TESIS DOCTORAL

2023

ALINEAMIENTO DE PARADIGMAS DE MODELADO APLICADO A ORIENTACION A OBJETOS Y ONTOLOGIAS

MOHAMED LARHRIB ES-SALEHY

PROGRAMA DE DOCTORADO EN INGENIERÍA DE SISTEMAS Y DE CONTROL

DIRECTORES:

DR. CARLOS CERRADA SOMOLINOS
DR. JUAN JOSE ESCRIBANO RODENAS

Agradecimientos

Un elemento fundamental para experimentar la felicidad plena es cultivar un profundo sentido de agradecimiento. El reconocimiento y aprecio por las circunstancias positivas que se han presentado en nuestras vidas son fundamentales para mantener una perspectiva positiva y constructiva.

En este contexto, me gustaría expresar mi más sincero agradecimiento a todas las personas que han contribuido de manera significativa para que este trabajo de investigación se haya hecho realidad. Sus valiosas contribuciones y apoyo han sido fundamentales en el logro de este proyecto, y estoy profundamente agradecido por ello.

Por tanto, quiero expresar mi gratitud a las personas que han hecho que este trabajo se haga realidad y especialmente a:

Dr. Carlos Cerrada Somolinos: por su gran apoyo.

Dr. Juan José Escribano Ródenas: por su invaluable orientación, gran apoyo, soporte, capacidad motivadora y de dirección.

Dr. Miguel Escribano Ródenas: por su gran colaboración y aporte desde el conocimiento de los sistemas de potencia y el perfil CIM desarrollado por la ENTSOE, CGMES.

Resumen

En la Ingeniería Dirigida por Modelos (MDE), los enfoques más extendidos y adoptados por la comunidad de desarrollo son el enfoque Orientado a Objetos y el enfoque ontológico, los cuales se formalizan utilizando el lenguaje de modelado unificado (UML) y RDF (Resource Description Framework), respectivamente. Sin embargo, existe una falta de alineamiento estructural y comportamental entre estos dos paradigmas. Aunque la transformación entre UML y RDF se puede realizar a nivel estructural, no se ha abordado la transformación de las restricciones definidas en el lenguaje OCL (Object Constraint Language) a SHACL (Shapes Constraint Language), lo cual genera una falta de alineamiento estructural. Además, en el paradigma ontológico no se han definido constructos para la especificación o definición del comportamiento, lo que resulta en una falta de alineamiento comportamental.

Esta tesis se divide en tres partes. En la primera parte, se aborda el alineamiento estructural entre la orientación a objetos y las ontologías. Se realiza la transformación de los diagramas de clase y objetos de UML a su correspondiente formato RDF, y se propone un enfoque para la transformación de OCL a SHACL. En la segunda parte, se aborda el alineamiento comportamental entre la orientación a objetos y las ontologías. Se definen constructos básicos para describir el comportamiento basándose en ontologías, y se establece un mapeo entre los constructos de UML y los propuestos. Se realiza el mapeo de elementos clave de UML, como el diagrama de actividades, la máquina de estados y el diagrama de interacción. La tercera parte se centra en las etapas iniciales del ciclo de vida de la ingeniería de software, como la especificación de requisitos, el análisis y el diseño. Se propone una metodología basada en MDE con el uso de ontologías para abordar estas fases.

El enfoque propuesto se evalúa en el dominio de las compañías eléctricas y sistemas de potencia, utilizando el estándar CGMES (Common Grid Model Exchange Specification). Se realiza la transformación de las reglas OCL definidas en el estándar a SHACL, y se modela el proceso de validación de CGMES utilizando el enfoque ontológico comportamental. Además, se modelan en SHACL las reglas del estándar basadas en texto, consideradas como especificaciones de requisitos.

Palabras Clave: MDE; CGMES; UML; OCL; RDF; SHACL; ODM; Orientado a Objetos; Ontología.

Abstract

In Model-Driven Engineering (MDE), the most widely adopted paradigms in software development are Object-Oriented and Ontological approaches, formalized using the Unified Modeling Language (UML) and Resource Description Framework (RDF), respectively. However, structural, and behavioral alignment is needed between these paradigms. While the transformation between UML and RDF has been explored at the structural level, the transformation of Object Constraint Language (OCL) constraints to Shapes Constraint Language (SHACL) still needs to be addressed, resulting in a lack of structural alignment. Additionally, the ontological paradigm lacks constructs for specifying or defining behavior, further hindering behavioral alignment.

This thesis is divided into three parts. The first part focuses on structural alignment between object-oriented and ontological approaches. It transforms UML class and object diagrams into their corresponding RDF format and proposes an approach for transforming OCL to SHACL. The second part addresses the behavioral alignment between object-oriented and ontological paradigms. Basic constructs for describing behavior based on ontologies are defined, and a mapping between UML constructs and the proposed ones is established. This research framework maps key UML elements such as activity, state machines, and interaction diagrams. The third part concentrates on the initial stages of the software engineering lifecycle, including requirements specification, analysis, and design. A methodology based on MDE using ontologies is proposed to address these phases effectively.

The proposed approach is evaluated in the domain of electric utilities, utilizing the Common Grid Model Exchange Specification (CGMES) standard. The transformation of OCL rules defined in the standard to SHACL is performed, and the behavioral ontological approach is used to model the validation process of CGMES, enabling direct execution. Furthermore, the standard text-based rules are modeled in SHACL, serving as requirements specifications within the proposed approach.

Keywords: MDE; CGMES; UML; OCL; RDF; SHACL; ODM; Object-Oriented; Ontology.

Tabla de contenido

1. **Motivación**

La necesidad inicial consistía en abordar el desafío de validar los modelos CGMES *(Common Grid Model Exchange Standard)* que se encuentran especificados en RDF *(Resource Description Framework)* contra un conjunto de reglas basadas en texto o definidas en OCL *(Object Constraint Language*), ambas pertenecientes al mismo estándar. Aunque estos modelos están conceptualmente orientados a objetos, se serializan en RDF para lograr la interoperabilidad requerida. Sin embargo, en el momento de validarlos, se realiza una transformación de vuelta a UML *(Unified Modeling Language)*, es decir, a la orientación a objetos, con el fin de llevar a cabo el proceso de validación.

A partir de este análisis, se evidencia la existencia de dos paradigmas complementarios en el ámbito de este estudio: el paradigma orientado a objetos (UML) y el paradigma de las ontologías (RDF). En consecuencia, se plantea la resolución del problema de validación de los grafos RDF mediante la utilización de un lenguaje de validación estándar. En el año 2017, el consorcio W3C introduce el lenguaje de validación estándar denominado SHACL *(Shapes Constraint Language)*.

Sin embargo, el estándar CGMES también incluye un conjunto de reglas definidas en OCL *(Object Constraint Language)*, lo cual ha generado la necesidad de convertir estas reglas a formato SHACL. Este proceso de transformación ha dado lugar a un alineamiento estructural entre los paradigmas UML+OCL y RDF+SHACL, permitiendo una integración coherente y efectiva de ambos enfoques.

Posteriormente, se ha observado que en el ámbito de la ingeniería de software basada en ontologías no se dispone de un lenguaje visual equivalente al que ofrece UML para describir el comportamiento de los sistemas. Para abordar esta limitación, se ha desarrollado un lenguaje visual específico para la especificación ontológica del comportamiento. Es importante destacar que la validación se considera un caso particular del razonamiento.

Esta necesidad ha impulsado la investigación y el enfoque hacia la ingeniería de software basada en ontologías, buscando alinearla con el paradigma predominante utilizado en la comunidad de ingeniería de software, es decir, la Orientación a Objetos con UML. De este modo, se busca unificar y armonizar estos enfoques, aprovechando la fortaleza de la orientación a objetos y la capacidad expresiva de las ontologías en el proceso de ingeniería de software.

2. **Introducción**

En esta tesis se aborda el alineamiento de los paradigmas de modelado ontológico y orientado a objetos, desde la perspectiva de la ingeniería dirigida por modelos MDE *(Model-Drive Engineering)*. El alineamiento tiene lugar en las tres partes principales de la ingeniera de software; la fase de la creación estructural, el área comportamental y la fase de especificación de requisitos.

La primera parte de la tesis se centrará en el alineamiento de la orientación a objetos y las ontologías en términos estructurales. Esto se debe a que UML representa las estructuras mediante diagramas de clases y diagramas de objetos, mientras que las restricciones se definen con el lenguaje OCL. Será necesario transformar los diagramas de clase y objetos a su correspondiente formato RDFS *(RDF Schema)* y RDF de datos. Sin embargo, el principal esfuerzo se ha enfocado en la transformación de OCL a SHACL, aspecto que será abordado en dicha parte de la tesis.

En la segunda parte de la tesis se va a abordar el alineamiento de la orientación a objetos con las ontologías en términos de comportamiento. Primero se van a definir un conjunto de constructos básicos para definir el comportamiento en base a las ontologías y con ello hacer un mapeo entre los constructos que ofrece UML y los constructos propuestos. En el marco de esta investigación, se llevará a cabo el mapeo de tres elementos clave de UML: el diagrama de actividades, la máquina de estados y el diagrama de interacción, porque se considera que estos tres elementos son básicos para la especificación del comportamiento en este paradigma. Este proceso de mapeo consistirá en establecer una correspondencia entre los constructos y conceptos utilizados en dichos diagramas en UML, y los equivalentes en RDF y OWL. Este enfoque permitirá una integración más sólida y coherente entre los modelos UML y los lenguajes semánticos RDF y OWL, contribuyendo así a mejorar la interoperabilidad y la representación formal de los sistemas de información.

La tercera parte de la tesis se enfocará específicamente en las etapas iniciales del ciclo de vida de la ingeniería de software, que incluyen la especificación de requisitos, el análisis y el diseño. Se propone una metodología basada en Modelado Dirigido por Ontologías (MDE) para destacar que todas estas fases son, en realidad, tareas de modelado que conducen a la obtención del producto final. Esta metodología proporcionará un enfoque sistemático y estructurado para abordar estas etapas críticas, permitiendo una mejor especificación y representación de los requisitos del sistema, así como una mayor coherencia y trazabilidad a lo largo de todo el proceso de desarrollo del software.

3. **Paradigmas de modelado**

Sería apropiado examinar la definición del término "paradigma" desde diversas fuentes con el fin de obtener una perspectiva más completa y precisa.

Según la RAE se define como: “Teoría o conjunto de teorías cuyo núcleo central se acepta sin cuestionar y que suministra la base y modelo para resolver problemas y avanzar en el conocimiento. El paradigma newtoniano.”

En el campo de la filosofía, el término "paradigma" se utiliza para describir un conjunto de supuestos, creencias y valores fundamentales que establecen un marco de referencia para comprender la realidad y orientar la investigación y el análisis de los fenómenos. En filosofía, un paradigma implica una forma particular de concebir el mundo, que está gobernada por un conjunto de reglas y convenciones que dictan cómo se debe llevar a cabo la investigación y la interpretación de los datos. Además,

implica una serie de preguntas y problemas considerados importantes y relevantes dentro de ese campo de estudio.

En resumen, dentro del ámbito filosófico, un paradigma se entiende como una perspectiva o marco conceptual que actúa como una lente a través de la cual se observa e interpreta el mundo. Esta perspectiva tiene una influencia significativa en la selección de las preguntas a plantear, los enfoques a adoptar y los métodos de investigación a utilizar. Un paradigma filosófico puede abarcar desde supuestos básicos sobre la naturaleza de la realidad y la existencia, hasta formas más específicas de investigación y análisis utilizadas en una disciplina particular.

Desde una perspectiva epistemológica, el concepto de paradigma se puede entender como una estructura, patrón o modelo que nos permite adquirir conocimiento y comprender el mundo.

En el ámbito de la ingeniería, un paradigma se define como un marco conceptual y metodológico que proporciona al ingeniero una base para abordar las necesidades y desarrollar soluciones con una utilidad específica. En este sentido, un paradigma se puede entender como un conjunto de teorías y técnicas que resultan útiles para comprender los sistemas y procesos, así como para el diseño, construcción y mantenimiento de productos y servicios.

Un paradigma en ingeniería ofrece una visión y enfoque coherentes para resolver problemas y desafíos técnicos, y guía al ingeniero en la selección y aplicación de herramientas, metodologías y prácticas adecuadas. Representa un conjunto de principios, modelos y estándares que han demostrado ser efectivos y eficientes en la disciplina de la ingeniería, facilitando la toma de decisiones informadas y fomentando el desarrollo de soluciones confiables y de alta calidad.

En el ámbito de la ingeniería, los paradigmas engloban una diversidad de enfoques metodológicos que se emplean para abordar la resolución de problemas. Estos enfoques incluyen, entre otros, el enfoque de diseño, el enfoque de sistemas y el enfoque de ingeniería de software. Cada uno de estos enfoques presenta un conjunto particular de principios, técnicas y herramientas que se aplican en diferentes etapas del proceso de ingeniería.

Los paradigmas en ingeniería están influenciados por las teorías y conceptos fundamentales derivados de diversas disciplinas científicas, como la física, la matemática, la electrónica y otras áreas relacionadas. Estos conocimientos científicos proporcionan la base teórica y conceptual para desarrollar soluciones ingenieriles sólidas y eficientes. Además, los paradigmas en ingeniería se nutren de los avances tecnológicos y las investigaciones en el campo científico, lo que permite incorporar nuevas técnicas y enfoques a medida que evoluciona el conocimiento científico y las demandas de la sociedad.

Un paradigma en ingeniería puede ser conceptualizado como una estructura establecida para la resolución de un problema técnico particular, que ofrece un marco

de referencia para la solución de problemas futuros en el campo. Los ingenieros pueden seleccionar diversos paradigmas según las características específicas y la complejidad del problema que están abordando, y pueden modificar o adaptar estos paradigmas a medida que evolucionan las tecnologías y cambian las demandas y necesidades de la sociedad.

