmic reticulum ATPase
OS=Mus musculus OX=10090 GN=Vcp PE=1
```

Figure 5.4: UniProtKB FASTA description line for a protein sequence with known PBD identifier.

mation about the sequences (such as their source organism or accession number in reference databases) is missing.

For example, UniProtKB and NCBI provide different specifications about the required and optional information in their sequences' header lines (see Figure 5.3). It can be very time-consuming to manually retrieve missing tags from different sources to switch between specifications. Thus, a mapping must be applied between UniProtKB and NCBI resources to find cross-referenced entries. SALON can help scientists overcome this issue by filling the information gaps between such services. As mentioned above, the proposed RDF repository integrates information about sequences from different repositories, which is stored after a new alignment record is inserted by running a federated SPARQL query. Therefore, a specific SPARQL query can be run to obtain the desired information and compose a custom description line for the desired service. For example, the SPARQL query presented in Listing 10 can be used to obtain the different fields needed to build the header in the UniProtKB format: i.e., db, UniqueIndentifier, EntryName, ProteinName, OrganismName, GeneName, and ProteinEx-

istence (see Figure 5.3), when the only known information is the PDB identifier of the associated protein. Figure 5.4 describes the header generated. Without the SALON RDF repository, the scientist would have needed to search in the reference database for the entry (or entries) that matches the PDB identifier of the related protein. It would then have been required to manually extract all the required information to construct the description line (or description lines, if a PDB identifier matches several entries). The sequences/alignments can be meaningfully compared and integrated using SALON to standardise FASTA file headers. Informative labels can be used in, for example, phylogenetic analysis, where the taxonomic lineage can be used to evaluate the phylogenetic tree.

```
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
PREFIX up:<http://purl.uniprot.org/core/>
PREFIX salon:<https://w3id.org/salon#>
SELECT DISTINCT ?db ?UniqueIdentifier ?EntryName
       ?OrganismName ?OrganismIdentifier ?ProteinName
       ?GeneName ?ProteinExistence
WHERE {
    <seq_uri> a salon:ProteinAlignmentSequence ;
              salon:identifier ?UniqueIdentifier ;
              salon:organism ?OrganismIdentifier ;
              salon:hasAssociationWith ?protein .
    ?protein a salon:Protein ;
              salon:proteinName ?ProteinName ;
              rdfs:seeAlso ?pdb .
    SERVICE <http://sparql.uniprot.org/sparql> {
        ?pdb a up:Protein ;
             up:reviewed ?db ;
             up:encodedBy ?gene ;
             up:mnemonic ?EntryName ;
             up:existence ?ProteinExistence ;
             up:organism ?organism .
        ?gene skos:prefLabel ?GeneName .
        ?organism a up:Taxon;
                  up:scientificName ?OrganismName .
    }
}
```

Listing 10: Query to return UniProtKB FASTA information from SALON RDF repository.

It is frequent that, when comparing a highly conserved protein or domain of such a protein from two closely related species, a single PDB identifier can be mapped to several UniProtKB entries. In those cases, manual intervention by the domain expert might be required to help identify the most suitable option for the experiment required.

5.3 Results

To test SALON, we have focused on a set of manually constructed and verified alignments from a relevant dataset such as BAliBASE 4.0. This dataset provides high-quality reference alignments of DNA and protein sequences in XML and FASTA formats. In this database, each multiple alignment contains information about the sequences and some quality measures related to the alignments themselves, such as the column score, i.e., the fraction of aligned columns that are correctly reproduced.

```
<?xml version="1.0"?>
<macsim>
 <alignment>
  <aln-name>1aab_ref1</aln-name>
  <sequence seq-type="Protein">
   <seq-name>1aab_</seq-name>
   <seq-info>
    <accession>1aab</accession>
    <nid>1aab_</nid>
    <length>96</length>
   </seq-info>
   <seq-data>
   ---gkgdpkkprgkmcsekgkfadka-ippkge----------
   </seq-data>
   </sequence>
  <sequence seq-type="Protein">....</sequence>
  <sequence seq-type="Protein">....</sequence>
 </alignment>
</macsim>
```

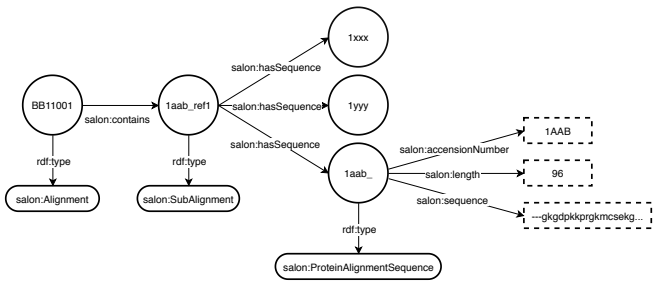

Figure 5.5: BB11001 alignment from BAliBASE (left) represented in RDF according to SALON (right). In this example, our multiple sequence alignment contains a sub-alignment of three protein sequences of known name, accession number, length and primary structure. Only one of the three sequences (bottom right corner) is fully represented for the sake of clarity.

Figure 5.5 depicts a simplified representation of a multiple sequence alignment as an RDF Graph using SALON Schema. On the left side, an alignment in MACSIM XML format is partially represented (i.e., BB11001 instance from BAliBASE). On the right side, the same alignment is shown according to our proposal, where nodes correspond to subjects or objects, and edges correspond to predicates (according to the RDF terminology).

BAliBASE alignments can also be downloaded in FASTA, a text-based file format representing either nucleotide sequences or peptide sequences using single-letter codes. FASTA files contain headers or description lines, distinguished from the sequence itself by a greater-than symbol. The information in the description line of the sequence (e.g., its protein identifier or source organism) can also be mapped to their corresponding terms in the ontology using a custom mapping function to transform FASTA to semantic-aware formats (i.e. RDF). When not present, this data can be retrieved from different sources to construct the de-

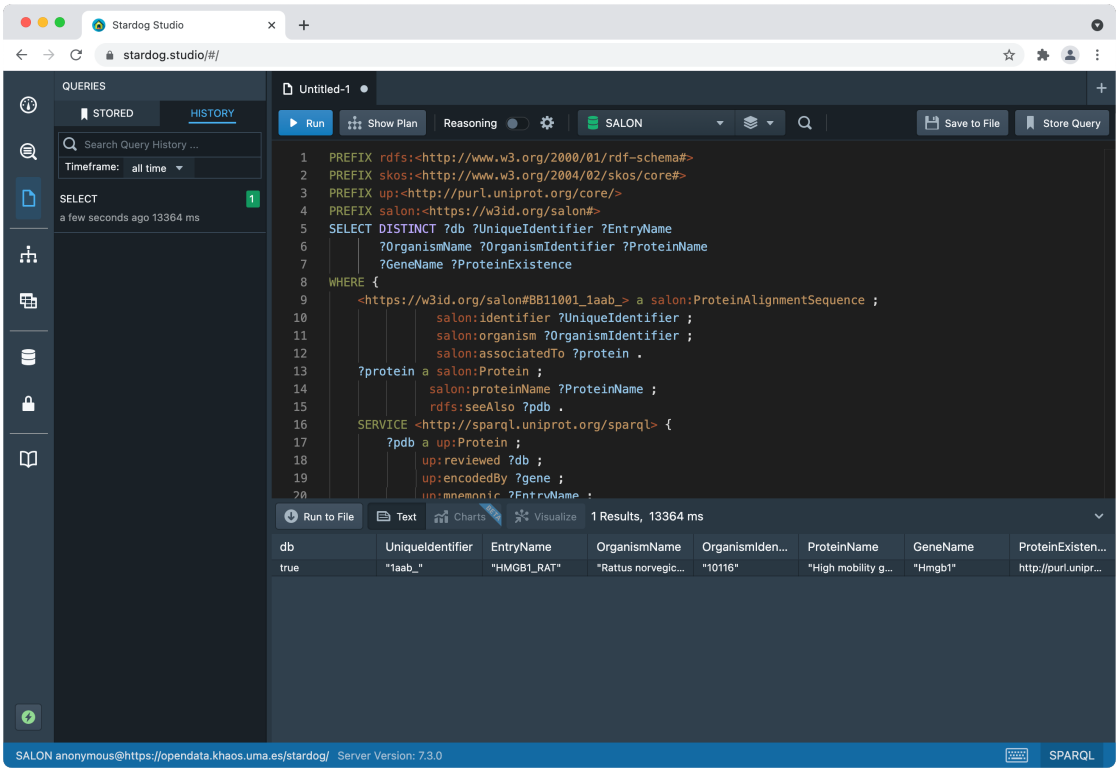

Figure 5.6: Sample query in Stardog Studio. This query returns additional information associated with the sequence *salon:BB11001_1aab_* from UniProtKB database.

scription line. By matching identifiers found in the description line of a sequence against entries in authoritative biological databases, we can add informative labels to ensure that third-party tools can be used without adaptation. This consistency problem has been addressed in the past by web tools such as SeqScrub Foley *et al.* 2019, but no SPARQL-based approach has been applied to the best of our knowledge. Figure 5.6 shows the SPARQL query in Listing 10 using Stardog Studio[8], a free web-based tool for working with Stardog RDF repositories. This query retrieves the missing labels needed to construct the description line of a sequence by querying UniProtKB, obtaining the gene name and the organism related to a given protein, among others. Then, the FASTA file header is constructed as shown in Figure 5.4. The SALON ontology instances for these data are then stored in the RDF repository to enhance sequence alignment information. To explore these results, a SPARQL endpoint is available[9]. Stardog Studio also provides a SPARQL query editor for exploring the alignment data, as shown

[8]In URL: `https://www.stardog.com/studio/`

[9]In URL: `https://ontologies.khaos.uma.es/salon/`. Authentication credentials *salon-demo*:*salondemo*.

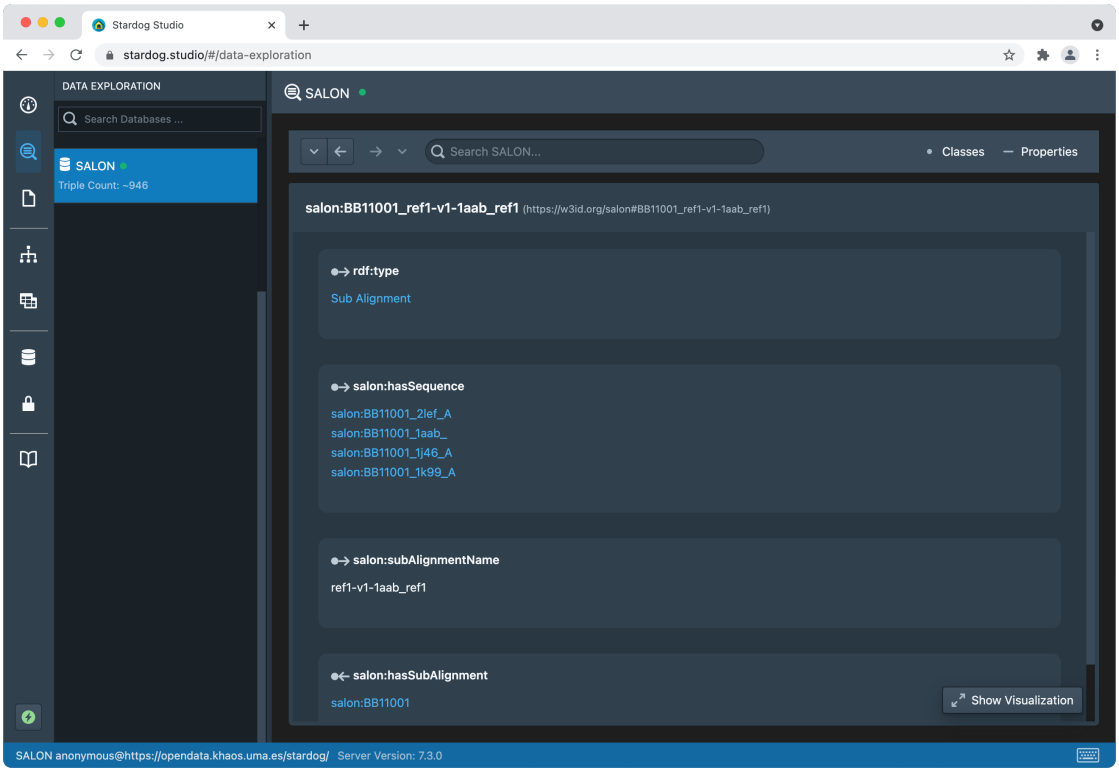

Figure 5.7: Stardog Studio Data Exploration tab showing an alignment represented using SALON (i.e., BB11001 instance from BAliBASE). This sub-alignment contains four sequences, namely *2lef_A*, *1aab_*, *1j46_A*, and *1k99_A*.

in Figure 5.7.

Additionally, these instances are provided in the GIT repository so users can use any Open Source tool with RDF and SWRL support (such as Protègè). With the incorporation of SWRL rules, alignments can be validated to ensure their correctness and robustness automatically. For example, an alignment inferred as incorrect by the reasoner can be due to an error in the underlying alignment algorithm, which can dramatically impact subsequent steps in the pipeline. Similarly, Figure 5.8 shows one column in the alignment inferred as unconfidently aligned by the reasoner. A further inspection of the Protégé Editor shows that, according to our data, the GUIDANCE confidence level of that column exceeds the cutoff and thus is marked as unreliable. We can remove unconfidently aligned columns to reduce errors caused by alignment errors in subsequent analysis (Wong *et al.* 2008; Privman *et al.* 2011).

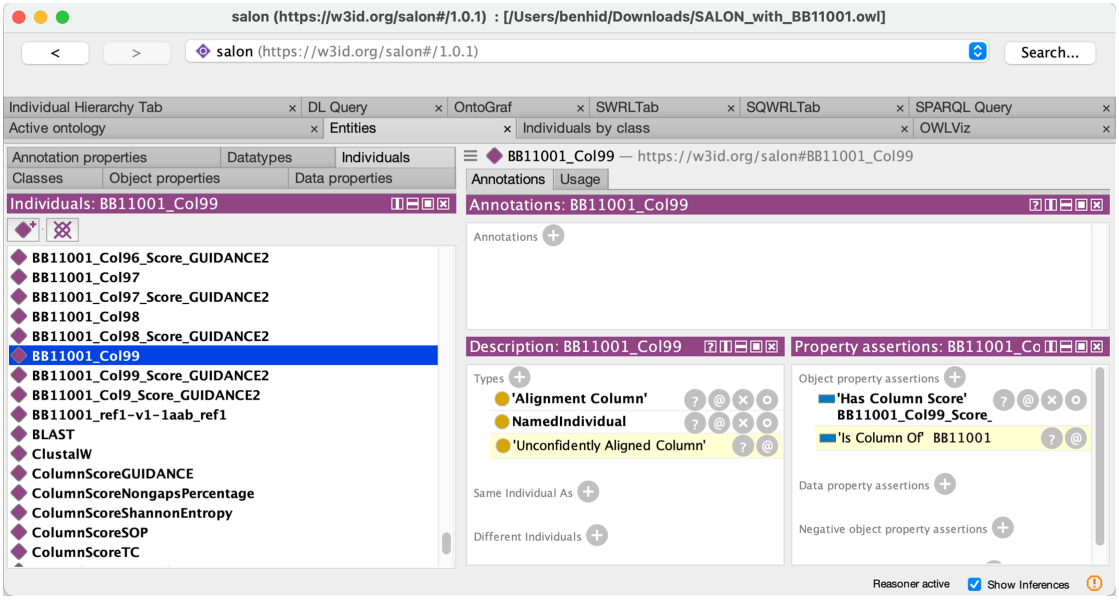

Figure 5.8: Protégé reasoning engine can be used to infer rule engine knowledge. In this example, the rule engine shows unreliable columns in the BB11001 alignment.

Chapter 6

META: Incorporating Semantics into Data Analysis

Machine Learning (ML) enables the discovery of patterns and the learning of models from data. One of the significant challenges in ML is integrating domain knowledge and context during the analysis process. The term "domain knowledge" refers to the general knowledge, or background knowledge, of the field or environment to which data analysis methods are applied. This contextual information can aid in selecting appropriate information, features, or techniques, reducing the search space, representing the output more understandably, and, overall, enhancing the entire process.

Knowledge sources for an application domain can vary significantly. We can acquire domain knowledge by consulting an expert or automatically extracting it from a data repository, a corpus of documents in different formats (XML, JSON, and so on), or a web page, among others. Although there are techniques that enable such extraction, such as ontology learning or schema inference, the way of representing this knowledge needs to be standardised, which complicates the integration of knowledge. For over 50 years, the use of knowledge in expert systems, semantic networks, or frames has made it possible to solve complex problems in various domains. Researchers in AI have developed various techniques for representing knowledge, including the use of ontologies. Recent approaches involves using Linked Open Data (LOD) and, more recently, Knowledge Graphs (KG). We can formally express the semantics of an application domain through ontologies and knowledge graphs since the ontology definition language OWL is based on Description Logic, a subset of First-Order Logic. Representing the domain's semantics in OWL allows us not only to formally represent the domain's features, relationships, rules, and constraints but also to infer new knowledge.

Despite the promising aspects of using OWL in representing domain knowledge, a significant limitation surfaces when tailoring it to specific problem-solving scenarios, which can often be critical when training ML models to predict or classify based on domain-specific data accurately. This disconnect can lead to misin-

terpretations or over-simplifications in the ML processes. It is essential, then, to address this gap. If not adequately addressed from the outset, we risk building models that, while technically accurate, miss the mark in real-world application scenarios.

In this chapter, we present META, an OWL ontology designed to represent domain knowledge in a standardised and formal manner as elements of an OWL ontology. META includes classes for entities such as Class, Property, Axiom, and Rule. META also models other elements like constraints and user preferences, offering a mechanism to integrate knowledge from various information sources and enabling its reuse in different applications. Thus, knowledge items can be easily encapsulated to be used as contextual elements in ML approaches. With META as the core, we have also developed a methodology allowing semi-automatic injection of this knowledge into machine learning algorithms. This integration and re-usability of knowledge enhance the efficiency and effectiveness of machine learning processes.

6.1 Related Work

Existing efforts to extend the expressiveness of OWL can be broadly grouped into two categories. The first category involves approaches that enhance the syntax of the ontology language itself, such as extensions to OWL DL to support annotations on axiom (Flügel *et al.* 2022), or modifications that allow for higher-order logic expressions (Lenzerini *et al.* 2021). While these approaches can extend the range of expressible statements, they often require significant alterations to the ontology language, which may lead to compatibility issues with existing tools and standards.