Un paradigma de modelado puede definirse como un conjunto sistemático y coherente de conceptos, enfoques y técnicas utilizados en la construcción y representación de modelos de sistemas y fenómenos en diversas disciplinas, incluyendo la ingeniería, la física, la biología, entre otras. Dichos paradigmas determinan los elementos fundamentales que deben ser considerados en el proceso de modelado, así como las relaciones y reglas que rigen su construcción y validación, junto con los métodos y herramientas requeridos para estos fines. Además, los paradigmas de modelado prescriben la manera en que el modelo se puede utilizar para analizar y simular el comportamiento del sistema o fenómeno en cuestión. Los paradigmas de modelado son dinámicos y evolutivos, influenciados tanto por los avances en la tecnología, la ciencia y las matemáticas, como por las demandas y necesidades cambiantes de la sociedad.

En el campo de la ingeniería, un paradigma de modelado se define como un conjunto de conceptos, enfoques y técnicas que se emplean para construir y representar modelos de sistemas y fenómenos en diferentes disciplinas de la ingeniería, como la ingeniería mecánica, eléctrica, de software, entre otras. Estos paradigmas de modelado abarcan tanto la definición de los elementos fundamentales del modelo, así como las relaciones entre ellos, los métodos y herramientas utilizados para la creación y validación del modelo, y la forma en que el modelo se aplica para analizar y simular el comportamiento del sistema o fenómeno en cuestión. Los paradigmas de modelado en ingeniería evolucionan y se desarrollan en función del avance de la tecnología y de los nuevos desafíos que surgen en la resolución de problemas en diferentes campos de la ingeniería.

En el ámbito de la ingeniería, un modelo se refiere a una representación simplificada y abstracta de un sistema o fenómeno real que se utiliza para el análisis, la comprensión, el diseño o la simulación del sistema o fenómeno. Estos modelos pueden ser representaciones matemáticas, físicas, conceptuales o computacionales del sistema o fenómeno en cuestión.

En ingeniería, los modelos pueden ser utilizados para predecir el comportamiento de un sistema o fenómeno en diferentes condiciones y situaciones, así como para probar diferentes diseños y soluciones antes de su implementación en la realidad. Los modelos son herramientas fundamentales para la investigación, el diseño y el desarrollo de sistemas y productos en distintas disciplinas de la ingeniería, como la ingeniería mecánica, la ingeniería eléctrica, la ingeniería civil, la ingeniería de software, entre otras.

La Ingeniería Dirigida por Modelos (MDE) es una metodología de desarrollo de software que se apoya en la creación de modelos para representar el comportamiento, estructura y funcionalidad del sistema que se desea desarrollar. En MDE, los modelos se emplean como la principal fuente de información para el diseño, la implementación y la validación del sistema, lo que permite un mayor nivel de automatización y eficiencia en el desarrollo de software.

MDE utiliza lenguajes de modelado específicos, como UML (Lenguaje de Modelado Unificado) y SysML *(Systems Modeling Language)* entre otros, así como herramientas de modelado para crear los modelos. Estos modelos son empleados para generar automáticamente el código fuente del sistema, lo que reduce el tiempo y los errores asociados a la escritura manual del código. Además, los modelos también se usan para verificar la corrección y coherencia del sistema, y para realizar pruebas y simulaciones del sistema antes de su implementación.

MDE se aplica en diversas áreas de la ingeniería de software, tales como la ingeniería de sistemas, la ingeniería de software empotrado, la ingeniería de software de tiempo real y la ingeniería de software basada en servicios.

En la actualidad la práctica de MDE se lleva a cabo usando el paradigma de modelado de la orientación a objetos con UML y lenguajes de propósito general orientados objetos como Java, C#, C++ entre otros.

La granularidad del modelado se refiere al nivel de detalle que se emplea para representar un sistema o fenómeno en un modelo. Se trata de la cantidad de información incluida en el modelo y la escala en la que se representan los componentes del sistema o fenómeno. En términos generales, un modelo con alta granularidad incluirá un mayor número de detalles y componentes específicos, mientras que un modelo con baja granularidad tendrá una representación más general y simplificada del sistema o fenómeno.

La selección de la granularidad adecuada para un modelo depende de varios factores, como el propósito del modelo, la complejidad del sistema o fenómeno que se va a modelar y las herramientas y técnicas de modelado disponibles. En la ingeniería, se busca utilizar la granularidad óptima que permita entender y diseñar el sistema o fenómeno de manera eficiente y efectiva.

En los contextos relevantes de este trabajo, se destacan dos enfoques de modelado particularmente relevantes: el modelado orientado a objetos y el modelado ontológico. Este último se caracteriza por una mayor granularidad en comparación con el primero. En el modelado orientado a objetos, se representan los sistemas y fenómenos mediante objetos y sus relaciones, utilizando conceptos como clases, atributos y métodos. Por otro lado, el modelado ontológico se enfoca en capturar la semántica y la estructura conceptual subyacente de un dominio, mediante la creación de ontologías que describen las entidades, sus propiedades y las relaciones entre ellas de manera más detallada y precisa. La granularidad fina del modelado ontológico permite una representación más rica y profunda de los conocimientos y las relaciones

en un dominio específico, lo que resulta especialmente útil en contextos donde se requiere una mayor precisión y claridad en la representación del conocimiento.

Las diferencias entre los paradigmas de modelado se encuentran principalmente en la cantidad de conceptos utilizados en la creación del modelo, en el tipo de lenguaje empleado para el modelado (ya sea gráfico o textual) y en la capacidad de razonamiento que puede estar o no implícita en el paradigma.

En cuanto a la cantidad de conceptos, algunos paradigmas de modelado pueden ser más extensos y complejos, involucrando una amplia gama de elementos conceptuales para representar y describir el sistema o fenómeno en cuestión. Otros paradigmas pueden ser más concisos, empleando un conjunto más reducido de conceptos esenciales.

En relación con el lenguaje de modelado, algunos paradigmas utilizan representaciones gráficas, como diagramas, para visualizar y comunicar las estructuras y relaciones del modelo. Estos diagramas proporcionan una representación visual intuitiva que facilita la comprensión del sistema modelado. Por otro lado, otros paradigmas emplean lenguajes de modelado basados en texto, donde se utilizan expresiones y reglas escritas para definir y describir el modelo.

Además, la capacidad de razonamiento puede variar entre los paradigmas de modelado. Algunos paradigmas incorporan mecanismos de razonamiento y deducción de manera intrínseca, lo que permite realizar análisis y deducciones lógicas sobre el modelo. Estos paradigmas suelen contar con una base formal y sólida que facilita el razonamiento automatizado. En cambio, otros paradigmas pueden no tener una capacidad de razonamiento explícita y requerir métodos y técnicas adicionales para llevar a cabo análisis y deducciones.

4. Ontologías vs Orientación a Objetos como paradigmas

4.1. El paradigma ontológico

En este trabajo se ha dado una definición al paradigma ontológico para hacerlo de utilidad para la ingeniería de software. Con este paradigma el mundo es entendido como un conjunto de conceptos y relaciones sobre el cual se lleva a cabo la tarea de razonamiento. Ver Figura 4.1.

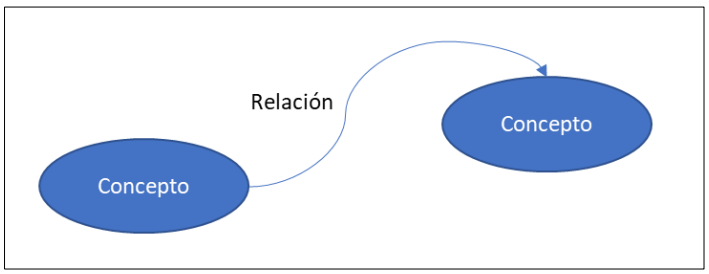


Figura 4.1: El esquema básico del paradigma ontológico.

4.2. **El paradigma Orientado a Objetos**

La definición adoptada en la actualidad es la que percibe el mundo como constituido de objetos que tienen características y desempeñan funciones. Ver Figura 4.2.

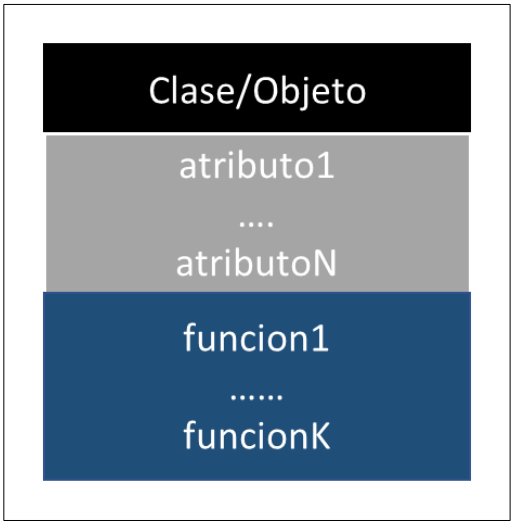


Figura 4.2: Esquema de la orientación a objetos

5. **RDF/OWL/SHACL VS UML**

El propósito de esta sección consiste en examinar los dos principales lenguajes utilizados en los paradigmas ontológico y orientado a objetos en la actualidad. El objetivo es realizar el mapeo de los conceptos de ambos lenguajes y enriquecer el paradigma ontológico con aquellos conceptos que actualmente les son ajenos.

La Arquitectura Orientada a Modelos (Model Driven Architecture o MDA) es un enfoque respaldado por la OMG (Object Management Group) para el diseño, desarrollo e implementación de software. MDA proporciona directrices para estructurar las especificaciones de software que se expresan mediante modelos [1].

OMG proporciona un conjunto de especificaciones formales y estandarizadas relacionadas con el modelado y los metadatos en el ámbito de la ingeniería de software. Estas especificaciones incluyen UML, MOF *(Meta-Object Facility)*, CWM *(Common Warehouse Metamodel)*, XMI *(XML Metadata Interchange)* y Perfil UML.

W3C *(World Wide Web Consortium)* especifica un conjunto de lenguajes estandarizados para el modelado estructural. El lenguaje principal es RDF, que se fundamenta en los conceptos fundamentales de recurso y relación. El constructo básico de modelado en RDF es la tripleta, que sigue la estructura (sujeto, predicado, objeto).

5.1. **RDF**

RDF es una especificación del W3C para representar la información en la web; donde el modelado es especificado mediante una tripleta de la forma (sujeto predicado objeto). El conjunto de tripletas que representan el modelo es un grafo

dirigido como se ilustra en la Figura 5.1. Este enfoque gráfico facilita la comprensión y el análisis del modelo RDF, permitiendo visualizar la interconexión de los elementos y la forma en que se relacionan entre sí.

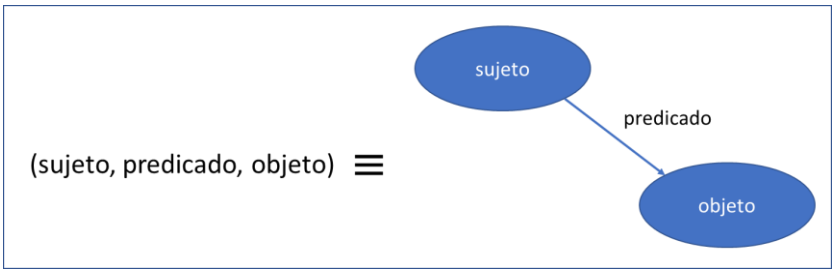


Figura 5.1: Representación de la tripleta en RDF.

5.2. **RDF Schema**

El RDF Schema proporciona un conjunto de términos y reglas semánticas que permiten crear metamodelos para describir la estructura y las relaciones de los modelos RDF.

Se presenta un resumen del vocabulario del RDF Schema para las clases en la Tabla 5.1, y para las propiedades en la Tabla 5.2. El RDF Schema de la parte normativa, que define el propio RDF Schema, se muestra en el Listado 5.1.

Nombre de la clase	Descripción
rdfs:Resource	Cualquier cosa, la clase resource.
rdfs:Literal	La clase de valores literales por ejemplo string y integers.
rdf:langString	La clase de valores literales de cadena etiquetados por idioma.
rdf:HTML	La clase de valores literales HTML.
rdf:XMLLiteral	La clase de valores literales XML.
rdfs:Class	La clase de las clases.
rdf:Property	La clase de propiedades RDF.
rdfs:Datatype	La clase de tipos de datos RDF.
rdf:Statement	La clase de sentencias RDF.
rdf:Bag	La clase de contenedores desordenados.
rdf:Seq	La clase de contenedores ordenados.
rdf:Alt	La clase de contenedores de alternativas.
rdfs:Container	La clase de contenedores RDF.
rdfs:ContainerMembershipProperty	La clase de propiedades de membresía del contenedor, rdf:_1, rdf:_2, ..., todas las cuales son subpropiedades de 'member'.
rdf:List	La clase de Listas RDF.

Tabla 5.1: Resumen vocabulario RDFS para las clases.