However, there is a growing interest in developing a more integrated approach to enhance the expressiveness of ontology languages. Some proposals have suggested embedding the meta-modeling capabilities directly into the ontology language, although these efforts are still at a preliminary stage. For example, in Vrandečić *et al.* 2006, the authors discuss the limitations of OWL DL in terms of its capacity to express complex assertions about the elements of an ontology, such as classes, properties, and, particularly, axioms. This limitation directly affects the ability to manage, maintain, and expand ontologies throughout their life cycle.

In Peters *et al.* 2007, the authors proposed a meta-model to associate linguistic and terminological data with OWL elements such as properties, classes, and individuals. The objective of the proposed model's design is to provide multilingual information for ontologies. As such, each element of the meta-model is associated with one or more lexicalizations (linguistic representations) drawn from one or multiple languages. However, it is important to note that this work represents a specific and focused contribution within the broader domain of knowledge representation and ontologies in the Semantic Web.

6.2 A Comprehensive Meta-Ontology for Effective Representation

We have defined a higher-order meta-ontology, which serves as a comprehensive framework for capturing and effectively representing intricate ontology constructs. By providing a structured and standardized way to represent ontological elements, this meta-ontology lays the groundwork for more efficient and interoperable knowledge-based systems. The proposed ontology has been made publicly accessible[1] and has been developed using the Prótegé OWL editor version 5 in OWL 2 format. The complete class hierarchy is depicted in Figure 6.1. To provide a clearer understanding of how OWL elements are modelled, Table 6.1 represents the object properties in META.

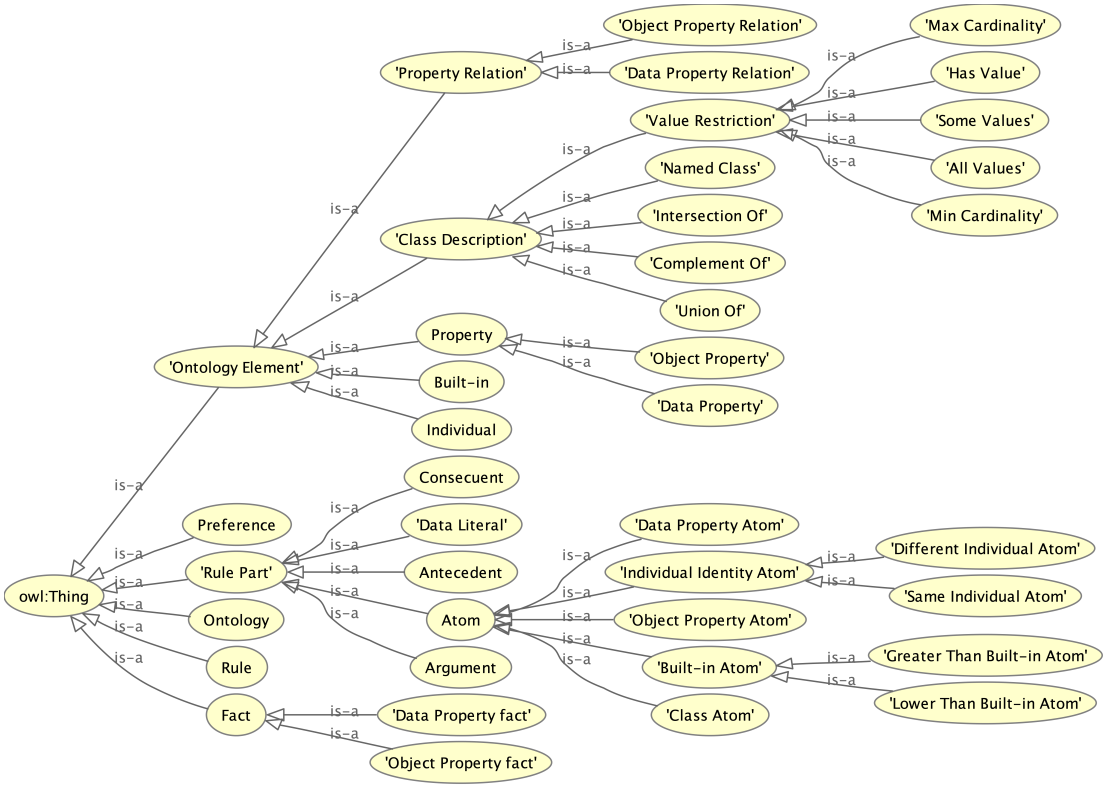

Figure 6.1: Overview of META's class hierarchy.

The developed ontology comprises several key concepts, including `Ontology`, `Class Description`, `Property`, `Built-in`, and `Individual`. Consequently, on-

[1] In URL: `https://w3id.org/meta`

Table 6.1: Summary of object properties with their respective domains and ranges.

Object property	Domain	Range
allValuesFrom	AllValues	ClassDescription
byProperty	ObjectPropertyRelation	Property
complementOf	ComplementOf	ClassDescription
consistsOf	Ontology	Ontology Element
disjointWith	ClassDescription	ClassDescription
domain	Property	ClassDescription
equivalentTo	ClassDescription	ClassDescription
hasAntecedent	Rule	Antecedent
hasArgument	Atom	Argument
hasAtom	Antecedent/Consequent	Atom
hasAxiom	Ontology	Rule
hasBuiltInExpression	BuiltinAtom	Builtin
hasClassPredicate	ClassAtom	ClassDescription
hasConsequent	Rule	Consequent
hasDataLiteral	Atom	DataLiteral
hasDataPropertyPredicate	DataPropertyAtom	DataProperty
hasFact	Preference	Fact
hasIndividual	Preference	Individual
hasObjectPropertyPredicate	ObjectPropertyAtom	ObjectProperty
hasObjectPropertyRelation	Individual	ObjectPropertyRelation
hasPreference	Ontology	Preference
hasValue	HasValue	Individual
intersectionOf	IntersectionOf	ClassDescription
isAntecedentOf	Antecedent	Rule
isAtomOf	Atom	Antecedent/Consequent
isAxiomOf	Rule	Ontology
onClass	IntersectionOf/UnionOf	ClassDescription
onDataProperty	DataPropertyFact	DataProperty
onIndividual	ObjectPropertFact	Individual
onObjectProperty	ObjectPropertFact	ObjectProperty
onProperty	ValueRestriction	Property
range	Property	ClassDescription
someValues	SomeValues	ClassDescription
subClassOf	ClassDescription	ClassDescription
type	Individual	ClassDescription
unionOf	UnionOf	ClassDescription
withDataLiteral	DataPropertyRelation	DataLiteral
withIndividual	ObjectPropertyRelation	Individual

tology axioms are capable of being reified as instances of these concepts. For example, an axiom describing a subclass relationship, such as `A rdfs:subClassOf B`, can be annotated in META by representing each class as an individual (of the META class *Class*) (*A rdf:type meta:NamedClass* and *B rdf:type meta:NamedClass*). Then, these instances are connected using the corresponding object property (A

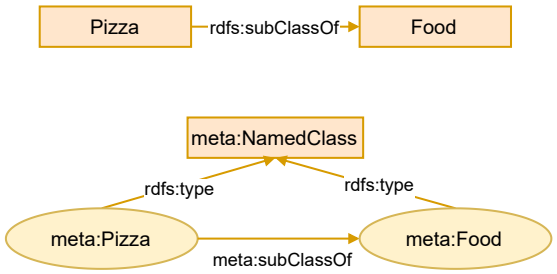

Figure 6.2: On the top, conventional OWL subclass representation. On the bottom, representing subclass relationships by mapping class elements to individual instances and linking them via the "subClassOf" data property.

meta:subClassOf B). This example is illustrated in Figure 6.2.

Object and data property assertions defines relations between individuals (or instance). OWL properties can be annotated using `ObjectPropertyRelation` and `DataPropetyRelation` classes in META, as depicted in Figure 6.3. These classes offer a structured way to showcase the various relationships that exist between instances within the ontology.

More complex axioms, such as class restrictions, can also be represented in this manner. Take, for instance, the following class description: `WomanCollege` $\equiv$ `College` $\sqcap$ $\forall$`HasStudent.(`$\neg$`Men)`. This description implies that a "Woman College" is defined as a college where all its students are not male students. The visual representation of this class relationship is illustrated in Figure 6.4.

Moreover, each `Ontology` is accompanied by a distinctive set of `Rules` and `Preferences`, extending the depth and versatility of the ontology structure. This integration brings in a multi-faceted dimension to the ontology, enabling it to cover a broad range of use cases and applications as described next.

Figure 6.5 (top) illustrates the rules hierarchy, which represents logical rules defined in the Semantic Web Rule Language. This enhances the ontology's capability to handle complex scenarios and support advanced knowledge-based applications, i.e., reasoning tasks.

On the other hand, Figure 6.5 (bottom) showcases the preference hierarchy, introducing a means of expressing user-defined preferences and priorities within the ontology. Preferences can range from simple ranking systems to complex criteria that influence decision-making processes or facilitate personalised information retrieval.

Existing ontologies can be converted into instance data of META by lever-

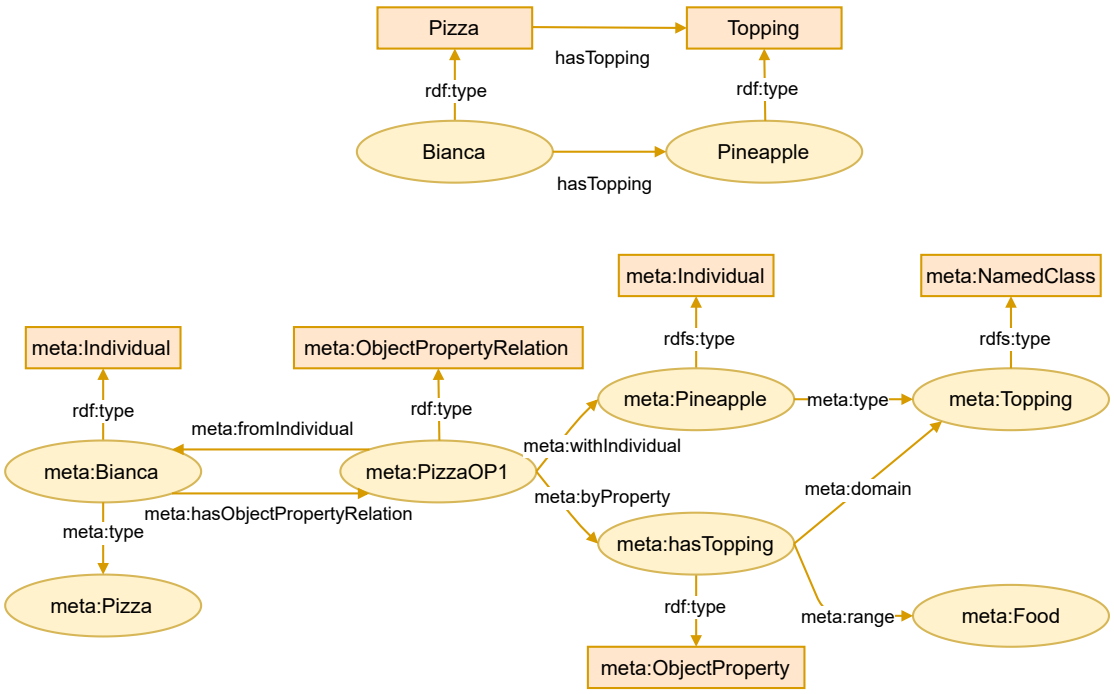

Figure 6.3: In the given example, a bianca pizza is described as having a pineapple topping. On the top, the traditional OWL representation is depicted where "Bianca" is an instance of "Pizza" and has a "Topping" of "Pineapple". Below this, META ontology annotation approach for OWL properties.

aging the extensive vocabulary within the ontology. As a part of the process, conceptual elements of the original ontology, such as classes, properties, relationships, and SWRL rules, are mapped to their corresponding instances within META, thereby maintaining the structural and semantic integrity of the original data. Meanwhile, these instances can be easily injected as contextual elements of ML processes.

In addition to developing META, we have created a comprehensive framework to support Knowledge Engineering. The META framework[2] provides researchers and practitioners with tools and guidelines for efficiently using META in various knowledge engineering tasks. Figure 6.6 represents the Knowledge Engineering Interface (KEI) we developed.

This KEI tool is tailored to knowledge engineers, offering features that allow them to import existing ontologies seamlessly. Furthermore, it enables mapping their conceptual elements to instances within META using an accessible and

[2]In URL: `https://github.com/benhid/meta-framework`

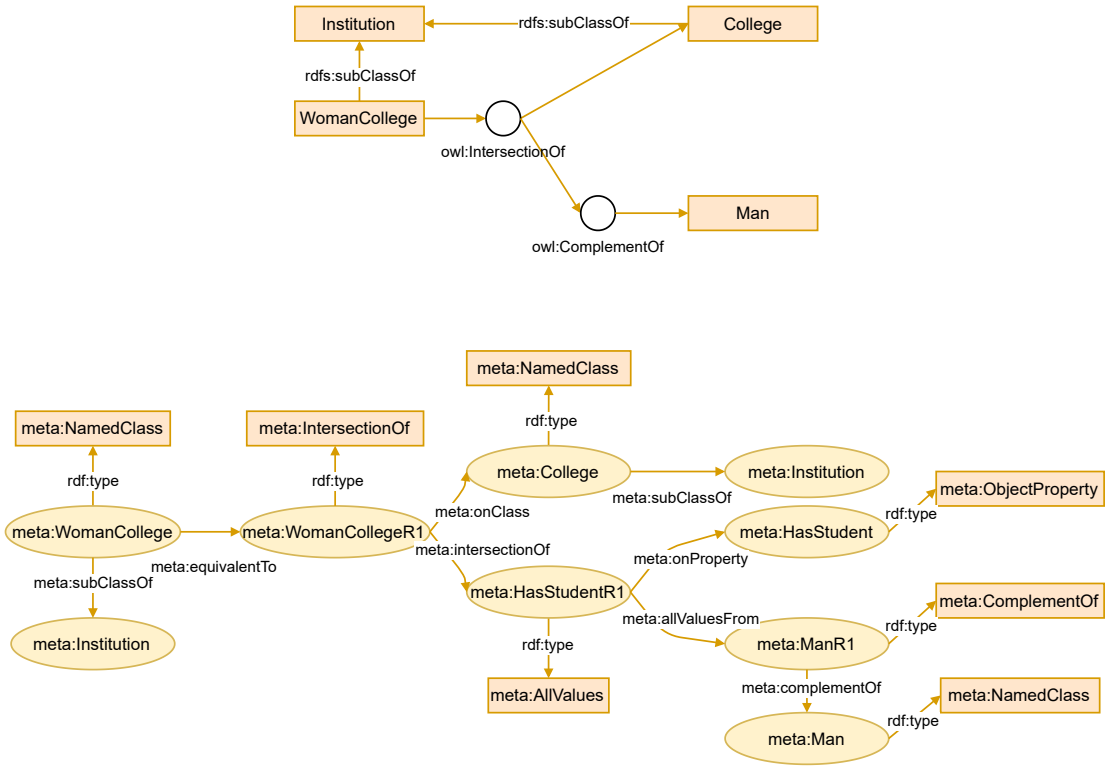

Figure 6.4: In the example above, a woman college is a college where every student is not male. On the top, a " WomanCollege" class is defined as an intersection of "Institution" and the complement of "Man". On the bottom, the same OWL class descriptions are represented in META.

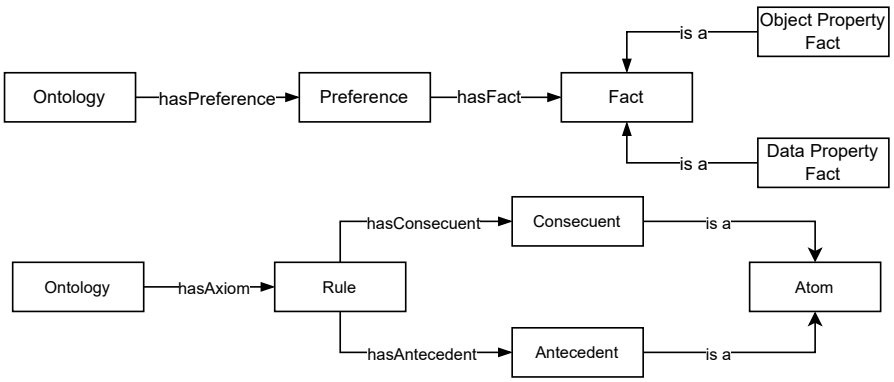

Figure 6.5: META ontology preferences (top) and rules (bottom) hierarchies.

straightforward command-line tool. This process simplifies the conversion of intricate ontological structures into META's framework, making the ontology more manageable and accessible. In addition to importing and mapping features, the

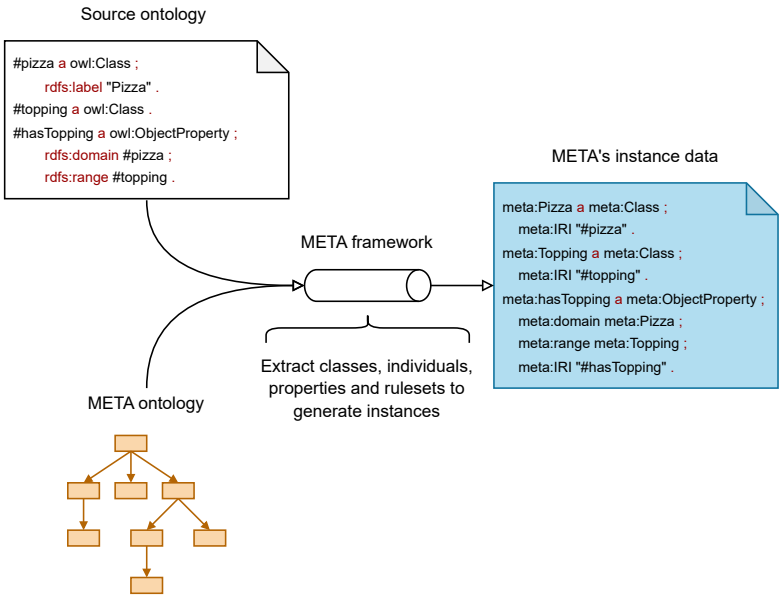

Figure 6.6: Mapping process facilitated by the framework, showcasing the conversion of an existing ontology into instance data using META. The framework enables users to easily map the ontology's conceptual elements, such as classes, properties, relationships, and SWRL rules, to their corresponding instances within the META ontology.

tool also supports the reverse process. This feature allows users to revert META's instances back into their original ontology form. It provides the flexibility to reconstruct the original ontology, preserving its structural and semantic integrity, which can be particularly useful in applications requiring direct interaction with the original ontology data. To illustrate the functionality, let's consider an example code snippet in Python using the proposed framework (Code Listing 11).

```
from rdflib import RDF

from meta_framework import KEI

kei = KEI()
kei.base = "meta.owl"
kei.ontology_path = "ontology.owl"

kei.import_ontology()

instances = kei.map_to_meta()

# Perform operations on the instances
#  (e.g., query, add, update, delete)

# Reverse process: reconstruct the original ontology
kei.save(instances, output="ontology.owl")
```

Listing 11: Example usage of the framework in Python.

Furthermore, the framework provides functionalities for ontology validation, consistency checking, and reasoning. It leverages the reasoning capabilities of OWL to infer implicit knowledge and detect inconsistencies within ontologies. This ensures that the converted ontologies conform to the guidelines and constraints defined by the META ontology.

6.3 Use cases

In this section, we evaluate the META ontology in terms of its functional capabilities, adaptability, efficiency in knowledge representation, and its impact on data analysis and reasoning tasks. To this end, we employ several experimental scenarios and use cases that simulate real-world knowledge engineering tasks.

6.3.1 Ontology driven data analytics

Ontology-driven approaches have proven to be useful in various data analysis tasks. For example, in Roldán-García *et al.* 2021 the authors proposed an ontology-driven approach to support the meta-modelling, selection, and reasoning of Key Performance Indicators (KPIs). In the original study, the authors suggested the use of an ontology, specifically KPIOWL, for the purpose of semantically validating the results obtained from the correlation analysis. This process is depicted in Figure 6.7. It starts with the collection of raw data. Next, the relationships and dependencies between different variables in the dataset are calculated. The correlation analysis helps identify potential patterns, associations, and dependencies among the data elements.

Acting as a knowledge representation model, the KPIOWL ontology captures domain-specific concepts, relationships, and rules pertinent to Key Performance Indicators. By leveraging this ontology, the authors can ensure that the identified correlations are contextually meaningful and align with the predefined domain

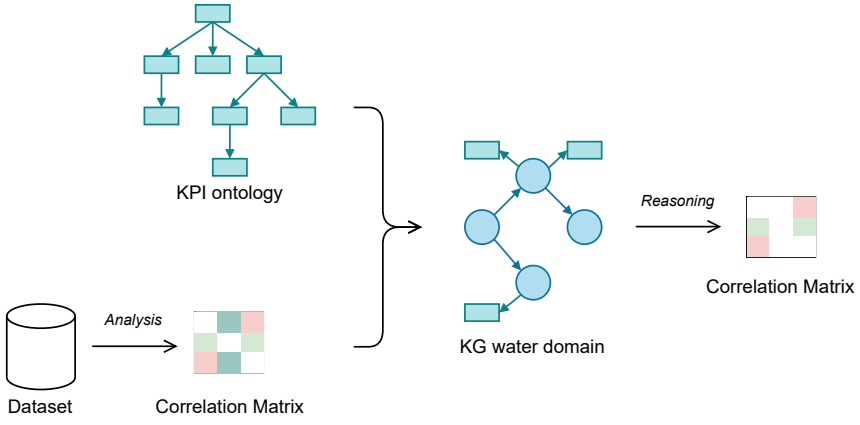

Figure 6.7: Ontology-driven approach for correlation analysis of Key Performance Indicators (KPIs). The process involves collecting data, performing correlation analysis, and utilizing the KPIOWL ontology to ensure contextually meaningful correlations aligned with domain-specific knowledge.

knowledge. Following the semantic validation, the authors draw further insights and conclusions from the data, such as inconsistencies in the correlation analysis.

```
Rule: monitors(?i2, ?o) ^ monitors(?i1, ?o)
    -> redundant(?i1, ?i2)
```

Listing 12: SWRL rule to infer redundant monitors.

Our META ontology has the potential to further enhance this process, facilitating more efficient and accurate validation of results derived from data analysis. In our refined approach, classes, properties and relationships, including SWRL rules, are mapped to instances within META. For example, the KPIOWL ontology defines a set of rule to infer relationships between indicators from the model. The SWRL rule outlined in Listing 12 indicates that when two distinct indicators are monitoring the same goal, a potential redundancy may exist in the relationship between these indicators. This rule is annotated in META as shown in Figure 6.8.

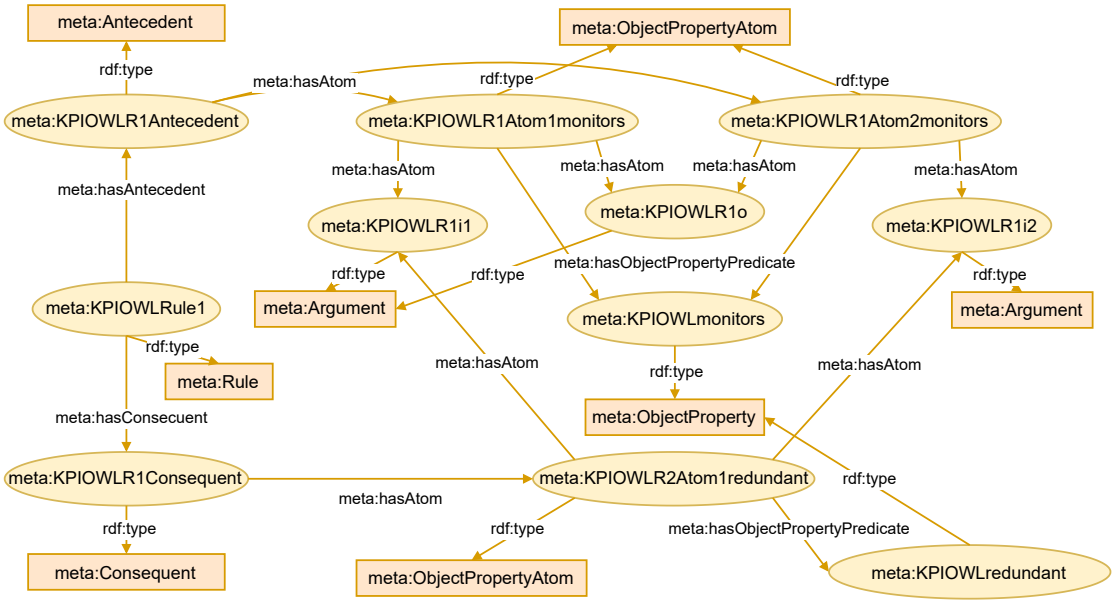

Figure 6.8: SWRL rule annotated using the META ontology.

We can also incorporate additional statements to characterise complex relationships between various data elements. Specifically, the object property `kpiowl:indicatorDirect`, which denotes a correlation between distinct Key Performance Indicators (KPIs), can be mapped to the construct `meta:Correlation` within the META ontology. During this process, the inherent semantic depth of the original object property is not only retained but also extended within the con-

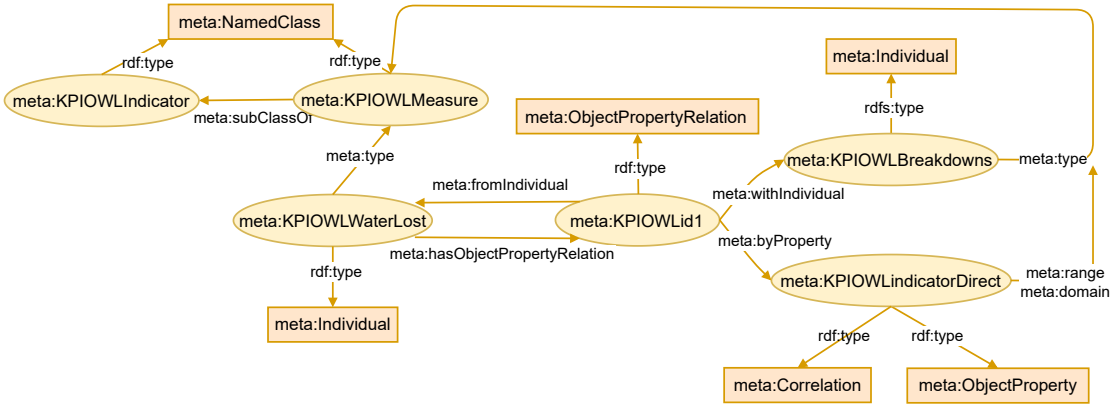

Figure 6.9: META instances for KPIOWL ontology.

text of the META ontology (see Figure 6.9). The implementation of this process in Python can be demonstrated as shown in Code Listing 13.