Property	Comment	Domain	Range
rdf:type	The subject is an instance of a class.	rdfs:Resource	rdfs:Class
rdfs:subClassOf	The subject is a subclass of a class.	rdfs:Class	rdfs:Class
rdfs:subPropertyOf	The subject is a subproperty of a property.	rdf:Property	rdf:Property
rdfs:domain	A domain of the subject property.	rdf:Property	rdfs:Class
rdfs:range	A range of the subject property.	rdf:Property	rdfs:Class
rdfs:label	A human-readable name for the subject.	rdfs:Resource	rdfs:Literal
rdfs:comment	A description of the subject resource.	rdfs:Resource	rdfs:Literal
rdfs:member	A member of the subject resource.	rdfs:Resource	rdfs:Resource
rdf:first	The first item in the subject RDF list.	rdf:List	rdfs:Resource
rdf:rest	The rest of the subject RDF list after the first item.	rdf:List	rdf:List
rdfs:seeAlso	Further information about the subject resource.	rdfs:Resource	rdfs:Resource
rdfs:isDefinedBy	The definition of the subject resource.	rdfs:Resource	rdfs:Resource
rdf:value	Idiomatic property used for structured values.	rdfs:Resource	rdfs:Resource
rdf:subject	The subject of the subject RDF statement.	rdf:Statement	rdfs:Resource
rdf:predicate	The predicate of the subject RDF statement.	rdf:Statement	rdfs:Resource
rdf:object	The object of the subject RDF statement.	rdf:Statement	rdfs:Resource

Tabla 5.2: Resumen vocabulario RDFS para las propiedades.

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

# Clases
rdfs:Class rdfs:subClassOf rdfs:Resource;
           rdf:type rdfs:Class.
rdfs:Resource rdf:type rdfs:Class.
rdfs:Literal rdf:type rdfs:Class;
             rdfs:subClassOf rdfs:Resource.
rdfs:Datatype rdf:type rdfs:Class;
              rdfs:subClassOf rdfs:Class.
rdf:langString rdf:type rdfs:Datatype;
               rdfs:subClassOf rdfs:Literal.
rdf:Property rdf:type rdfs:Class.

# Propiedades
```

```
rdfs:range rdf:type rdf:Property;
                rdfs:domain rdf:Property;
          rdfs:range rdfs:Class.
rdfs:domain rdf:type rdf:Property;
             rdfs:domain rdf:Property;
             rdfs:range rdfs:Class.
rdf:type rdf:Property;
             rdfs:domain rdfs:Resource;
             rdfs:range rdfs:Class.
rdfs:subClassOf rdf:type rdf:Property;
```

Listado 5.1: RDF Schema de RDF Schema.

5.3. **SPARQL**

SPARQL 1.1 *(SPARQL Protocol and RDF Query Language)* es un conjunto de especificaciones desarrolladas por el Consorcio W3C que proporcionan un conjunto de lenguajes y protocolos para realizar consultas y manipular contenido de grafos RDF en la web o en un almacén de RDF. Estas especificaciones están diseñadas para permitir a los usuarios realizar operaciones avanzadas sobre datos RDF, como consultas complejas, agregación, subconsultas y actualizaciones de datos. SPARQL 1.1 consta de un conjunto de estándares interrelacionados que definen la sintaxis, la semántica y los protocolos necesarios para la implementación y el uso de SPARQL en entornos distribuidos y heterogéneos. Estas especificaciones incluyen "SPARQL 1.1 Query Language", "SPARQL 1.1 Query Results JSON Format", "SPARQL 1.1 Federated Query", "SPARQL 1.1 Entailment Regimes", "SPARQL 1.1 Update Language", "SPARQL 1.1 Protocol for RDF", "SPARQL 1.1 Service Description" y "SPARQL 1.1 Graph Store HTTP Protocol". [2] proporciona una descripción general de SPARQL 1.1

En la sesión 6.3.11 se abordará una limitación de SPARQL en relación con el formato actual de los resultados de la consulta, que se limita únicamente a una estructura tabular. Como propuesta de solución, se planteará la extensión del lenguaje para que también admita la devolución de un grafo como resultado de la consulta.

5.3.1 **SPARQL 1.1 Query Language**

SPARQL 1.1 Query Language, especificado en [3], es un lenguaje de consulta diseñado para recuperar y manipular información almacenada en grafos RDF. Es una parte integral de SPARQL, el cual es un estándar del W3C para consultar y manipular datos RDF. Proporciona una sintaxis y semántica formal para expresar consultas que permiten acceder a los datos RDF de manera flexible y eficiente. Con este lenguaje, es posible realizar consultas complejas y precisas sobre los grafos RDF, lo que permite buscar patrones, filtrar resultados y combinar múltiples criterios de búsqueda.

5.3.2 **SPARQL 1.1 Update**

SPARQL 1.1 Update es una extensión del lenguaje SPARQL que permite realizar modificaciones en los grafos RDF, además de las operaciones de consulta proporcionadas por SPARQL 1.1 Query Language. Esta extensión fue introducida por el consorcio W3C como una parte del estándar SPARQL 1.1. Se proporcionan un conjunto

de comandos y operaciones que permiten insertar, modificar y eliminar datos en los grafos RDF. Estas operaciones de actualización se realizan mediante consultas SPARQL especiales que se envían a un punto de acceso SPARQL que soporte la funcionalidad de actualización. Esta especificación se encuentra en [4].

5.3.3 **SPARQL1.1 Service Description**

En [5] se proporciona una descripción de servicio SPARQL donde se enumeran las características de un servicio SPARQL disponible a través del Protocolo SPARQL 1.1 para RDF. La especificación de la descripción de servicio SPARQL proporciona un método para descubrir el servicio a partir de un servicio SPARQL específico, y un esquema RDF para codificar dicha descripción.

El Listado 5.2 contiene el RDFS de la descripción del servicio SPARQL. Este RDF es el metamodelo de la descripción, esto sirve de ejemplo para ilustrar que el modelado está presente en todas las áreas de la ingeniería del software cuando se adopta MDE.

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

# Clases

sd:Service rdf:type rdfs:Class .
sd:Feature rdf:type rdfs:Class .
sd:Language rdf:type rdfs:Class ;
            rdfs:subClassOf sd:Feture .
sd:Function rdf:type rdfs:Class ;
            rdfs:subClassOf sd:Feature .
sd:Aggregate rdf:type rdfs:Class ;
             rdfs:subClassOf sd:Feature .
sd:EntailmentRegime rdf:type rdfs:Class ;
                    rdfs:subClassOf sd:Feature .
sd:EntailmentProfile rdf:type rdfs:Class ;
                     rdfs:subClassOf sd:Feature .
sd:GraphCollection rdf:type rdfs:Class .
sd:DataSet rdf:type rdfs:Class ;
           rdfs:subClassOf sd:GraphCollection .
sd:Graph rdf:type rdfs:Class .
sd:NameGraph rdf:type rdfs:Class .

# Propiedades.

sd:endpoint rdf:type owl:InverseFunctionalProperty ;
            rdfs:domain sd:Service.
```

```
sd:feature   rdf:type rdf:Property ;
             rdfs:domain sd:service ;
             rdfs:range sd:Feature.

sd:defaultEntailmentRegime rdfs:subPropertyOf sd:feature ;
                           rdfs:domain:sd:Service ;
                           rdfs:range:   sd:EntailmentRegime .

sd:entailmentRegime rdf:type rdf:Property ;
                    rdfs:domain sd:NamedGraph ;
                    rdfs:range:  sd:EntailmentRegime .

sd:defaultSupportedEntailmentProfile rdfs:subPropertyOf sd:feature ;
                                     rdfs:domain sd:Service ;
                                     rdfs:range sd:EntailmentProfile .

sd:supportedEntailmentProfile rdf:type rdf:Property ;
                              rdfs:domain sd:NamedGraph ;
                              rdfs:range sd:EntailmentProfile .

sd:extensionFunction rdfs:subPropertyOf sd:feature ;
                     rdfs:domain sd:Service ;
                     rdfs:range sd:Function .

sd:languageExtension rdfs:subPropertyOf sd:feature ;
                     rdfs:domain sd:Service ;
                     rdfs:range sd:feature .

sd:supportedLanguage rdfs:subPropertyOf sd:feature ;
                     rdfs:domain sd:Service ;
                     rdfs:range sd:Language .

sd:propertyFeature   rdfs:subPropertyOf sd:feature ;
                     rdfs:domain sd:Service ;
                     rdfs:range sd:Feature .

sd:defaultDataset    rdf:type owl:InverseFunctionalProperty ;
                     rdfs:domain sd:Service ;
                     rdfs:range sd:DataSet .

sd:availableGraphs    rdf:type rdf:Property ;
                      rdfs:domain sd:Service ;
                      rdfs:range sd:GraphCollection .

sd:resultFormat rdf:type sd:Service ;
                rdfs:domain sd:Service ;
                rdfs:range <http://www.w3.org/ns/formats/Format> .
```

```
sd:defaultGraph    rdf:type rdf:Property ;
                   rdfs:domain sd:DataSet ;
                   rdfs:range sd:Graph .

sd:namedGraph    rdf:type rdf:Property ;
                 rdfs:domain sd:GraphCollection ;
                 rdfs:range sd:NamedGraph .

sd:name rdf:type rdf:Property ;
        rdfs:domain sd:NamedGaraph.

sd:graph rdf:type rdf:Property ;
         rdfs:domain sd:NamedGraph ;
         rdfs:range sd:Graph .

# Instancias

# Instancias de la clase sd:Feature.

sd:Feature rdf:type sd:Feature .
sd:DereferencesURIs rdf:type sd:Feature .
sd:UnionDefaultGraph rdf:type sd:Feature .
sd:RequiresDataset rdf:type sd:Feature .
sd:EmptyGraphs rdf:type sd:Feature .
sd:BasicFederatedQuery rdf:type sd:Feature .

# Instancias de la clase sd:Language.

sd:SPARQL10Query rdf:type sd:Language .
sd:SPARQL11Query rdf:type sd:Language .
sd:SPARQL11Update rdf:type sd:Language .
sd:SPARQL12Query rdf:type sd:Language .
sd:SPARQL12Update rdf:type sd:Language .
sd:DereferencesURIs rdf:type sd:Language .
sd:UnionDefaultGraph rdf:type sd:Language .
sd:RequiresDataset rdf:type sd:Language .
sd:EmptyGraphs rdf:type sd:Language .
sd:BasicFederatedQuery rdf:type sd:Language .
```

Listado 5.2: RDFS de la descripción de servicio SPARQL 1.1.

5.3.4 **SPARQL 1.1 Federated Query**

La especificación, referenciada en [6], establece la sintaxis y semántica de la extensión de consulta federada de SPARQL 1.1, la cual posibilita la ejecución de consultas distribuidas en diversos "*endpoints*" de SPARQL. La inclusión de la palabra clave *SERVICE* en SPARQL 1.1 amplía su alcance al admitir consultas que fusionan datos distribuidos en la Web.

5.3.5 **SPARQL 1.1 Query Results JSON Format**

En [7] se detalla la forma de serializar los resultados de consultas SPARQL (en las variantes SELECT y ASK) en formato JSON. Dicho formato ha sido diseñado para proporcionar una representación exhaustiva de la información contenida en los resultados de la consulta. En el caso de una consulta SELECT, los resultados se serializan como un conjunto de elementos en un arreglo, donde cada elemento corresponde a una "fila" de los resultados de la consulta. Por su parte, una consulta ASK devuelve un valor booleano que representa el resultado de la consulta.

Además, se establece un tipo de medio de Internet específico para el formato JSON de los resultados de SPARQL, denominado application/sparql-results+json.

El esquema JSON se muestra en Listado 5.3

```
{
  "head": {"vars": ["var1", "var2", ... ,"varn"]},
  "results": {
              "bindings":[
                          {
                               "var1": {"type": "tipo1", "value": "valor1"}
                               .....
                               "varn": {"type": "tipon", "value": "valorn"}
                          }
                           ........
              ]
      }
  }
```

Listado 5.3: Esquema JSON para resultado de una consulta SPARQL.

El esquema del resultado de una consulta ASK se muestra en el Listado 5.4.

```
{
  "head" : { } ,
  "boolean" : true|false
}
```

Listado 5.4: Esquema JSON para resultado de una consulta ASK.

5.3.6 **SPARQL 1.1 Query Results CSV and TSV Formats**

El documento referenciado en [8] aborda la descripción de los formatos CSV (valores separados por comas) y TSV (valores separados por tabulaciones) empleados para la representación de los resultados de consultas SELECT en SPARQL. Dichos formatos desempeñan un papel fundamental al servir como un estándar compartido entre sistemas que hacen uso de diferentes tecnologías de implementación

5.3.7 **SPARQL Query Results XML Format (Second Edition)**

La especificación en [9] describe la codificación de los resultados de una consulta SPARQL, ya sea de tipo SELECT o ASK, en formato XML.

La plantilla XML correspondiente a los resultados de una consulta SPARQL se muestra en el Listado 5.5.