```
from rdflib import RDF

from meta_framework import KEI, meta

kei = KEI()
kei.base = "meta.owl"
kei.ontology_path = "KPIOWL.owl"

kei.import_ontology(infer_triples=True)

instances = kei.map_to_meta()

# Add a new statement to specify that the object property
#  `indicatorDirect' from `KPIOWL` denotes a correlation.
indicator = URIRef(meta + "KPIWOLindicatorDirect")
instances.add((indicator, RDF.type, meta.Correlation)
```

Listing 13: Python code for the enhanced ontology-driven correlation analysis.

The provided Code Listing 14 illustrates the process of calculating the correlation matrix using the set of instances generated in Code Listing 13. Note that the methodology for computing the correlation matrix is generic, making it applicable to any domain ontology loaded through the META framework. In this case, data and restrictions (coming from the expert knowledge) are provided to the algorithm, directly injecting the knowledge in the calculation process. Thus, the results of this processing step will not produce results that are incoherent with the experts' knowledge.

Furthermore, in our approach the validation process can be conducted earlier in the workflow, potentially reducing the number of false positives generated by the correlation analysis. As a result, the data sets can be reduced before con-

```
import pandas as pd
from rdflib import RDF

from meta_framework import meta

data = pd.read_csv("path/to/data.csv")

pairs = set()
for o in instances.subjects(RDF.type, meta.ObjectPropertyRelation):
    for p in instances.objects(o, meta.byProperty):
        if instances.value(p, RDF.type, meta.Correlation):
            variables = set()
            variables.add(instances.value(o, meta.fromIndividual, None).split("#")[1])
            variables.add(instances.value(o, meta.withIndividual, None).split("#")[1])
            pairs.add(tuple(variables))

# Compute correlation matrix.
corr = pd.DataFrame()

for col1, col2 in list(combinations(data.columns, 2)):
    if (col1, col2) in pairs or (col2, col1) in pairs:
        c = -1.0
    else:
        c = df[col1].corr(df[col2], method="pearson")
    corr.at[col1, col2] = c
    corr.at[col2, col1] = c
```

Listing 14: The code identifies specific pairs of variables for which the correlation should be skipped, and then computes the correlation matrix based on the remaining variable pairs.

ducting further analyses, thereby reducing the required computational resources and saving time while maintaining the same level of accuracy. This example illustrates the flexibility and extensibility of the META framework, showing how it allows for a seamless transition from an existing ontology to a META-structured ontology, while enhancing the semantic richness of the represented data.

6.3.2 Injecting domain knowledge to machine learning algorithms

In practice, injecting domain knowledge involves incorporating additional information about the problem domain into the machine learning algorithm. This can take various forms, such as including domain-specific features in the input data or using domain-specific constraints in the learning process.

In Barba-González, Nebro, *et al.* 2021, the authors proposed a method to inject domain knowledge into multi-objective optimisation algorithms. To achieve this, they developed an ontology, called KPIOWL[3], in the urban domain to represent the problem and the constraints that must be satisfied. KPIOWL cap-

[3]In URL: `https://github.com/KhaosResearch/KPIOWL`

tures concepts such as Goals (representing business objectives and their interrelations), Relationships (modeling connections between goals with subcategories like Contribution and Decomposition), and Indicators (which monitor goals and have subclasses like Measure, KRI, and KPI). They then integrated this ontology into the optimisation process to generate a set of feasible solutions. While this approach shows promise, it does have some limitations that can be improved upon. One major limitation of their method is its tight coupling to the urban domain. While this specialisation can be advantageous in specific contexts, it restricts the approach's applicability to only urban-related optimisation problems.

Following a similar approach as described before, we can use the META ontology to incorporate domain knowledge in any type of optimisation algorithms, including machine learning. In particular, we can leverage the META ontology to provide contextual information and domain-specific constraints to the algorithms, allowing them to make better-informed decisions and achieve more accurate results. By integrating this domain-specific semantics, we effectively reduce the search space of the problem, which can potentially improve both the runtime and the quality of the solutions generated by the algorithm (Pei and Han 2000). Additionally, the inclusion of domain knowledge aids in guiding the algorithm towards more feasible and practical solutions, as it aligns the algorithm's processing with the intrinsic characteristics and limitations of the domain. Consequently, this alignment not only enhances the efficiency of the algorithm but also increases the likelihood of generating solutions that are not just theoretically optimal but also practically applicable and meaningful within the specific domain.

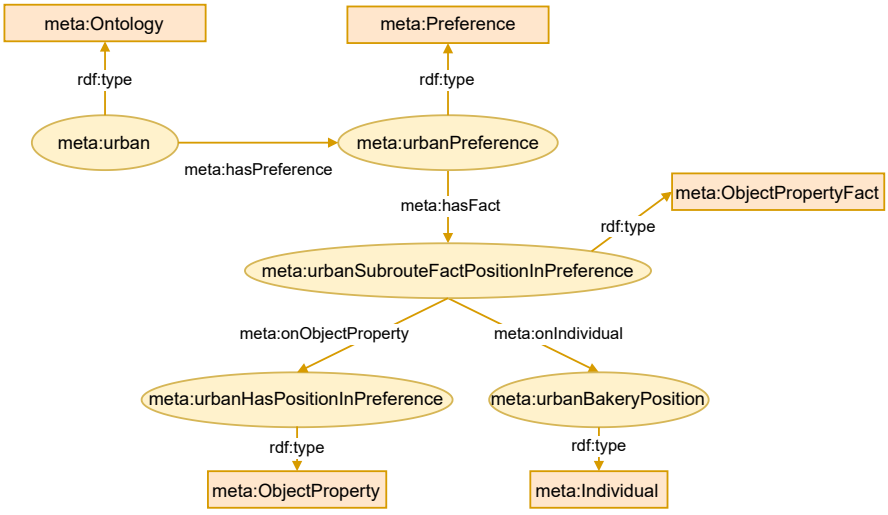

Figure 6.10: META instances for Urban ontology.

For example, a sub-route preference, introducing a sequencing constraint typical of TSP scenarios, indicates that a deliverer must go to a given restaurant after visiting a bakery. This constraint can be expressed in terms of the Urban ontology by defining an `urban:Preference` with a pair of `urban:hasPositionInPreference` relationships linking the individuals. The process is summarised in Code Listing 15. Initially, the input ontology is aligned with the META ontology through a mapping process as shown in Figure 6.10. Next, constraints are added to the ontology using `meta:Preference` individuals, which will later aid the optimisation algorithm in generating feasible solutions.

```
kei = KEI()
kei.base = "meta.owl"
kei.ontology_path = "urban.owl"

kei.import_ontology()

instances = kei.map_to_meta()

# Add simple position preference.
preference = URIRef(meta + "urbanPreference")
instances.add((meta.urban, meta.hasPreference, preference))
instances.add((preference, RDF.type, meta.Preference))

bakeryPosition = URIRef(meta + "urbanBakeryPosition")
fact = URIRef(meta + "urbanSubrouteFactPositionInPreference")
instances.add((preference, meta.hasFact, fact))
instances.add((fact, RDF.type, meta.ObjectPropertyFact))
instances.add((fact, meta.onIndividual, bakeryPosition))
instances.add((fact, meta.onObjectProperty, meta.hasPositionInPreference))
```

Listing 15: Python code for injecting user preferences in Machine Learning algorithms.

Code Listing 16 demonstrates how to use the set of instances generated earlier to extract a set of constraints. In the code snippet, we iterate through the set of preferences defined in the ontology and extract the associated facts. These facts represent specific conditions or criteria that need to be satisfied for generating optimal solutions. For each preference, we identify the relevant variables and their corresponding values that are essential for guiding the optimisation process. Once the constraints are extracted, they can be used in different ways depending on the type of machine learning algorithm being employed. For instance, in optimisation algorithms, the extracted constraints can be integrated as objective functions or constraints to influence the optimisation process. In classification tasks, the constraints can serve as rules to guide the decision boundaries and improve the model's interoperability.

```
constraints = set()
for preference in instances.objects(None, meta.hasPreference):
    print("Declared preference", preference)

    for fact in instances.objects(preference, meta.hasFact):
        on_individual = instances.value(fact, meta.onIndividual, None)

        if (fact, RDF.type, meta.ObjectPropertyFact) in instances:
            variable = instances.value(fact, meta.onObjectProperty, None)
        else:
            variable = instances.value(fact, meta.onDataProperty, None)

        constraints.add((preference, variable, on_individual))
        print("  Fact:", fact, variable, on_individual)
```

Listing 16: Extracting constraints from ontology domain knowledge.

Regardless of the nature of the input ontology, the core methodology remains the same–the aforementioned process is universal and can be applied to any input ontology, making it a highly versatile and adaptable framework.

Chapter 7

Conclusions and Future Work

The progression of Big Data has changed the data processing landscape and imposed challenges to unify knowledge from diverse sources. As addressed in this thesis, domain semantics have made significant strides in data analysis. However, challenges remain, particularly in the seamless integration of knowledge from heterogeneous sources and tailoring algorithmic design based on this knowledge.

The central theme of this Thesis revolves around understanding the role of domain semantics in facilitating the integration of knowledge in Big Data processes. The goal was to explore ways to extract, represent, and inject knowledge from linked sources seamlessly, ensuring that it is meaningful and actionable. As a result, we have addressed the Objectives set out in Chapter 1 as indicated in the following sections.

7.1 Conclusions

7.1.1 Workflows in Big Data

With the exponential growth of Big Data, creating semantic-based solutions that are inherently scalable is imperative. Furthermore, knowledge-based systems underscore the significance of interlace domain semantics with data processing, providing scalable solutions and meaningful insights from vast datasets. Recognizing this, we introduced TITAN (**Ch3**), a Workflow Management System (WMS) platform explicitly designed to address these complexities. TITAN platform leverages the principles of domain semantics to guide smart workflow designs and validation (**O.1**). TITAN's architecture is designed to be inherently scalable, making it a robust solution for the diverse challenges posed by Big Data.

7.1.2 Scalable reasoning

In tandem, we developed NORA (**Ch4**), a scalable reasoning system anchored on Big Data technologies like Apache Spark and Cassandra. NORA uses a materialization strategy for reasoning. This method produces new data, or *inferences*, and put them back into the Knowledge Base (i.e., Apache Cassandra database). The inference rules implementation follows a fix point approach using Apache Spark, thus the reasoning process is evaluated iteratively until no new inferred knowledge is derived.

Our choice of NoSQL for storage was intentional to provide scalability and reliability to the system. However, it is worth noting that we had to compensate for the absence of some relational features found in other SQL systems, such as join operations, by integrating Apache Spark, at the expense of increased computational time.

Nevertheless, NORA can deal with massive datasets without compromising performance. It also complements TITAN by enhancing its capabilities (**O.2**). TITAN streamlines workflow management and integrates semantically rich data. In contrast, NORA provides the computing power to conduct deep, scalable reasoning over these workflows when necessary.

We have validated our approach using the Lehigh University Benchmark and the University Ontology Benchmark. The initial experimentation shows promising results regarding scalability and execution time and completeness. Our work is open-source, so it is open to improvement by the community.

7.1.3 Biological ontologies

We have also introduced SALON (**Ch5**), an ontology designed to describe biological sequence alignments. This includes not only particular terms related to multiple alignments but also the description and conceptualization of specific algorithms to perform these alignments. Specifically, SALON offers a structured framework for the reliable interpretation and application of sequence alignments acting as a pivotal tool for knowledge injection in this domain (**O.3**). Identifying biological sequences is probably one of the most critical tasks in bioinformatics. Sequence alignments have many applications, such as gene finding, gene function prediction, or genome sequence assembly. Therefore, the availability of biological databases with information about correct and complete sequence alignments is particularly interesting to scientists and researchers in this area. Annotating biological sequence alignment (and its associated information) with an ontology allows the development of linked data repositories, providing homogeneous access to heterogeneous information and enabling intelligent and advanced applications.

The SALON ontology could act as a mediated schema for virtual data integration from different sources using federated SPARQL queries. To validate sequence alignments, SALON can be further exploited by defining SWRL rules, which automatically determine if a sequence alignment is plausible based on its global assigned score. This feature has been tested in the BAliBASE dataset but can be applied according to the scientists' data. Moreover, SALON not only stands as a foundational framework for defining and developing algorithms with semantic awareness but also plays a crucial role in enhancing TITAN's capabilities. By harnessing the domain-specific annotations provided by SALON, TITAN is able to refine its workflow management and data processing capabilities more effectively.

7.1.4 Injection of semantics in algorithms

It is noteworthy that neither of the aforementioned systems inherently considers the algorithmic implementation. Tailoring algorithms based on domain semantics ensures efficient data processing and guarantees that the results align with the domain-specific knowledge and requirements.

In light of this, we have worked on a methodology to inject semantic knowledge into Big Data analysis algorithms (**O.4**). Our approach utilizes the META ontology. Within this framework, we model concepts derived from the Web Ontology Language (OWL), such as classes and properties, user restrictions and preferences, as OWL elements. By doing so, we aim to offer a mechanism to

integrate knowledge from various information sources and enable its reuse in different applications.

This ontology makes it feasible to automate the process of loading domain ontologies into a standardized framework. Once loaded, this information becomes readily available for use in various algorithms. This way, processes can be enriched with specifics of the domain, enabling it to make more informed decisions. For example, in the case of smart cities, the detailed knowledge about city infrastructures, traffic patterns, and zoning laws, can be integrated through META, greatly enhancing the performance and accuracy of predictive models. Moreover, META annotations can also provide constraints or preferences that guide algorithms' learning or optimisation process. These constraints can act as rules, ensuring that the solutions or outcomes produced by the algorithms align with expert knowledge and domain-specific considerations.

META has been validated through the presentation of two use cases demonstrating its effectiveness in improving data analysis. In the first one, we performed a more refined, contextually meaningful correlation analysis that aligns closely with the domain knowledge. Furthermore, by transitioning from the existing KPIOWL ontology to our META-structured ontology, we not only retained the inherent semantic depth of the original ontology but also enriched the semantic representation of the data. In the second use case, we explored the potential of the META ontology in injecting domain knowledge into machine learning algorithms. Through the META ontology, we provided a systematic way to map domain knowledge and constraints directly into machine learning models, which can then guide the learning process to produce more informed, accurate, and interpretable results. We illustrated this with a practical example where user preferences were integrated into a machine learning algorithm to enhance its decision-making capability in the urban domain.

Finally, to assess the compliance of the META ontology against the FAIR principles (Wilkinson, Dumontier, *et al.* 2016), we have used the FOOPS! validator[1], a web service for detecting best practices according to each FAIR principle, scoring the highest mark in 20 out of 24 tests.

[1]In URL: `https://foops.linkeddata.es/FAIR_validator.html`

7.2 Future Work

While this thesis has successfully addressed the initial objectives outlined in Chapter 1, there are still some challenges to be studied. They are mainly in the intersection of the proposed objectives and provide a clear postdoctoral research line, as described below:

- **Enhancing TITAN's Semantic Capabilities:** While TITAN has laid the groundwork for semantically-guided workflows, future iterations of the platform will incorporate advanced semantic reasoning capabilities, potentially integrating more closely with reasoning systems like NORA.

  A promising step in this direction is the incorporation of the META ontology. META provides a structured way to represent algorithmic knowledge and preferences. We aim to integrate META within TITAN, making the system more knowledgeable about algorithmic semantics, and leading to more intelligent and efficient workflows.

- **Versioning NORA with other storage layers:** We intent to evaluate alternative storage layers, allowing users to choose the most suitable option for materialization during the reasoning process. It includes examining drop-in replacements for Cassandra, such as ScyllaDB[2]. Additionally, we intend to assess the performance of SQL systems with join operators, to determine the efficiency and effectiveness of such approaches in the context of our reasoner. Exploring alternative storage solutions, like MongoDB, for NORA can offer benefits in terms of flexibility and performance. By versioning NORA with MongoDB, we aim to achieve more efficient storage and querying capabilities, especially for complex semantic datasets.

- **Extending NORA to more rich semantics:** We will continue to develop new rule sets to support semantically richer fragments of OWL, including OWL DL. This encompasses remodelling existing rule sets to enhance the system's performance, identify bottlenecks, and optimize the reasoning process.

- **Extending NORA with SWRL Support:** The Semantic Web Rule Language (SWRL) is a powerful tool for expressing rules in ontologies. We aim to extend NORA's capabilities to natively support and process SWRL rules. This would allow for more expressive knowledge representation and reasoning on the platform.

- **Integrating SALON with NORA post-SWRL Support:** In future work, our

[2]In URL: `https://www.scylladb.com`

plans include creating a novel database on sequence alignments to enable an easy way to use this software solution by users with limited technical knowledge. This includes defining new classes and properties to describe sequence alignments' construction methods in collaboration with domain experts. Furthermore, once NORA achieves full SWRL support, there is an opportunity to test SALON within this extended environment. This will validate both systems and showcase how semantically rich domain-specific knowledge can be processed at scale.