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
 <head>
  <variable name="name1."/>
  .....
  <variable name="namek"/>
 </head>
 <results>
  <result>
   <binding name="name1"> value11 </binding>
   ....
   <binding name="namek"> value1k </binding>
  </result>
  ....
  <result>
   <binding name="name1"> valueN1 </binding>
   ....
   <binding name="namek"> valueNk </binding>
  </result>
 </results>
</sparql>
```

Listado 5.5: Formato XML del resultado de una consulta SPARQL

5.3.8 **SPARQL 1.1 Entailment Regimes**

El procesamiento de los resultados de una consulta en SPARQL está basado en el encaje de subgrafo que representa la consulta contra el grafo de datos. Los dos grafos son grafos dirigidos construidos mediante tripletas. El encaje se lleva a cabo mediante variables que actúan como comodines. Los estándares RDF y OWL ofrecen interpretaciones semánticas para inferir nuevas aserciones a partir de las implícitamente declaradas a través del grafo. En [10] se encuentra la especificación.

5.3.9 **SPARQL 1.1 Protocol**

El protocolo SPARQL, especificado en [11], establece las especificaciones para las solicitudes de los clientes y las respuestas de los servicios a través del protocolo HTTP. Este protocolo se compone de dos operaciones principales: una para realizar consultas SPARQL y otra para realizar actualizaciones SPARQL. En el contexto de este protocolo, se destacan los siguientes conceptos:

- El protocolo SPARQL cliente, que es esencialmente un cliente HTTP utilizado para llevar a cabo las operaciones del protocolo SPARQL.
- El servicio de protocolo SPARQL, que actúa como un servidor HTTP para atender las solicitudes y proporcionar respuestas HTTP en relación con las operaciones del protocolo SPARQL.
- El SPARQL *endpoint*, que se refiere a la URI *(Universa Resource Identifier)* utilizada por un servicio de protocolo SPARQL para recibir solicitudes de clientes que utilizan el Protocolo SPARQL.
- La operación del protocolo SPARQL, que implica una solicitud y una respuesta HTTP que se ajustan a las especificaciones del protocolo.
- El RDF *Dataset*, que se compone de un grafo predeterminado y cero o más grafos con nombre. Este conjunto de grafos se utiliza para representar la información en formato RDF y se utiliza en las operaciones del protocolo SPARQL.

5.3.10 **SPARQL 1.1 Graph Store HTTP Protocol**

Se trata de una especificación no normativa que se presenta como una opción al Protocolo de Actualización SPARQL 1.1. Esta especificación describe el uso de las operaciones HTTP para la gestión de colecciones de grafos RDF. Para llevar a cabo consultas, modificaciones y administración de los grafos y su contenido en un *dataset* dentro de un *endpoint*, se utilizan los verbos o métodos HTTP GET, PUT, POST y DELETE. Estos métodos permiten implementar las operaciones mencionadas, brindando así un conjunto de herramientas para interactuar con los datos de manera efectiva y eficiente. En [12] se encuentra la especificación de este protocolo.

5.3.11 **Propuesta de extensión de SPARQL**

Una limitación de la especificación actual de SPARQL consiste en que los resultados de las consultas se devuelven únicamente en formato tabular. Sería útil poder realizar consultas que devuelvan un grafo como resultado, lo cual permitiría utilizarlo posteriormente como un recurso en la web semántica. Por ejemplo, se podría alojar como un nuevo grafo y realizar razonamiento y consultas sobre él.

Un ejemplo de una consulta básica y su resultado en la especificación actual de SPARQL se muestra en la Figura 5.2.

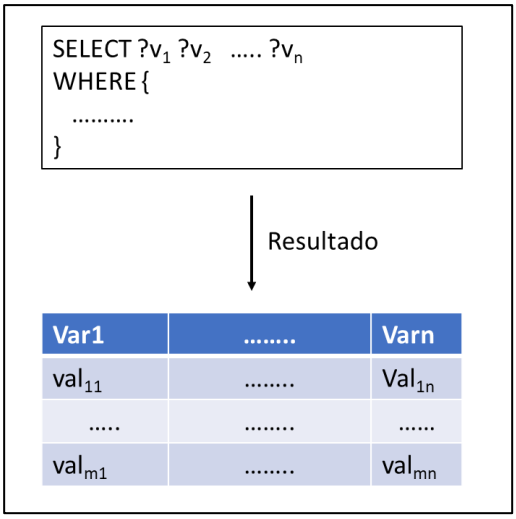


Figura 5.2: Consulta básica y sus resultados en SPARQL.

Se presenta una propuesta de extensión del lenguaje SPARQL para permitir la realización de consultas con la finalidad de obtener un grafo como resultado. Esta propuesta se ilustra en la Figura 5.3.

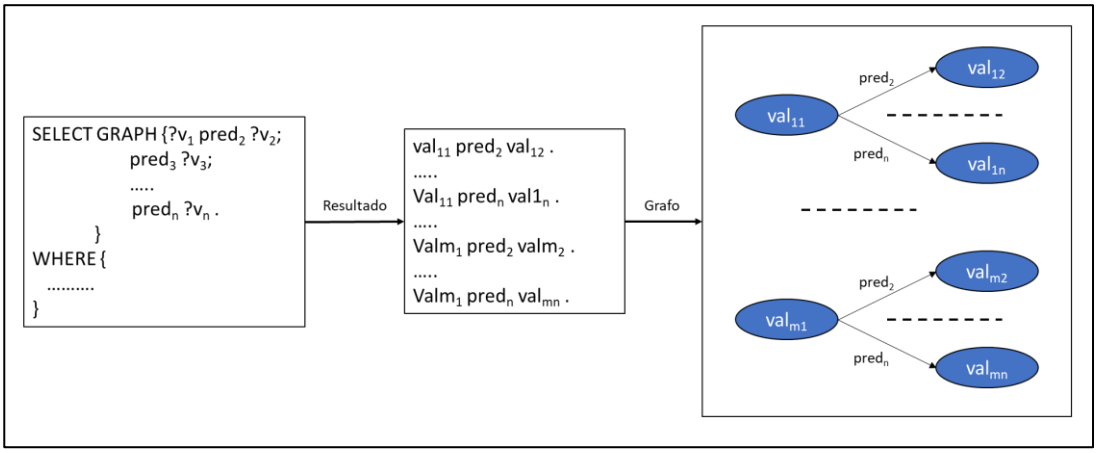


Figura 5.3: Extensión de SPARQL para permitir grafos como resultado de consulta.

5.4. **OWL**

El lenguaje de ontología web OWL está concebido para ser legible e interpretable tanto por los humanos como por las máquinas. Añade más vocabulario y semántica a RDF y RDF-S y dispone de lenguajes con expresividad creciente OWL Lite, OWL DL y OWL FULL que tienen sus bases en la lógica de descripciones [13].

El lenguaje OWL se describe mediante un conjunto de documentos, cada uno de los cuales cumple un propósito diferente y atiende a una audiencia diferente. A continuación, se proporciona una breve hoja de ruta para navegar a través de este conjunto de documentos:

- En la referencia [14] se encuentra disponible una descripción general de OWL que proporciona una presentación concisa de las funcionalidades del lenguaje OWL.

- En [15] se presenta la guía OWL para el uso del lenguaje mediante ejemplos didácticos.
- La referencia que especifica las primitivas del modelado de OWL se encuentra en [16].
- La semántica y la sintaxis abstracta del lenguaje están especificadas en el documento [17].
- El documento que contiene los casos de prueba del lenguaje OWL se encuentra en [18].
- El documento de casos de uso y requisitos de OWL se encuentra disponible en [19].

5.5. **SHACL**

SHACL *(Shapes Constraint Language)* es un lenguaje para validar grafos de datos RDF contra un conjunto de condiciones que son expresadas como grafos RDF.

El prefijo para el espacio de nombres para SHACL es el siguiente:

sh: *http://www.w3.org/ns/shacl#*

SHACL ofrece tres clases para la definición de las formas y están jerarquizas como se muestra en el Listado 5.6, tomado del vocabulario del lenguaje de restricciones de formas (SHACL) del W3C (https://www.w3.org/ns/shacl.ttl).

```
sh:Shape
        a rdfs:Class ;
        rdfs:label "Shape"@en ;
        rdfs:comment "A shape is a collection of constraints that may be targeted for certain
nodes."@en ;
        rdfs:subClassOf rdfs:Resource ;
        rdfs:isDefinedBy sh: .

sh:NodeShape
        a rdfs:Class ;
        rdfs:label "Node shape"@en ;
        rdfs:comment "A node shape is a shape that specifies constraint that need to be met
with respect to focus nodes."@en ;
        rdfs:subClassOf sh:Shape ;
        rdfs:isDefinedBy sh: .

sh:PropertyShape
        a rdfs:Class ;
        rdfs:label "Property shape"@en ;
        rdfs:comment "A property shape is a shape that specifies constraints on the values of a
focus node for a given property or path."@en ;
        rdfs:subClassOf sh:Shape ;
        rdfs:isDefinedBy sh: .
```

Listado 5.6: Jerarquía de formas en SHACL.

5.5.1 **SHACL CORE**

5.5.1.1. Forma

S es una forma SHACL si es un IRI *(Internationalized Resource Identifier)* o un nodo en blanco que está definido de alguna de las siguientes maneras:

- Es un ejemplar de *sh:NodeShape* o de *sh:PropertyShape*.
- Es el sujeto en una tripleta cuyo predicado es *sh:targetClass*, *sh:targetNode*, *sh:targetObjectOf* o *sh:targetSubjectOf*.
- Es el sujeto de una tripleta cuyo predicado es un parámetro. Un parámetro es esta definido en EBNF (Extended Backus–Naur Form) en el Listado 5.7.
- Es un valor de un parámetro que recibe formas y no toma listas, como *sh:node*, o un miembro de una lista SHACL que es un valor de un parámetro que recibe formas y toma listas, como *sh:or*.

```
parametro ::= 'targetNode' | 'targetObjectsOf' | 'targetSubjectsOf' |
        'deactivated' | 'severity' | 'message' |
        'class' | 'datatype' | 'nodeKind' |
        'minExclusive' | 'minInclusive' | 'maxExclusive' | 'maxInclusive' |
        'minLength' | 'maxLength' | 'pattern' | 'flags' | 'languageIn' |
        'equals' | 'disjoint' | 'lessThan' | 'lessThanOrEquals' |
        'qualifiedValueShape' | 'qualifiedMinCount' | 'qualifiedMaxCount' |
            'qualifiedValueShapesDisjoint' |
        'closed' | 'ignoredProperties' | 'hasValue' | 'in'
```

Listado 5.7: EBNF de término parámetro en SHACL.

5.5.1.2. Nodo forma (Node Shape)

Un nodo forma es una forma en el grafo de formas que no es el sujeto de una tripleta con *sh:path* como predicado. Se recomienda, pero no es obligatorio, que un nodo *forma* se declare como una instancia SHACL de *sh:NodeShape*. Las instancias SHACL de *sh:NodeShape* no pueden tener un valor para la propiedad *sh:path*. De manera informal, los nodos *forma* especifican las restricciones que deben cumplirse con respecto a los nodos de enfoque. A diferencia de las formas de propiedad, se aplican principalmente al propio nodo de enfoque *(sh:targetClass)*, no a sus valores de propiedad.

5.5.1.3. Propiedad forma (Property shape)

Una forma de propiedad es una forma en el grafo de formas que es el sujeto de una tripleta que tiene *sh:path* como predicado. Una forma tiene como máximo un valor para *sh:path*. Cada valor de sh:path en una forma debe ser una ruta de propiedad SHACL bien formada. Se recomienda, pero no es obligatorio, que una forma de propiedad se declare como una instancia SHACL de *sh:PropertyShape*. Las instancias SHACL de *sh:PropertyShape* tienen un valor para la propiedad *sh:path*.

5.5.2 **Validación y grafos**

La validación implica tomar un grafo de datos y un grafo de formas como entrada, y genera un informe de validación que contiene los resultados obtenidos. Por otro lado, la verificación de conformidad es una versión simplificada de la validación,

donde se produce un resultado booleano que indica si se cumple o no con los requisitos establecidos. Un sistema capaz de llevar a cabo la validación se conoce como procesador de validación.

5.5.2.1. Grafos de formas

Un grafo de formas es un grafo RDF que contiene cero o más formas que se pasan a un proceso de validación SHACL para que un grafo de datos puede ser validado en relación con las formas definidas.

5.5.2.2. Grafos de datos

Un grafo de datos es cualquier grafo RDF, ya sea un grafo RDF que contiene ejemplares de un determinado dominio, un grafo RDFS que contiene esquemas o metamodelos de un dominio específico, OWL que contiene ontologías, o incluso el propio grafo SHACL donde se pueden validar un conjunto de reglas contra otro conjunto de reglas (meta reglas).

5.5.2.3. Validación

Durante la validación, tanto el grafo de datos como el grafo de formas deben permanecer inmutables. Esto significa que, al finalizar la validación, ambos grafos deben ser idénticos a como estaban al comienzo de la validación. Los procesadores SHACL no deben realizar cambios en los grafos que utilizan para construir el grafo de formas o el grafo de datos, incluso si estos grafos forman parte de un almacenamiento RDF que permite modificaciones en los grafos almacenados.

Los procesadores SHACL pueden almacenar los grafos que crean, como un grafo que contiene los resultados de la validación. Esta operación puede afectar a los grafos existentes en el almacenamiento RDF, pero no debe modificar ninguno de los grafos utilizados para construir el grafo de formas o el grafo de datos. En resumen, el procesamiento SHACL es idempotente, lo que significa que el resultado de aplicar repetidamente el procesamiento a un conjunto de datos es el mismo que aplicarlo una sola vez.

5.5.2.4. Comprobación de conformidad

Un nodo foco se considera válido con respecto a una forma si y solo si el conjunto de resultados de la validación del nodo foco contra la forma está vacío y no se ha informado ninguna falla.

La verificación de conformidad devuelve verdadero si y solo si un nodo foco dado se ajusta a una forma dada, y falso en caso contrario.

Téngase en cuenta que algunos componentes de restricción en el núcleo de SHACL (por ejemplo, *sh:not*, *sh:or* y *sh:node*) dependen de la verificación de conformidad. En estos casos, los resultados de la validación utilizados para determinar el resultado de la verificación de conformidad se separan del proceso de validación circundante y, por lo general, no se incluyen en el mismo informe de validación (excepto, posiblemente, como valores de *sh:detail*).

5.5.2.5. Informe de validación

El informe de validación es el resultado obtenido tras llevar a cabo el proceso de validación, el cual proporciona información sobre la conformidad y contiene el conjunto completo de resultados obtenidos durante la validación. El informe de validación se describe utilizando el Vocabulario del Informe de Validación SHACL. Dicho vocabulario establece propiedades de RDF que se utilizan para representar información estructural, brindando orientación sobre cómo identificar o corregir violaciones en el grafo de datos.

Los procesadores que cumplen con SHACL tienen la obligación de generar un informe de validación que contenga todos los resultados de validación requeridos, tal como se describen la especificación. Además, los procesadores compatibles con SHACL tienen la capacidad de aceptar argumentos opcionales que permitan restringir la cantidad de resultados que son devueltos en el informe. Esta flexibilidad resulta fundamental en diversos escenarios de validación de conjuntos de datos a gran escala, garantizando así una mayor eficiencia en el procesamiento de la validación.

El Listado 5.8 representa el informe de validación para el resultado de la validación de un grafo de datos que cumple con un grafo de formas.