- **Development of User-Friendly Interfaces for META:** As future work, we are exploring the development of user-friendly interfaces and tools that can make the META ontology more accessible to a broader audience. This initiative aims to enable domain experts, even those with limited technical background, to contribute to, modify, and use the META ontology in their data analytics projects.
- **META methodology refinement:** Our work will also focus on refining the methodology presented in META, and ensuring that the injection of semantics into algorithms is seamless. We will also look into how these semantically-aware algorithms can be integrated into platforms like TITAN and NORA, enhancing their capabilities and expanding their application areas.

List of Figures

List of Tables

Bibliography

1. P. Buitelaar, P. Cimiano, B. Magnini, Eds., *Ontology Learning from Text: Methods, Evaluation and Applications* (IOS Press, Amsterdam, 2005), vol. 123, ISBN: 978-1-58603-523-5.
2. E. Peis, M.-d.-C. J. M., D.-L. J. A., *Hipertext.net*, (https://raco.cat/index.php/Hipertext/article/view/131957) (May 2009).
3. A. Pinto *et al.*, presented at the Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015), pp. 324–327.
4. N. Phan *et al.*, presented at the Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics, pp. 433–442, ISBN: 9781450338530, (https://doi.org/10.1145/2808719.2808764).
5. C. Diamantini, D. Potena, E. Storti, presented at the, pp. 13–24.
6. J. Kietz, F. Serban, A. Bernstein, S. Fischer (Sept. 2010).
7. M. Žaková, P. Křemen, F. Železný, N. Lavrač, *IEEE Transactions on Automation Science and Engineering* **8**, 253–264 (2011).
8. L. Parody, M. T. Gómez-López, R. M. Gasca, *Information and Software Technology* **70**, 140–154, ISSN: 0950-5849, (https://www.sciencedirect.com/science/article/pii/S0950584915001731) (2016).
9. C. Barba-González, J. García-Nieto, M. del Mar Roldán-García, *et al.*, *Expert Systems with Applications* **115**, 543–556, ISSN: 0957-4174, (https://www.sciencedirect.com/science/article/pii/S0957417418305347) (2019).
10. M. d. M. Roldan-Garcia, J. F. Aldana-Montes, presented at the, pp. 174–179.
11. Z. A. Al-Sai, R. Abdullah, M. h. husin, presented at the 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT), pp. 150–155.
12. D. Dou, H. Wang, H. Liu, presented at the Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015), pp. 244–251.
13. P. K. Sinha *et al.*, *Data Technol. Appl.* **56**, 172–204, (https://api.semanticscholar.org/CorpusID:240525018) (2021).

14. C. Lemke, M. Budka, B. Gabrys, *Artificial Intelligence Review* **44**, 117–130, ISSN: 1573-7462, (https://doi.org/10.1007/s10462-013-9406-y) (June 2015).
15. P. Nguyen, M. Hilario, A. Kalousis, *J. Artif. Int. Res.* **51**, 605–644, ISSN: 1076-9757 (Sept. 2014).
16. C. M. Keet *et al.*, *Journal of Web Semantics* **32**, 43–53, ISSN: 1570-8268, (https://www.sciencedirect.com/science/article/pii/S1570826815000025) (2015).
17. A. Benítez-Hidalgo, C. Barba-González, *et al.*, *Knowledge-Based Systems* **232**, 107489, ISSN: 0950-7051, (https://www.sciencedirect.com/science/article/pii/S0950705121007516) (2021).
18. A. Benítez-Hidalgo, I. Navas-Delgado, M. d. M. Roldán-García, *Software: Practice and Experience* **n/a**, eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.3258, (https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3258) (2023).
19. A. Benítez-Hidalgo, J. F. Aldana-Montes, I. Navas-Delgado, M. d. M. Roldán-García, *BMC Bioinformatics* **24**, 69, ISSN: 1471-2105, (https://doi.org/10.1186/s12859-023-05190-7) (Feb. 2023).
20. A. Neilson, Indratmo, B. Daniel, S. Tjandra, *Big Data Research*, ISSN: 2214-5796 (2019).
21. W. Raghupathi, V. Raghupathi, *Health Information Science and Systems* **2**, 3, ISSN: 2047-2501 (Feb. 2014).
22. I. J. Taylor, E. Deelman, D. B. Gannon, M. Shields, *et al.*, *Workflows for e-Science: scientific workflows for grids* (Springer, 2007), vol. 1.
23. S. Wolfert, L. Ge, C. Verdouw, M.-J. Bogaardt, *Agricultural Systems* **153**, 69–80, ISSN: 0308-521X (2017).
24. C. Barba-González, J. García-Nieto, M. Roldán-García, *et al.*, *Expert Systems with Applications* **115**, 543–556, ISSN: 0957-4174 (2019).
25. Y. Gil, V. Ratnakar, *et al.*, *IEEE Intelligent Systems* **26** (2011).
26. F. Gessert, W. Wingerath, S. Friedrich, N. Ritter, *Comput. Sci.* **32**, 353–365, ISSN: 1865-2034, (https://doi.org/10.1007/s00450-016-0334-3) (July 2017).
27. A. Lakshman, P. Malik, *SIGOPS Oper. Syst. Rev.* **44**, 35–40, ISSN: 0163-5980, (https://doi.org/10.1145/1773912.1773922) (Apr. 2010).
28. M. Bhandarkar, presented at the 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), pp. 1–1.
29. M. Zaharia, M. Chowdhury, M. J. Franklin, *et al.*, presented at the Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, p. 10.

30. M. Zaharia, M. Chowdhury, T. Das, *et al.*, presented at the Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, p. 2.
31. R. Mitchell *et al.*, presented at the 2019 IEEE International Conference on Big Data (Big Data), pp. 4537–4544.
32. C. E. Scheidegger *et al.*, presented at the Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 1251–1254, ISBN: 978-1-60558-102-6.
33. J. Freire *et al.*, presented at the International Provenance and Annotation Workshop, pp. 10–18.
34. T. M. McPhillips *et al.*, *CoRR* **abs/1502.02403**, arXiv: 1502.02403, (http://arxiv.org/abs/1502.02403) (2015).
35. M. R. Berthold *et al.*, *AcM SIGKDD explorations Newsletter* **11**, 26–31 (2009).
36. *Apache Airflow Documentation*, 2019, (https://airflow.apache.org/).
37. K. Wolstencroft *et al.*, *Nucleic Acids Research* **41**, 557–561 (2013).
38. B. T. G. S. Kumara *et al.*, in *2015 IEEE International Conference on Web Services*, pp. 495–502.
39. Y. Gil, K. Cobourn, *et al.* (2018).
40. S. D. Peckham, presented at the International Congress on Environmental Modelling and Software.
41. M. Albrecht, P. Donnelly, P. Bui, D. Thain, presented at the Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies, pp. 1–13.
42. T. Tannenbaum, D. Wright, K. Miller, M. Livny, in *Beowulf Cluster Computing with Windows*, pp. 307–350.
43. M. D. Wilkinson, M. Links, *Briefings in bioinformatics* **3**, 331–341 (2002).
44. M. Atkinson, S. Gesing, J. Montagnat, I. Taylor, *Scientific workflows: Past, present and future*, 2017.
45. V. K. Vavilapalli *et al.*, presented at the Proceedings of the 4th annual Symposium on Cloud Computing, pp. 1–16.
46. J. A. Novella *et al.*, *Bioinformatics* **35**, 839–846 (2019).
47. M. Palankar, A. Iamnitchi, M. Ripeanu, S. Garfinkel, presented at the Proceedings of the 2008 international workshop on Data-aware distributed computing, pp. 55–64.
48. B. Wilder, *Cloud architecture patterns: using microsoft azure* (" O'Reilly Media, Inc.", 2012).
49. A. K. Mishra, J. L. Hellerstein, W. Cirne, C. R. Das, *ACM SIGMETRICS Performance Evaluation Review* **37**, 34–41 (2010).
50. P. Di Tommaso *et al.*, *Nature biotechnology* **35**, 316–319 (2017).
51. D. Merkel, *Linux journal* **2014**, 2 (2014).

52. G. M. Kurtzer, *Singularity 2.1. 2-Linux application and environment containers for science*, 2016.
53. V. Korkhov *et al.*, *Scientific Programming* **15**, 173–188 (2007).
54. Y. L. Simmhan, B. Plale, D. Gannon, *International Journal of Web Services Research (IJWSR)* **5**, 1–22 (2008).
55. P. Hitzler *et al.* (Jan. 2012).
56. S. Bechhofer *et al.*, "OWL Web Ontology Language Reference", Recommendation, http://www.w3.org/TR/owl-ref/ (World Wide Web Consortium (W3C), Feb. 2004).
57. E. Sirin *et al.*, *Journal of Web Semantics* **5**, Software Engineering and the Semantic Web, 51–53, ISSN: 1570-8268, (https://www.sciencedirect.com/science/article/pii/S1570826807000169) (2007).
58. R. Shearer, B. Motik, I. Horrocks, presented at the.
59. V. Haarslev, K. Hidde, R. Möller, M. Wessel, *Semantic Web* **3** (July 2012).
60. I. Horrocks, P. F. Patel-Schneider, S. Bechhofer, D. Tsarkov, *Web Semantics: Science, Services and Agents on the World Wide Web* **3**, 23–40, ISSN: 1570-8268 (2005).
61. R. McClatchey *et al.*, *Designing Traceability into Big Data Systems*, 2015, arXiv: 1502.01545 [cs.DB].
62. T. A. S. Siriweera, I. Paik, B. T. Kumara, presented at the 2017 IEEE International Conference on Services Computing (SCC), pp. 116–123.
63. T. Akila, I. Paik, S. Siriweera, *IEEE Access* **6**, 48797–48814 (2018).
64. M. del Mar Roldan-Garcia, J. Aldana-Montes, presented at the 22nd International Conference on Data Engineering Workshops (ICDEW'06), pp. 58–58.
65. G. Antoniou *et al.*, *The Knowledge Engineering Review* **33**, e21 (2018).
66. R. Gu *et al.*, presented at the, pp. 700–709.
67. Y. Liu, P. McBrien, presented at the Proceedings of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond, ISBN: 9781450350198, (https://doi.org/10.1145/3070607.3070609).
68. H. Mohamed., S. Fathalla., J. Lehmann., H. Jabeen., presented at the Proceedings of the 13th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - KEOD, pp. 51–60, ISBN: 978-989-758-533-3.
69. G. Ladwig, A. Harth, presented at the.
70. O. Curé *et al.*, presented at the, pp. 166–173.
71. F. Michel, C. Faron Zucker, J. Montagnat, in *Transactions on Large-Scale Data- and Knowledge-Centered Systems XL* (Springer, Jan. 2019), pp. 125–165, (https://hal.archives-ouvertes.fr/hal-01926379).

72. L. Reyes-Álvarez, M. d. M. Roldan, J. Aldana Montes, *Software: Practice and Experience* **49** (Oct. 2018).
73. Y. Guo, Z. Pan, J. Heflin, *Journal of Web Semantics* **3**, Selected Papers from the International Semantic Web Conference, 2004, 158–182, ISSN: 1570-8268, (https://www.sciencedirect.com/science/article/pii/S1570826805000132) (2005).
74. L. Ma *et al.*, presented at the, vol. 4011, pp. 125–139, ISBN: 978-3-540-34544-2.
75. J. Thompson *et al.*, *Nucleic acids research* **33**, 4164–71 (Feb. 2005).
76. The UniProt Consortium, *Nucleic Acids Research* **47**, D506–D515, ISSN: 0305-1048, eprint: https://academic.oup.com/nar/article-pdf/47/D1/D506/27437297/gky1049.pdf, (https://doi.org/10.1093/nar/gky1049) (Nov. 2018).
77. K. Clark *et al.*, *Nucleic Acids Research* **44**, D67–D72, ISSN: 0305-1048, eprint: https://academic.oup.com/nar/article-pdf/44/D1/D67/9483609/gkv1276.pdf, (https://doi.org/10.1093/nar/gkv1276) (Nov. 2015).
78. D. McSkimming *et al.*, *Molecular bioSystems* **12** (Oct. 2016).
79. N. Noy, D. Mcguinness, *Knowledge Systems Laboratory* **32** (Jan. 2001).
80. J. D. Thompson, P. Koehl, R. Ripp, O. Poch, *Proteins: Structure, Function, and Bioinformatics* **61**, 127–136, eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/prot.20527, (https://onlinelibrary.wiley.com/doi/abs/10.1002/prot.20527) (2005).
81. J. D. Thompson, A. Muller, *et al.*, *BMC Bioinformatics* **7**, 318, ISSN: 1471-2105, (https://doi.org/10.1186/1471-2105-7-318) (Jan. 2006).
82. C. E. Shannon, *The Bell System Technical Journal* **27**, 623–656 (1948).
83. O. Penn *et al.*, *Molecular Biology and Evolution* **27**, 1759–1767, ISSN: 0737-4038, eprint: https://academic.oup.com/mbe/article-pdf/27/8/1759/3112601/msq066.pdf, (https://doi.org/10.1093/molbev/msq066) (Mar. 2010).
84. G. Landan, D. Graur, *Molecular Biology and Evolution* **24**, 1380–1383, ISSN: 0737-4038, eprint: https://academic.oup.com/mbe/article-pdf/24/6/1380/5805188/msm060.pdf, (https://doi.org/10.1093/molbev/msm060) (Mar. 2007).
85. C. Kemena, J.-F. Taly, J. Kleinjung, C. Notredame, *Bioinformatics* **27**, 3385–3391, ISSN: 1367-4803, eprint: https://academic.oup.com/bioinformatics/article-pdf/27/24/3385/16902900/btr587.pdf, (https://doi.org/10.1093/bioinformatics/btr587) (Oct. 2011).
86. G. Foley *et al.*, *BioTechniques* **67**, PMID: 31218882, 50–54, eprint: https://doi.org/10.2144/btn-2018-0188, (https://doi.org/10.2144/btn-2018-0188) (2019).

87. K. M. Wong, M. A. Suchard, J. P. Huelsenbeck, *Science* **319**, 473–476, eprint: https://www.science.org/doi/pdf/10.1126/science.1151532, (https://www.science.org/doi/abs/10.1126/science.1151532) (2008).
88. E. Privman, O. Penn, T. Pupko, *Molecular Biology and Evolution* **29**, 1–5, ISSN: 0737-4038, eprint: https://academic.oup.com/mbe/article-pdf/29/1/1/13647607/msr177.pdf, (https://doi.org/10.1093/molbev/msr177) (July 2011).