```
[  a sh:ValidationReport ;
   sh:conforms true ;
] .
```

Listado 5.8: Informe de validación conforme.

En el Listado 5.9 se presenta un ejemplo de un informe de validación de un grafo de datos que no cumple con un grafo de formas. Es importante destacar que SHACL no especifica un valor específico para *sh:resultMessage*, ya que se considera dependiente de la implementación particular

```
[ a sh:ValidationReport;
        sh:conforms false;
                sh:result [
                a sh:ValidationResult;
                sh:resultSeverity sh:Violation ;
                sh:focusNode ex:Bob;
                sh:resultPath ex:edad ;
                sh:value "veintidós" ;
                sh:resultMessage "ex:age espera un literal de tipo de datos
xsd:integer" ;
                sh:sourceConstraintComponent sh:DatatypeConstraintComponent ;
                sh:sourceShape ex:PersonaForma-edad ;
        ]
] .
```

Listado 5.9: Informe de validación con no conforme.

5.5.3 **Componentes de restricción centrales**

La Tabla 5.3 representa los componentes centrales de SHACL CORE.

Propiedad	Resumen y reglas de sintaxis
sh:class	El tipo de todos los nodos de valor. Los valores de sh:class en una forma son IRI.
sh:datatype	El tipo de datos de todos los nodos de valor (por ejemplo, xsd: entero). Los valores de sh:datatype en una forma son IRI. Una forma tiene como máximo un valor para sh:datatype.

Tabla 5.3: Los componentes centrales de SHACL CORE.

5.5.4 **SHACL-SPARQL**

SHACL-SPARQL ofrece un conjunto de constructos que permiten agregar flexibilidad en la definición de las formas SHACL al incorporar SPARQL dentro de su definición. En términos informales, las formas de propiedad especifican las restricciones que deben cumplirse en relación a los nodos que se pueden alcanzar desde el nodo de enfoque siguiendo directamente una propiedad específica (especificada como *IRI*) o cualquier otra ruta de propiedad SHACL, la cual se define mediante *sh:path*.

5.6. **Herramientas para las tecnologías de la web semántica**

En la actualidad, se dispone de una amplia variedad de herramientas para la web semántica, que abarcan tanto opciones de código abierto como soluciones comerciales. En este trabajo, nos centraremos exclusivamente en las herramientas de código abierto, las cuales gozan de una amplia adopción por parte de la comunidad de usuarios. Estas herramientas se presentan en forma de bibliotecas de programación, puntos de acceso (*Endpoints*) y entornos que facilitan el desarrollo e implementación de soluciones basadas en la web semántica.

5.6.1 **Librerías de programación**

En el ámbito de las tecnologías de la web semántica, existen diversas librerías disponibles para su incorporación en proyectos. Estas librerías se dividen en dos categorías: las de código abierto y las comerciales. Una de las opciones más populares en la categoría de código abierto es Apache Jena, que cuenta con una licencia Apache. Esta librería, escrita en Java, ofrece una API para la gestión de grafos RDF, lo cual permite realizar operaciones como la inserción, actualización y eliminación de datos en un grafo RDF. Además, Apache Jena proporciona funcionalidades para llevar a cabo consultas SPARQL sobre los grafos y realizar validaciones utilizando el lenguaje de restricciones SHACL. Gracias a su amplia adopción y versatilidad, Apache Jena se ha convertido en una opción de referencia para el uso de tecnologías de la web semántica en proyectos basados en código abierto.

5.6.2 **Endpoints**

Apache Fuseki

Uno de los endpoints más reconocidos en la categoría de código abierto es Apache Fuseki, el cual es un servidor SPARQL con diversas opciones de implementación. Puede ser ejecutado como un servicio del sistema operativo, como una aplicación web Java (archivo WAR) o como un servidor independiente.

Fuseki ofrece dos variantes: una como una "aplicación web" de un solo sistema que incluye una interfaz de usuario para la administración y consulta, y otra como una versión "principal" que se puede integrar fácilmente en implementaciones más grandes, incluso mediante el uso de Docker o ejecuciones integradas. Ambas variantes utilizan el mismo motor de protocolo central y comparten un formato de archivo de configuración común.

Este servidor proporciona soporte para los protocolos SPARQL 1.1 tanto para consultas como para actualizaciones, así como para el protocolo SPARQL Graph Store. Además, Fuseki está estrechamente integrado con el componente de almacenamiento y consulta RDF llamado TDB, lo que brinda una capa de almacenamiento transaccional persistente y sólida. Asimismo, incorpora la capacidad de realizar consultas de texto mediante Jena.

5.6.3 **Entornos y herramientas**

Enterprise architect:

Enterprise Architect es una plataforma de modelado que ha implementado los estándares propuestos por la OMG. Este entorno también ha incorporado ODM *(Ontology Metamodel Definition)*, que es esencialmente un perfil UML diseñado para las ontologías OWL y RDF. Proporciona funcionalidades que permiten exportar o serializar los modelos OWL o RDF en formato RDF/XML. Esto facilita la interoperabilidad y la integración de los modelos ontológicos con otras herramientas y sistemas que soportan este estándar.

Topbraid

TopQuadrant, Inc. es la empresa encargada del desarrollo de una suite integrada de herramientas en el ámbito de la web semántica. Esta suite cuenta con componentes principales, como TopBraid Composer, utilizado para editar ontologías OWL y RDF Schema, y el servidor de aplicaciones web semánticas, TopBraid Live.

TopBraid Composer es una herramienta completa para el desarrollo de aplicaciones basadas en la web semántica, que cubre todas las etapas del ciclo de vida de desarrollo. Además de funcionar como un editor de ontologías con soporte de refactorización, Topbraid Composer también puede ser utilizado como un entorno para ejecutar reglas, consultas, razonadores y mash-ups. Está basado en el framework Eclipse, lo que permite personalizar y ampliar su funcionalidad mediante complementos Java, lo que facilita el desarrollo ágil de aplicaciones basadas en la web semántica en una única plataforma. En [20] se proporciona una implementación de código abierto de SHACL basada en Apache Jena.

Protégé

El Centro Stanford para la Investigación en Informática Biomédica de la Facultad de Medicina de la Universidad de Stanford es el responsable del desarrollo de Protégé. Este software es una herramienta para la creación de sistemas basados en el

conocimiento y un recurso para la investigación en informática biomédica y otras áreas.

Protégé es un editor de ontologías y un marco de trabajo gratuito y de código abierto para la creación de sistemas inteligentes. Cuenta con una comunidad activa de usuarios académicos, gubernamentales y corporativos que utilizan Protégé para desarrollar soluciones basadas en el conocimiento en diversas áreas, como la biomedicina, el comercio electrónico y el modelado organizacional. Esta comunidad respalda el desarrollo y la mejora continua de Protégé como una herramienta clave para la creación de sistemas inteligentes.

A través de su interfaz gráfica, Protégé permite modelar ontologías, escribir y ejecutar consultas SPARQL, editar y ejecutar reglas SWRL (*Semantic Web Rule Language*), e instalar complementos para ampliar sus funcionalidades, como el uso de SHACL, entre otros.

Además, Protégé proporciona varios razonadores, como ELK, HERMIT, ONTOP y PELLET, que permiten realizar inferencias y razonamiento en ontologías.

5.7. **UML**

UML y BPMN *(Business Proces Modeling Notation)* son estándares desarrollados y refinados por el OMG, un consorcio internacional sin ánimo de lucro dedicado al desarrollo y mantenimiento de estándares relacionado con la orientación a objetos.

Los conceptos Elemento y Relación constituyen la base fundamental para el resto de los conceptos de modelado en UML. En efecto, los elementos son las unidades básicas que conforman los modelos en UML, mientras que las relaciones representan las interacciones y asociaciones entre ellos.

En este sentido, se puede afirmar que los conceptos de Elemento y Relación son los pilares sobre los que se construye todo el edificio teórico y práctico del modelado en UML. A partir de ellos se derivan otros conceptos fundamentales como clases, objetos, herencia, interfaces, entre otros, que son esenciales para el desarrollo de aplicaciones y sistemas de software.

Por tanto, se puede concluir que comprender adecuadamente los conceptos raíz de Elemento y Relación es crucial para el dominio y aplicación eficaz de UML en el desarrollo de software.

El lenguaje de modelado proporcionado es visual mediante diagramas tanto para al aspecto estructural como para el aspecto comportamental y su clasificación se muestra en la Figura 5.4, tomada de la especificación OMG para UML *(the taxonomy of structure and behavior diagrams)*.

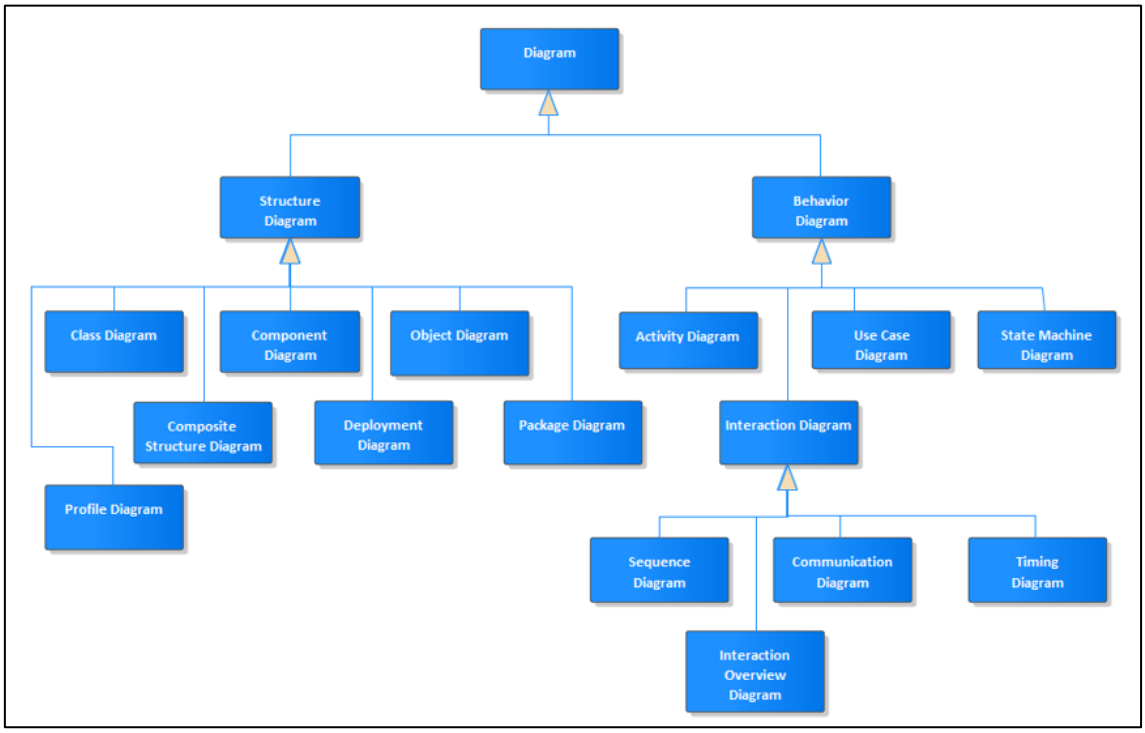


Figura 5.4: La taxonomía de los diagramas estructurales y de comportamiento.

En la especificación proporcionada por OMG para UML se presenta la sintaxis y la semántica de los conceptos de modelado.

5.7.1 **Aspecto estructural**

Los conceptos raíz de todo el modelado en UML son el concepto *Element* y el concepto *Relationship*. Estos dos conceptos son los que UML utiliza para modelar otros conceptos. Como se puede observar en RDF los dos elementos fundamentes para el modelado o representación del conocimiento son conceptos y relaciones entre ellos. Esto presenta una similitud clara entre el patrón de modelado de RDF y UML en términos de meta modelado.

En la parte estructural los dos diagramas que son de interés para el trabajo actual son el diagrama de clase y el diagrama de objeto. Pero antes conviene mencionar los elementos básicos sobre los cuales están construidos los elementos estructurales de UML (ver Figura 5.5)

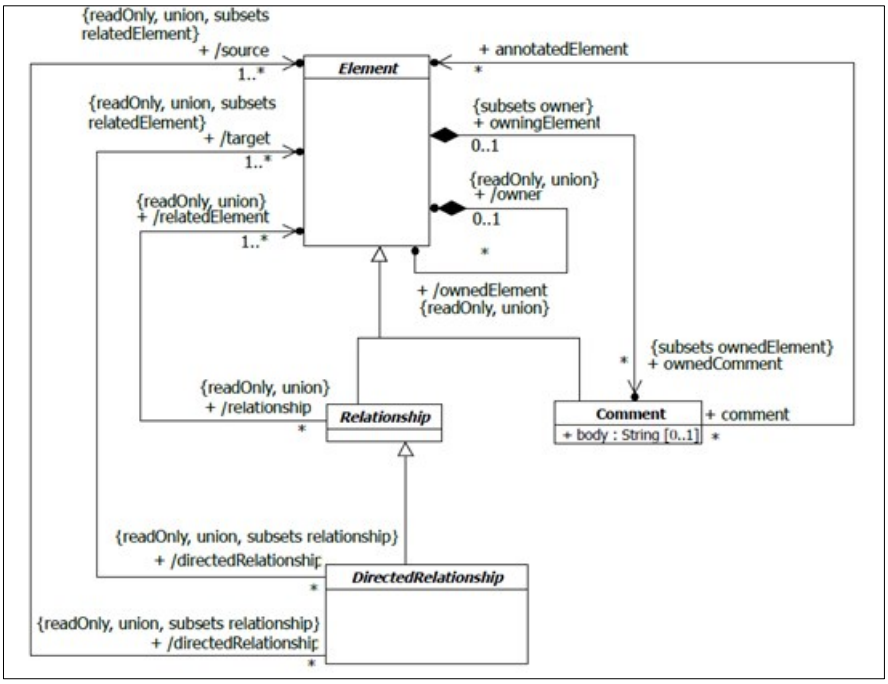


Figura 5.5: Modelado de los elementos raíz de UML (Fuente: Figure 7.1 de la Especificación UML de OMG V2.5.1.)

Un elemento se refiere a un componente fundamental dentro de un modelo. Los elementos descendientes tienen la tarea de proveer una semántica apropiada al concepto que representan. Cada elemento posee la habilidad inherente de contener otros elementos. Si un elemento es eliminado del modelo, todos los elementos que le pertenecen también son eliminados necesariamente. La sintaxis abstracta para cada tipo de elemento especifica qué otros tipos de elementos pueden ser contenidos. Cada elemento dentro del modelo debe ser propiedad exclusiva de otro elemento dentro del mismo modelo, exceptuando los paquetes de nivel superior

Cualquier tipo de Elemento es capaz de contener Comentarios. Los comentarios que pertenecen a un elemento no aportan semántica al modelo, sin embargo, pueden representar información valiosa para el lector de este.

Una relación es un componente que define un tipo de conexión entre otros elementos. Los descendientes de las relaciones se encargan de proporcionar una semántica adecuada al concepto que representan. Una relación dirigida representa una conexión entre un grupo de elementos dentro del modelo de origen y otro grupo de elementos dentro del modelo de destino. Se considera que una relación dirigida se dirige desde los elementos de origen hacia los elementos de destino.

Como veremos más delante también las ontologías o RDF y RDFS están también basadas en conceptos y relaciones y esto servirá de mapeo entre los dos paradigmas. Pero en UML una relación o atributo no puede existir por sí solo siempre tiene que ser propiedad de otro; esta restricción no existe en RDF donde un concepto o una relación puede tener existencia propia sin ninguna relación de propiedad con respecto a otros elementos.

Los dos diagramas claves para el modelado en UML son el diagrama de clases y el diagrama de objetos. Como se verá más adelante corresponden al RDF Schema y RDF de datos.

5.7.2 **Aspecto comportamental:**

El comportamiento en UML especifica o modela cómo las instancias de los modelos estructurales cambian con el tiempo.

Los constructos fundamentales ofrecidos por UML para modelar el comportamiento son *Behavior*, *Event* y *Trigger*. Donde *Behavior* es el concepto básico del comportamiento para crear modelos de comportamiento. La ejecución del comportamiento la puede provocar una invocación directa, mediante la creación de objeto que contiene el comportamiento o mediante la interacción entre objetos. La ocurrencia de un evento puede desencadenar un comportamiento.

UML permite entre otros los siguientes mecanismos de especificación de comportamiento: Maquinas de estados para modelar autómatas finitos, Diagramas de actividades definidos mediante grafos de tipo red de Petri y diagramas de Interacciones. Cada problema de modelado le conviene un mecanismo u otro u una combinación de ellos.

Se dará una definición más rigurosa al comportamiento entendiéndolo como un constructo basado en el concepto de transformación.

En UML la especificación de los comportamientos está representada en el diagrama de clases de la Figura 5.6.

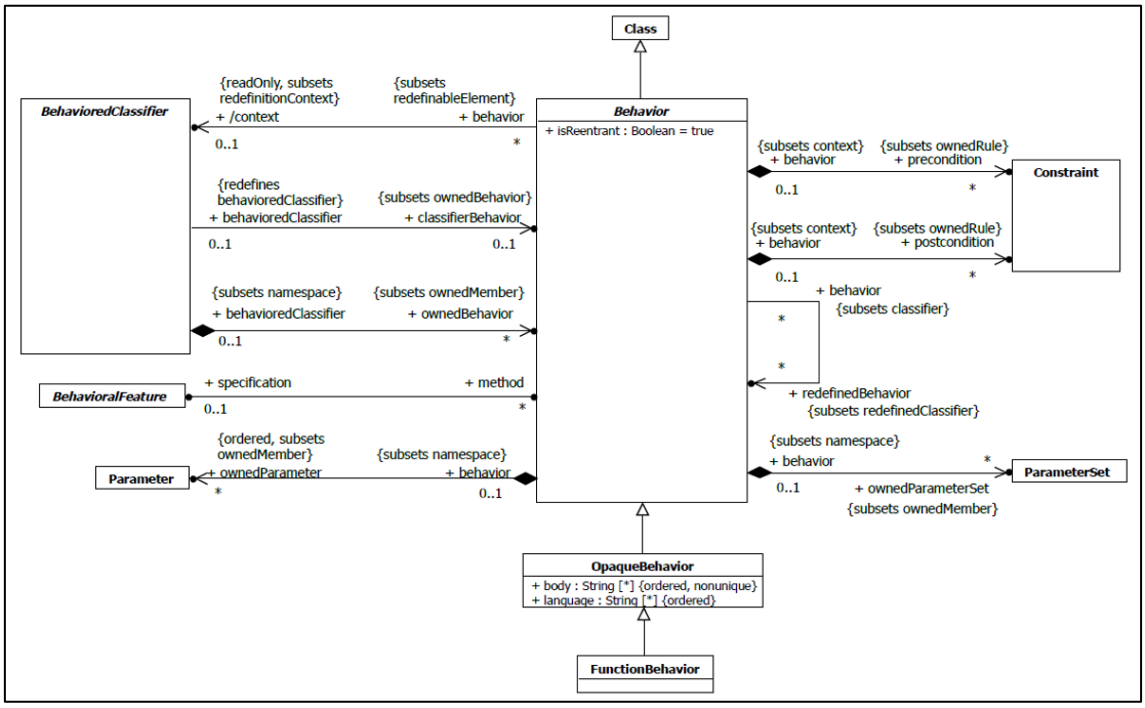


Figura 5.6: Comportamiento en UML. (Fuente: Figure 13.1 de la Especificación UML de OMG V2.5.1.)

La especificación del comportamiento en UML versión 2.5.1 considera que “Un Comportamiento es una especificación de eventos que pueden ocurrir dinámicamente a lo largo del tiempo (ver también la subcláusula 13.3 sobre el modelado explícito de Eventos en UML). Esta especificación puede prescribir específicamente qué eventos pueden ocurrir en qué situaciones, describiendo el comportamiento emergente o ilustrativo de posibles secuencias de sucesos. Cada comportamiento define al menos un evento, el evento de su invocación. Un Comportamiento puede invocarse directamente, a través de una Característica de Comportamiento que se implementa como un método o como *classifierBehavior* de un *BehavioredClassifier*.”.

Como se verá más adelante se redefinirá el concepto de evento y se formalizará la idea de las circunstancias que generan los eventos. También se define la *performedTask* (tarea) como la que se encarga de la implementación del comportamiento.

Un punto de gran importancia en el diagrama anterior son los conceptos *precondition* (precondición) y *postcondition* (postcondición) como se verá más delante, servirán para la creación de un lenguaje visual para el comportamiento en el paradigma ontológico.

Las precondiciones para un comportamiento son aquellas que se deben cumplir en el momento de invocar el comportamiento. Estas precondiciones pueden considerarse en la especificación detallada del comportamiento. La semántica de la invocación de un comportamiento cuando las precondiciones no se cumplen se define intencionalmente como ambigua.

En cuanto a las postcondiciones para un comportamiento, estas definen las condiciones que deben ser verdaderas al completar con éxito la invocación del comportamiento, siempre y cuando se cumplan las precondiciones. Estas postcondiciones también se consideran en la especificación detallada del comportamiento.

Los parámetros asociados a un comportamiento se les puede asignar valores que pueden ser de solo entrada "in" o entrada y salida "inout".

Un *OpaqueBehavior* se define como un tipo de Comportamiento en UML, sin embargo, su especificación se presenta en un lenguaje textual distinto al de UML. El *OpaqueBehavior* contiene un cuerpo que se compone de una secuencia de cadenas de texto que se utilizan para representar diferentes formas de especificar el comportamiento requerido. Para indicar los idiomas en los que se interpretarán cada una de las cadenas del cuerpo, se utiliza una secuencia de Cadenas de idioma que se asocian con las cadenas del cuerpo por orden. Aunque la especificación UML no define cómo se deben interpretar las cadenas del cuerpo en relación con un idioma específico, otras especificaciones pueden definir Cadenas de lenguaje que se utilizarán para indicar la interpretación en relación con esas especificaciones, como "OCL" para las expresiones que se interpretarán según la especificación OCL.

En la definición de los constructos para el comportamiento para las ontologías la especificación se lleva a cobo como una "*Glass box*" y con la idea de la ejecutabilidad directa.

Como constructos de comportamiento en el paradigma ontológico se crearán los conceptos *inputModel* y *OutputModel* que tienen una similitud con los parámetros asociados a un comportamiento.

En la parte del comportamiento los tres diagramas que se consideran de interés y los que se va a poner el foco en ellos son: diagrama de actividad, máquina de estados y el diagrama de interacción.

5.8. Modelo y Notación de Procesos de Negocio (BPMN)

Las personas involucradas en el dominio del negocio tienen una preferencia por presentar los procesos utilizando diagramas visuales, como los diagramas de flujo. Esta elección se debe a que los diagramas visuales permiten alcanzar un entendimiento común del modelo de negocio. Sin embargo, existía una brecha entre cómo los analistas de negocio presentaban los procesos y cómo eran ejecutados por los lenguajes formales. Esta brecha se ha resuelto mediante el mapeo entre el lenguaje visual utilizado y el lenguaje formal de ejecución. Este mapeo ha permitido alinear y vincular de manera efectiva el lenguaje visual con el lenguaje formal, proporcionando coherencia y facilitando la transición entre la representación gráfica y la ejecución real de los procesos de negocio.