89. S. Flügel, M. Glauer, F. Neuhaus, J. Hastings, *When one Logic is Not Enough: Integrating First-order Annotations in OWL Ontologies*, Oct. 2022.
90. M. Lenzerini, L. Lepore, A. Poggi, *Artificial Intelligence* **292**, 103432, ISSN: 0004-3702, (https://www.sciencedirect.com/science/article/pii/S0004370218301875) (2021).
91. D. Vrandečić *et al.*, presented at the, pp. 109–123.
92. W. Peters, E. Montiel Ponsoda, G. Aguado de Cea, A. Gómez-Pérez, presented at the Actas OntoLex 2007, Ontology Engineering Group - OEG, (https://oa.upm.es/5195/).
93. M. Roldán-García *et al.*, *International Journal of Information Management* **58**, 102018, ISSN: 0268-4012, (https://www.sciencedirect.com/science/article/pii/S026840121930581X) (2021).
94. C. Barba-González, A. J. Nebro, *et al.*, *Computer Standards & Interfaces* **78**, 103546, ISSN: 0920-5489, (https://www.sciencedirect.com/science/article/pii/S0920548921000416) (2021).
95. J. Pei, J. Han, presented at the Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 350–354, ISBN: 1581132336, (https://doi.org/10.1145/347090.347166).
96. M. D. Wilkinson, M. Dumontier, *et al.*, *Scientific Data* **3**, 160018, ISSN: 2052-4463, (https://doi.org/10.1038/sdata.2016.18) (Mar. 2016).

Appendix A: TITAN publication

Knowledge-Based Systems 232 (2021) 107489

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

TITAN: A knowledge-based platform for Big Data workflow management☆

Antonio Benítez-Hidalgo[1], Cristóbal Barba-González, José García-Nieto, Pedro Gutiérrez-Moncayo, Manuel Paneque, Antonio J. Nebro, María del Mar Roldán-García, José F. Aldana-Montes, Ismael Navas-Delgado *

Dept. de Lenguajes y Ciencias de la Computación, ITIS Software, Universidad de Málaga, Málaga 29071, Spain

ARTICLE INFO

Article history:
Received 30 November 2020
Received in revised form 2 September 2021
Accepted 6 September 2021
Available online 10 September 2021

Keywords:
Big Data analytics
Semantics
Knowledge extraction

ABSTRACT

Modern applications of Big Data are transcending from being scalable solutions of data processing and analysis, to now provide advanced functionalities with the ability to exploit and understand the underpinning knowledge. This change is promoting the development of tools in the intersection of data processing, data analysis, knowledge extraction and management. In this paper, we propose TITAN, a software platform for managing all the life cycle of science workflows from deployment to execution in the context of Big Data applications. This platform is characterised by a design and operation mode driven by semantics at different levels: data sources, problem domain and workflow components. The proposed platform is developed upon an ontological framework of meta-data consistently managing processes and models and taking advantage of domain knowledge. TITAN comprises a well-grounded stack of Big Data technologies including Apache Kafka for inter-component communication, Apache Avro for data serialisation and Apache Spark for data analytics. A series of use cases are conducted for validation, which comprises workflow composition and semantic meta-data management in academic and real-world fields of human activity recognition and land use monitoring from satellite images.

1. Introduction

Big Data is usually described by the well-known V's (Volume, Velocity, and Variety, Variability, Veracity, and Value, among others) and different approaches generally focus on some of these features. A plethora of technologies (most of them Open Source projects, such as those from Apache Foundation) have emerged in the last years around Big Data to propose generic solutions that could underline any of these V's [1,2]. For example, for dealing with Volume, NoSQL databases (Apache Cassandra, MongoDB, Apache HBase, Neo4J among many more[2]) propose new approaches to the efficient management of large datasets. In terms of handling extensive scale data, there exist several proposals to leverage distributed computing in an efficient manner (such as Apache Spark), that ultimately speed up the data processing time. Velocity is not only taken into consideration in NoSQL databases (to ensure the ingestion of fast data streams), but also in the analysis of data streams (e.g., Apache Kafka, Flink, and Spark).

However, one of the main focuses nowadays is on providing Value out of all the available data. Getting this Value is not an easy task that could be solved with a single horizontal technology. Thus, there are also many approaches related to Big Data analytics (i.e., the use of advanced analysis techniques applied to large and diverse datasets) coming from heterogeneous vertical domains, such as transportation [3], healthcare [4], e-Science [5], and agriculture [6]. The goal is to provide solutions that can deal with several Big Data issues in an application using the technologies that best suit for different aspects of the problem.

The development of a solution that provides Value to a given Big Data analysis process is usually hard-coded in the application. With the explosion of data and the expectations to make use of it, organisations start automating data extraction processes, which would end up in data repositories from which extracting Value is a challenge. The use of semantics, including contextual information, is a promising approach to deal with data to improve the data analysis processes and to ensure the efficient reuse

☆ This work has been partially funded by the Spanish Ministry of Science and Innovation via Grant PID2020-112540RB-C41 (AEI/FEDER, UE) and Andalusian PAIDI program with grant P18-RT-2799. Funding for open access charge: Universidad de Málaga / CBUA.

* Corresponding author.
E-mail address: ismael@uma.es (I. Navas-Delgado).

[1] Supported by Grant PRE2018-084280 (Spanish Ministry of Science, Innovation and Universities).

[2] See https://hostingdata.co.uk/nosql-database/ for a long list of classified NoSQL databases.

https://doi.org/10.1016/j.knosys.2021.107489

Appendix B: NORA publication

Received: 9 January 2023 | Accepted: 8 August 2023

DOI: 10.1002/spe.3258

RESEARCH ARTICLE

WILEY

NORA: Scalable OWL reasoner based on NoSQL databases and Apache Spark

Antonio Benítez-Hidalgo | **Ismael Navas-Delgado** | **María del Mar Roldán-García**

KHAOS Research, ITIS Software, Universidad de Málaga, Málaga, Spain

Correspondence
Antonio Benítez-Hidalgo, KHAOS Research, ITIS Software, Universidad de Málaga, Málaga, Spain.
Email: antonio.b@uma.es

Funding information
AETHER-UMA, Grant/Award Number: PID2020-112540RB-C41; Spanish Ministry of Science, Innovation and Universities, Grant/Award Number: PRE2018-084280; Universidad de Málaga / CBUA

Abstract

Reasoning is the process of inferring new knowledge and identifying inconsistencies within ontologies. Traditional techniques often prove inadequate when reasoning over large Knowledge Bases containing millions or billions of facts. This article introduces NORA, a persistent and scalable OWL reasoner built on top of Apache Spark, designed to address the challenges of reasoning over extensive and complex ontologies. NORA exploits the scalability of NoSQL databases to effectively apply inference rules to Big Data ontologies with large ABoxes. To facilitate scalable reasoning, OWL data, including class and property hierarchies and instances, are materialized in the Apache Cassandra database. Spark programs are then evaluated iteratively, uncovering new implicit knowledge from the dataset and leading to enhanced performance and more efficient reasoning over large-scale ontologies. NORA has undergone a thorough evaluation with different benchmarking ontologies of varying sizes to assess the scalability of the developed solution.

KEYWORDS

Apache Spark, knowledge base, NoSQL, OWL, reasoner

1 | INTRODUCTION

The Semantic Web is an extension of the World Wide Web proposed by the World Wide Web Consortium (W3C). The main goal of the Semantic Web is to enable computers to understand the meaning of the data available on the Internet. This vision has been evolving into the concept of the Web of data. This evolution has been done through a technology stack to support this goal. This vision is being materialised through a set of standards for ontologies such as OWL 2 (mainly for TBox, i.e., terminologies) and RDF (mainly for ABox, i.e., assertions). Thus, Semantic Web users have a wide range of possibilities to create data stores on the Web, build vocabularies, and write rules for handling data.

However, as the Web is not a set of isolated Web pages, the goal is to enable ways to connect different data repositories instead of having isolated datasets. In this sense, the Linked Data approach proposes connecting related datasets in the same domain. In line with this concept, FAIR Data principles (Findability, Accessibility, Interoperability, and Reuse) propose a set of requirements to improve how data is published. These recommendations include using standards for

Abbreviations: DL, Description Logic; HDFS, Hadoop Distributed File System; KB, Knowledge Base; NoSQL, Not Only SQL; OWL, Web Ontology Language; RDD, Resilient Distributed Dataset; RDF, Resource Description Framework; SPARQL, SPARQL Protocol and RDF Query Language; SWRL, Semantic Web Rule Language; W3C, World Wide Web Consortium.

Appendix C: SALON publication

BMC Bioinformatics

RESEARCH

Open Access

SALON ontology for the formal description of sequence alignments

Antonio Benítez-Hidalgo[1,2,3*], José F. Aldana-Montes[1,2,3], Ismael Navas-Delgado[1,2,3] and María del Mar Roldán-García[1,2,3]

*Correspondence:
antoniobenitez@lcc.uma.es

[1] Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, Málaga, Spain
[2] University of Málaga, ITIS Software, Ada Byron Research Building, Málaga, Spain
[3] Instituto de Investigación Biomédica de Málaga – IBIMA, Málaga, Spain

Abstract

Background: Information provided by high-throughput sequencing platforms allows the collection of content-rich data about biological sequences and their context. Sequence alignment is a bioinformatics approach to identifying regions of similarity in DNA, RNA, or protein sequences. However, there is no consensus about the specific common terminology and representation for sequence alignments. Thus, automatically linking the wide existing knowledge about the sequences with the alignments is challenging.

Results: The Sequence Alignment Ontology (SALON) defines a helpful vocabulary for representing and semantically annotating pairwise and multiple sequence alignments. SALON is an OWL 2 ontology that supports automated reasoning for alignments validation and retrieving complementary information from public databases under the Open Linked Data approach. This will reduce the effort needed by scientists to interpret the sequence alignment results.

Conclusions: SALON defines a full range of controlled terminology in the domain of sequence alignments. It can be used as a mediated schema to integrate data from different sources and validate acquired knowledge.

Keywords: Sequence alignment, Ontology, Linked data, Data integration, Semantic web

Background

Introduction

Sequence alignment is a well-known bioinformatics approach to identifying regions of similarity in DNA, RNA, or protein sequences. Finding the optimum alignment of two or more biological sequences allows for identifying highly-conserved regions resulting from functional, structural, or evolutionary relationships between the sequences. Thus, these techniques aim to compare nucleic sequences (DNA, RNA) or amino acid sequences (protein) across species or within a genome to establish regions of similarity. Software tools can produce these alignments in different formats, such as those generated by CLUSTAL W [1] or MAFFT [2], making comparative analysis a complex task. Thus, providing a unified way of representing sequence alignments would help