BPMN es el acrónimo en ingles de "*Business Process Model and Notation"*, y es una notación gráfica estandarizada utilizada para representar procesos de negocio. Esta metodología permite crear una representación visual de las actividades, eventos y flujos que conforman un proceso de negocio, lo cual resulta especialmente beneficioso en la gestión de procesos de negocio (BPM) cuyo acrónimo en inglés es "*Business Process Management*" y en las iniciativas de mejora de procesos.

BPMN proporciona un conjunto de símbolos y reglas para describir de manera clara y concisa los pasos y las interacciones dentro de un proceso de negocio. Estos símbolos representan actividades, eventos, decisiones, flujos de secuencia y otros elementos clave del proceso. Al utilizar BPMN, los equipos de trabajo pueden comunicarse de manera efectiva, comprender y analizar los procesos existentes, identificar áreas de mejora y diseñar nuevos procesos de manera más eficiente.

En los diagramas de BPMN se utilizan símbolos similares a los de los diagramas de flujo para representar los diferentes elementos que conforman un proceso de negocio. Entre los símbolos más utilizados se encuentran las tareas, representadas por rectángulos; las puertas de enlace, representadas por rombos; los eventos, representados por círculos; y los conectores, representados por flechas.

Además, BPMN incluye un conjunto de reglas que establecen cómo deben utilizarse estos símbolos para representar diferentes tipos de procesos y flujos de

procesos. Por ejemplo, estas reglas definen cómo representar caminos de ramificación y unión, cómo manejar bucles y repeticiones, y cómo gestionar excepciones y errores.

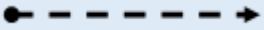
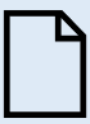
BPMN, por lo tanto, proporciona una metodología eficaz, eficiente y flexible para modelar y visualizar procesos de negocio, lo que resulta de gran ayuda para las organizaciones en la mejora de su eficiencia, la reducción de costos y el incremento de su rendimiento en general.

Como se verá más adelante, la interoperabilidad entre la parte humana interesada en el dominio de negocio y sus procesos se logra mediante la adopción del BPMN como estándar. Esto se debe a que BPMN ofrece un conjunto de diagramas que permiten realizar análisis, diseño y gestión de los procesos de negocio. BPMN tiene la capacidad de mapear el lenguaje visual utilizado en los diagramas con un lenguaje ejecutable llamado WSBPEL *(Web Services Business Process Execution Language*). Sin embargo, es importante tener en cuenta que BPMN no ofrece capacidad de razonamiento y no está diseñado ni a nivel visual ni a nivel de serialización como un grafo. Además, es importante mencionar que la validación de los modelos obtenidos con BPMN no se realiza mediante un lenguaje estándar y la implementación se realiza de manera opaca.

Sin embargo, las estructuras visuales de los diagramas BPMN son en su base grafos, aunque desde OMG no están concebidos como tales. Pero esto servirá para alinear este tipo de diagramas con las ontologías proporcionadas por las tecnologías de la web semántica cuya base es el grafo sobre el cual se llevan a cabo tareas de razonamiento.

La Tabla 5.4 reproduce la tabla 7.1 de la especificación los elementos básicos de BPMN Versión 2.0.2 proporcionada por OMG.

Elemento	Descripción	Notación
Evento	Un evento se refiere a la ocurrencia de algo específico en el contexto de un proceso o una coreografía. Su presencia implica un cambio en el flujo del proceso. Cabe destacar que cada evento está sujeto a una condición que lo desencadena y, a su vez, provoca consecuencias que tienen un impacto en el contexto en el que se desarrolla. El símbolo visual que lo representa es un círculo vacío que permite poner un marcador para poder cambiar la semántica del evento. Hay tres tipos de eventos: inicio, intermedio y fin.	

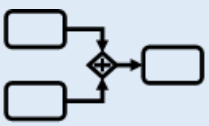
Elemento	Descripción	Notación
Actividad	Una actividad es un término genérico para el trabajo que se realiza en un proceso. Una actividad puede ser atómica o no atómica (compuesta). Los tipos de actividades que forman parte de un modelo de proceso son: el subproceso y la tarea, que son rectángulos redondeados. Las actividades se utilizan en procesos y en coreografías.	
Compuerta	Se utiliza un *Gateway* para controlar la divergencia y convergencia de flujos de secuencia en un proceso y en una coreografía. Así, se determinan las ramificaciones, las bifurcaciones, la fusión y unión de caminos. Los marcadores internos indicar el tipo de control del comportamiento.	
Flujo de secuencia	Se utiliza un flujo de secuencia para mostrar el orden en que Las actividades se realizarán en un proceso y en una coreografía.	
Flujo de mensaje	Se utiliza para mostrar el flujo de mensajes entre dos participantes que están preparados para enviarlos y recibirlos. En BPMN, dos piscinas separadas en un diagrama de colaboración representarán los dos participantes .	
Asociación	Una asociación se utiliza para vincular información y artefactos con elementos gráficos BPMN. Anotaciones de texto y otros artefactos se pueden asociar con los elementos gráficos. Una punta de flecha en la asociación indica una dirección de flujo (por ejemplo, datos).	
Piscina	Una piscina es la representación gráfica de un participante en una colaboración. Puede usarse como un contenedor para particionar un conjunto de actividades.	
Carril	Dentro de una piscina, los carriles se usan para organizar y categorizar actividades.	
Objeto de datos	Los objetos de datos proporcionan información acerca de qué actividad se requiere realizar y/o lo que van a producir. Los objectos de datos proporcionan información a los procesos.	
Mensaje	Un mensaje se utiliza para representar el contenido de una comunicación entre dos Participantes.	

Elemento	Descripción	Notación
Grupo	Es una agrupación de elementos gráficos que están dentro de la misma categoría. Este tipo de agrupación no afecta a la secuencia flujos dentro del grupo. El nombre de la categoría aparece en el diagrama como la etiqueta del grupo. Las categorías se pueden utilizar para la documentación o para propósitos de análisis. Los grupos son una forma en la que las categorías de objetos se pueden mostrar visualmente en el diagrama.	
Anotación textual	Las anotaciones de texto son un mecanismo para proporcionar información de texto adicional para el lector de un Diagrama BPMN.	

Tabla 5.4: Constructos básicos de BPMN.

En la definición de BPMN se indica que cada evento está sujeto a una condición que lo desencadena y provoca unas consecuencias que tienen un impacto en el contexto. Cómo se verá en el enfoque propuesto que aborda el lenguaje visual para el modelado con ontologías, para especificar el comportamiento estos conceptos aparecen de nuevo, pero tendrán entidad propia con su sintaxis y semántica que son *preCondition, perfomedTask, outputModel y postCondition*. También se definen los objectos de datos como elementos que proporcionan información a los procesos. Esto es lo que corresponderá al *inputModel* en el enfoque propuesto.

Es de interés para este trabajo describir algunos elementos que son considerados por la especificación de BPMN como extensión de los elementos básicos mencionados en la tabla anterior (ver Tabla 5.5).

Elemento	Descripción	Notación
Flujo condicional	Un flujo de secuencia puede tener una expresión que es evaluada en tiempo de ejecución para determinar si flujo de secuencia es usado o no.	
Dato de entrada/salida	Los objetos de datos proporcionan información sobre las actividades que requieren ser realizadas y/o qué producen.	
Join	BPMN utiliza el término “JOIN” para referirse a la combinación de dos o más caminos paralelos en una ruta (también conocida como AND-Join o sincronización). Se utiliza una puerta de enlace paralela para mostrar la unión de múltiples flujos de secuencia.	

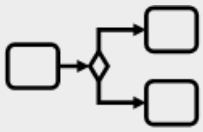
Elemento	Descripción	Notación
Exclusivo	Esta Decisión representa un punto de bifurcación donde las alternativas se basan en expresiones condicionales contenidas en el flujo de secuencia saliente. Sólo una de las alternativas será elegida.	
Bifurcación	Se pueden utilizar múltiples flujos de secuencia salientes. Esto representa el flujo "descontrolado" es el método preferido para la mayoría de las situaciones.	

Tabla 5.5: Extensión de los elementos básicos de BPMN.

En esta extensión de los elementos de BPNM se menciona el flujo condicional y los datos de entrada y salida, que tendrán, cuando se especifica el enfoque propuesto con respecto al comportamiento, su propia entidad: *inputModel* y *outPutModel*. También se verá que la misma transición tendrá su *precondition* para ejecutarse o no.

Estos elementos sirven en este trabajo para aportar un lenguaje visual para el comportamiento con el enfoque ontológico y serán unos constructos visuales con su correspondiente semántica y representación en RDF.

6. Ingeniería dirigida por modelos

La ingeniería contiene cinco elementos básicos: el ingeniero, la realidad en la que se lleva a cabo el proceso de ingeniería, el modelo que representa tanto la realidad como el artefacto que el ingeniero pretende desarrollar, y el proceso de ingeniería en sí mismo, (ver Figura 6.1.).

Cabe señalar que una vez que el artefacto ha sido desarrollado, vuelve a formar parte de la realidad debido a su existencia física o conceptual, y también interactúa con la realidad que lo aloja. Para desarrollar un artefacto, se requiere un modelo tanto del propio artefacto como de la realidad en la que se integrará. El ingeniero utiliza los recursos de la realidad y cuenta con un modelo tanto de esta como del artefacto que desea fabricar, para llevar a cabo su proceso de ingeniería. Este proceso implica un conjunto de competencias necesarias para transformar la realidad y obtener un artefacto con una determinada utilidad.

Es importante destacar que tanto la realidad, los modelos de la realidad, los modelos de los artefactos para satisfacer una necesidad específica y el conocimiento del dominio en cuestión tienen un carácter evolutivo.

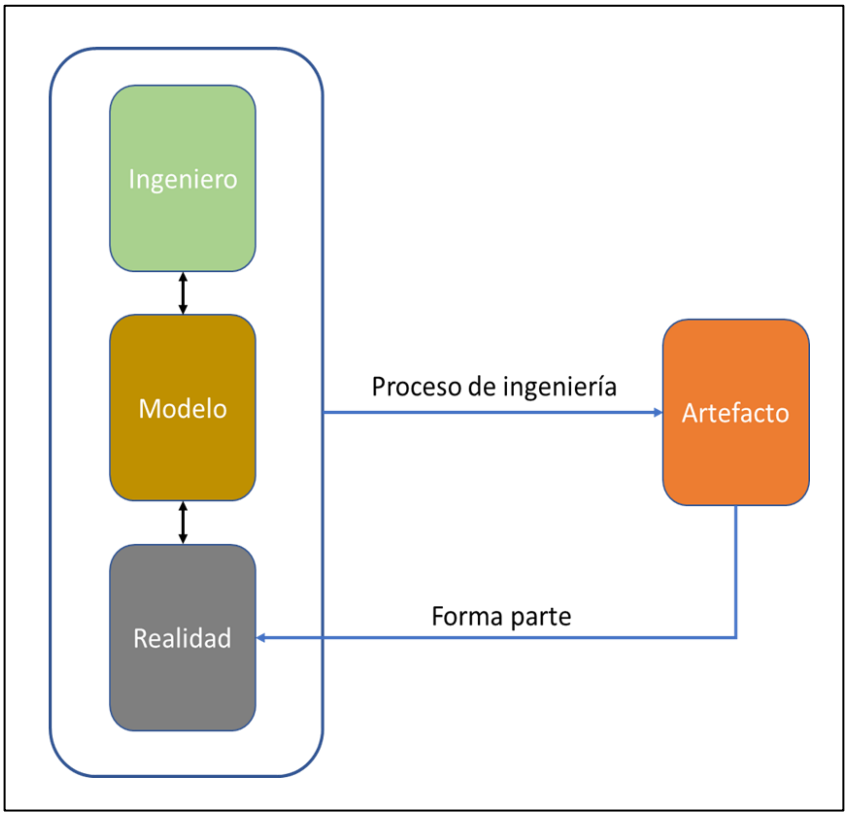


Figura 6.1: Rol del modelado en el proceso de ingeniería.

El MDE es un paradigma utilizado para abordar los procesos de ingeniería con el objetivo de producir artefactos con una utilidad específica. En este paradigma, todos los dominios se representan mediante modelos. Un modelo es una simplificación de la realidad que se centra en un conjunto limitado de conceptos relevantes para la utilidad del artefacto en desarrollo. Un metamodelo, por su parte, es una descripción del modelo.

Los procesos de modelado son herramientas conceptuales que resultan útiles tanto desde una perspectiva metodológica como productiva para abordar el proceso de ingeniería. Y para ello existen diversos lenguajes formales empleados para describir metamodelos, entre los cuales se encuentran los siguientes:

EBNF

EBNF es un lenguaje utilizado para definir gramáticas libres de contexto. También se considera un metalenguaje orientado a la construcción de lenguajes. Las producciones son reglas que especifican cómo se forman las construcciones gramaticales en un lenguaje y se definen utilizando símbolos terminales y no

terminales, y operadores especiales para indicar las relaciones entre ellos. A continuación, se presentan algunos constructos de EBNF.

- Producción básica:

  <no-terminal> ::= <expresión>

  Se puede construir <no-terminal> usando <expresión>

- Alternativa:

  <no-terminal> ::= <expresión1> | <expresión2>

  Se puede construir <no-terminal> usando como opciones <expresión1> o <expresión2>

- Concatenación:

  <no-terminal> ::= <expresión1> <expresión2>

  Se puede construir <no-terminal> usando la concatenación de <expresión1> seguida de <expresión2>

- Repetición:

  <no-terminal> ::= <expresión>*

  Se puede construir <no-terminal> repitiendo <expresión> cero o más veces.

RDF

El modelado se fundamenta en el concepto de la tripleta (sujeto, predicado, objeto), la cual permite representar el modelo como un grafo dirigido.

UML

Desde la perspectiva de UML, todas las entidades se representan como clases. Una clase se define mediante un nombre y puede tener atributos (propiedades o características) y operaciones (funciones o métodos) asociados a ella.

FOL

La lógica de primer orden, gracias a su lenguaje algebraico, se utiliza para definir modelos con un riguroso fundamento lógico-matemático.

MOF

OMG proporciona un estándar mediante el cual es posible definir los metamodelos de los diferentes paradigmas de modelado. Este estándar se denomina MOF, en el momento de redactar este trabajo se encuentra en la versión 2.5.1.

MOF es considerado por OMG como una norma Internacional que establece las bases para definir metamodelos en la familia de lenguajes MDA de OMG, y se fundamenta en una simplificación de las capacidades de modelado de clases de UML2.

También es considerado como un marco de gestión de metadatos que ofrece una solución abierta e independiente de la plataforma.

El modelo MOF 2 se compone de dos paquetes principales: Essential MOF (EMOF) y Complete MOF (CMOF). EMOF está diseñado para adaptarse a las capacidades de los lenguajes de programación orientados a objetos y de los mapeos a XMI o JMI. MOF completo (CMOF) ofrece todas las capacidades de meta modelado de MOF 2. Para lograr esto se extiende a través del uso de reflexión. La Figura 6.2 y la Figura 6.3 muestras los diagramas de clase de las clases EMOF de UML y los tipos de datos EMOF.

Cabe destacar que los conceptos esenciales del modelado son el *Clasificador* y la *Instancia*, también conocidos como la Clase y el Objeto, y la capacidad de navegar desde una instancia hasta su metaobjeto, que es su clasificador.

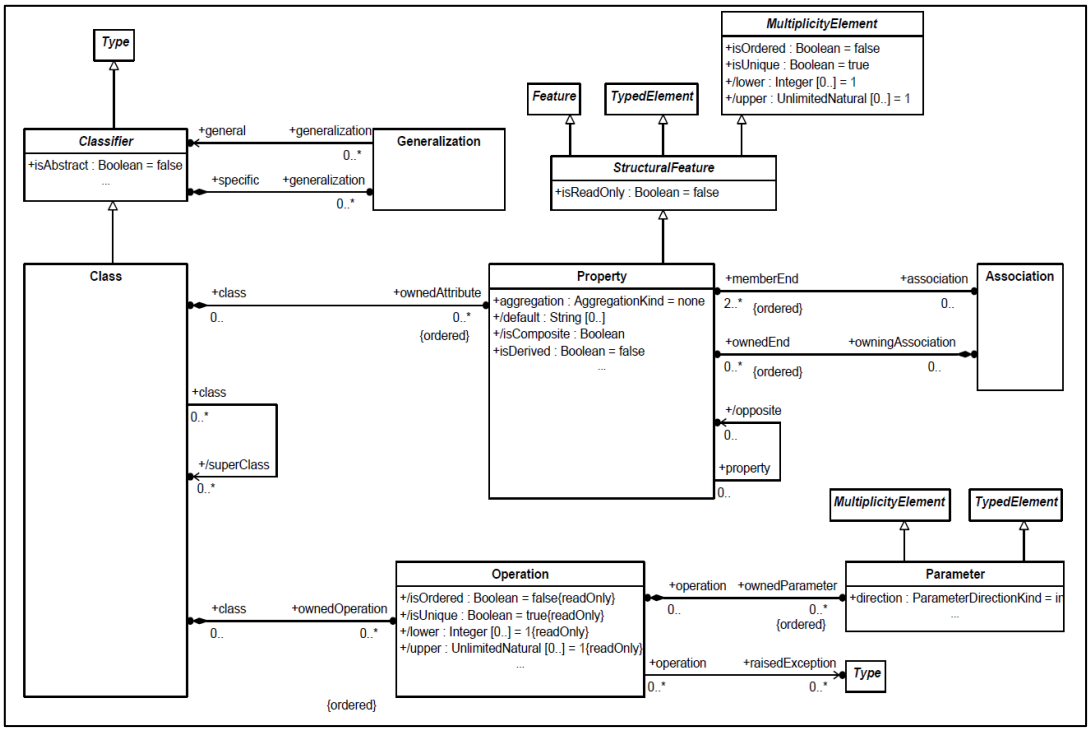


Figura 6.2: Clases EMOF de UML (Fuente: Figura 12.2 OMG Meta Object Facility (MOF) Specification Versión 2.5.1)

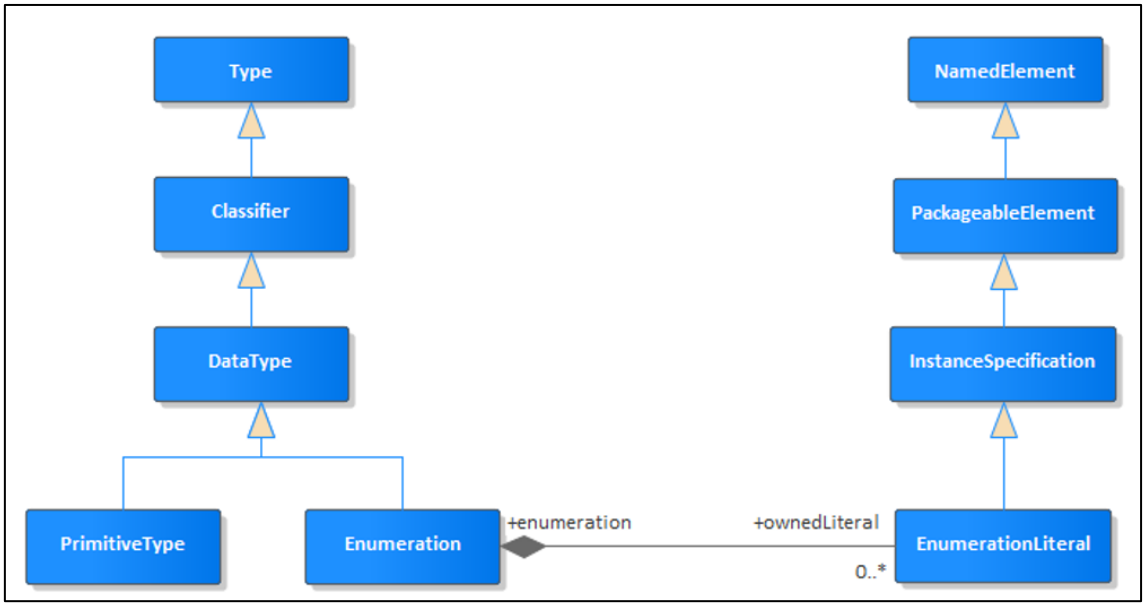


Figura 6.3: Tipos de datos EMOF (Fuente: Figura 12.3 OMG Meta Object Facility (MOF) Core Specification Versión 2.5.1)

7. Alineamiento estructural

El alineamiento estructural es un esfuerzo para modelar explícitamente tanto los conceptos como el metamodelo de un paradigma en relación con otro. En el caso de OMG y ODM, no se ha tenido en cuenta el mapeo estructural o de metamodelo de RDF a UML, solo se han mapeado los conceptos elementales.

Es conveniente determinar si el alineamiento se realiza a nivel CIM, PIM o PSM. El objetivo principal es lograr la ejecución de la especificación requerida en la máquina, además de contar con un lenguaje de modelado que abarque los tres niveles. Un lenguaje común presente en todos ellos es el grafo, que se encuentra en BPMN, reconocido por OMG como el estándar para el CIM, y en UML, considerado por OMG como el estándar para el PIM. Además, RDF se basa en el concepto de grafo dirigido, con una semántica y lógica simple y rigurosa, lo cual le confiere capacidad de razonamiento y ejecución directa. La transformación de CIM a PIM y su posterior PSM presenta un esfuerzo considerablemente inferior en comparación con su equivalente en UML.

Como se analizará en la sesión 9, se plantea una transformación de UML a RDF y de OCL a SHACL con el propósito de distinguir la transformación de la estructura de la transformación de las reglas aplicadas a dicha estructura.

La transformación bidireccional básica del grafo que representa una tripleta (sujeto, predicado, objeto) en RDF es (Clase, propiedad, Clase) o (objeto, propiedad, objeto). El caso en el que el objeto no es un literal se ilustra en la Figura 7.1.

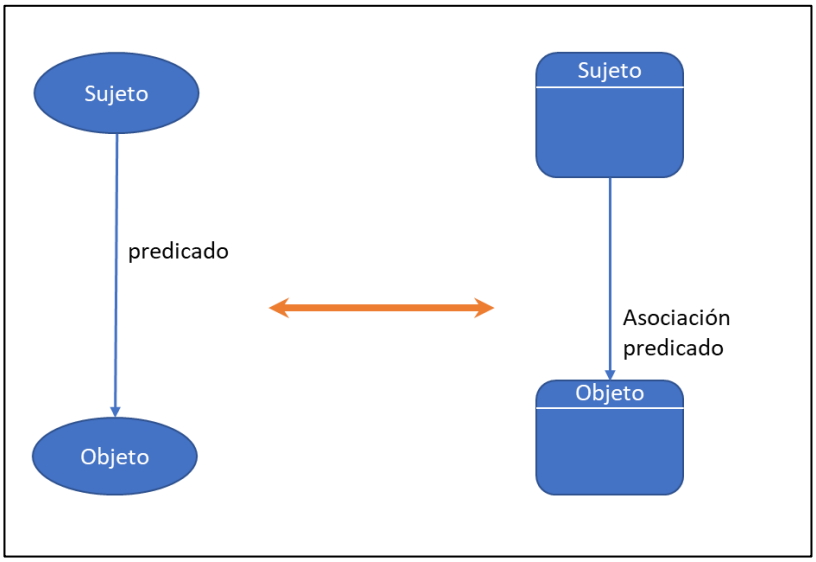


Figura 7.1: Transformación básica entre RDF y UML

La transformación bidireccional, en caso de que un sujeto S tenga varios predicados y sus objetos no sean literales, se muestra en la Figura 7.2. En esta figura, la clase S tiene tantas asociaciones como predicados tiene el sujeto S.

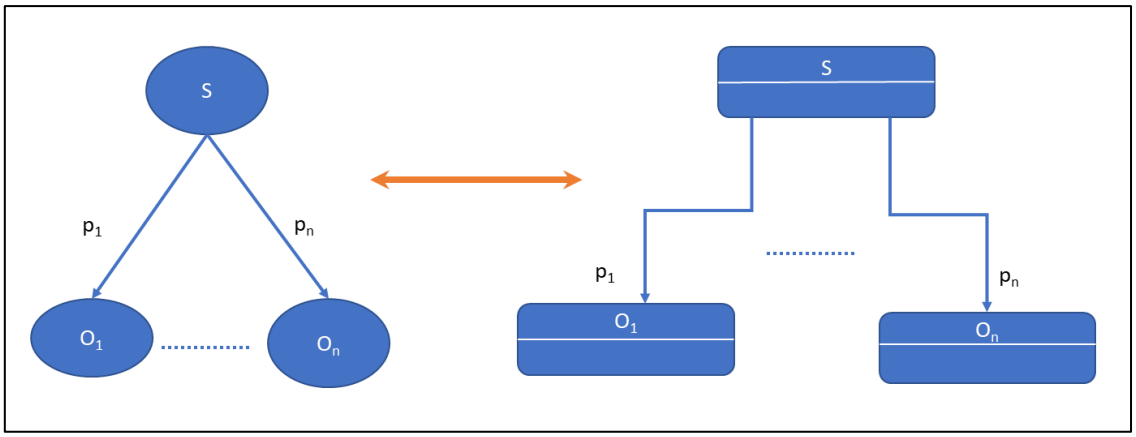


Figura 7.2: Transformación entre RDF y UML en el caso de Objetos no literales

En el caso más general, cuando el sujeto S tiene un conjunto de predicados con objetos literales y otro conjunto de predicados con objetos no literales, los literales se representan como atributos en la clase UML, donde el tipo del atributo corresponde al tipo del objeto. Por otro lado, los predicados con objetos no literales se representan como asociaciones, como se muestra en la Figura 7.3.

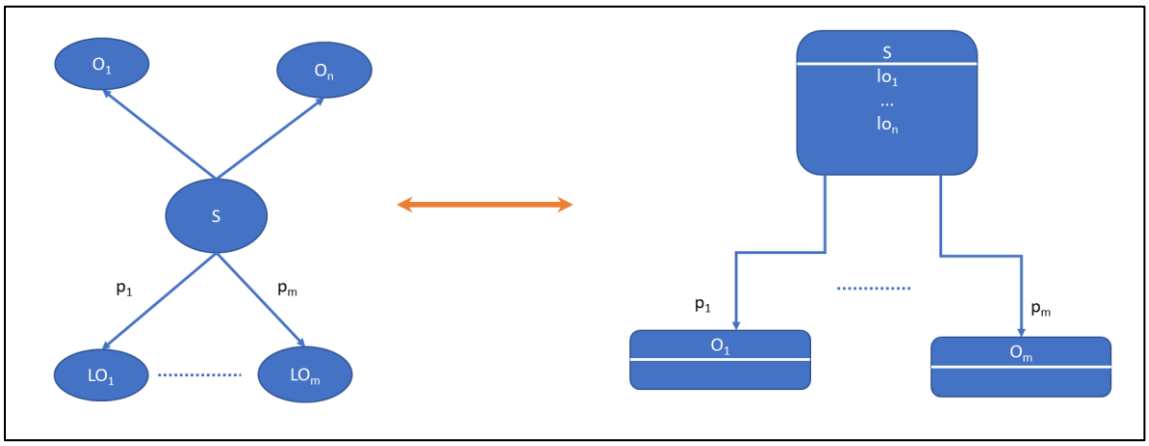


Figura 7.3: Transformación entre RDF y UML en el caso general

7.1. Transformación de UML a RDF/OWL

OMG, consciente de que su gran esfuerzo ha sido la transformación de RDF/OWL a UML, publicó en septiembre de 2021 una especificación con el objetivo de realizar el mapeo de MOF a RDF (MOF2RDF). Hay que tener en cuenta que algunos conceptos en UML no se pueden mapear directamente en RDF/OWL, como es el caso del concepto de "abstract". La propuesta de OMG en este caso consiste en encapsular la semántica de UML en RDF, lo cual puede resultar en cierta dificultad al trabajar con la esencia misma de RDF después de la transformación. Esto se evidencia claramente en la transformación del concepto de asociación en UML.

A continuación, se presentará cómo OMG ha llevado a cabo el mapeo de los conceptos de MOF a RDF. Como se detallará más adelante en este trabajo, se han utilizado reglas SHACL en lugar del uso del constructo *owl:Restriction*, debido a que SHACL puede considerarse como el equivalente del lenguaje OCL en UML.

Paquete:

El mapeo de un paquete desde MOF a RDF se representa en la Figura 7.4.

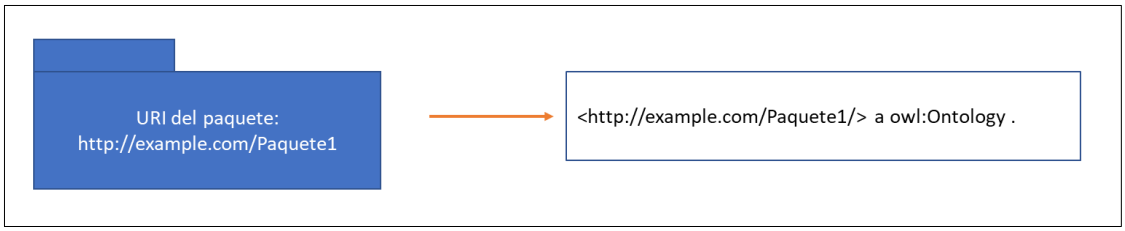


Figura 7.4: Mapeo de paquete a UML

Tipo de dato:

El mapeo de un tipo dato desde MOF a RDF se ilustra en la Figura 7.5.

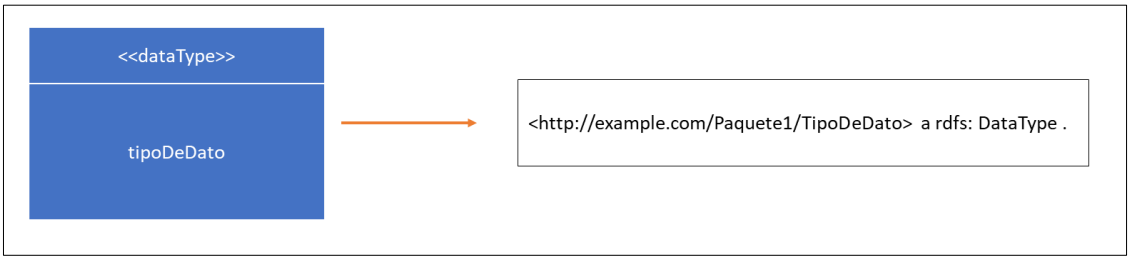


Figura 7.5: Mapeo de tipo de dato a RDF

Clase:

El mapeo de una Clase desde MOF a RDF se muestra en la Figura 7.6.

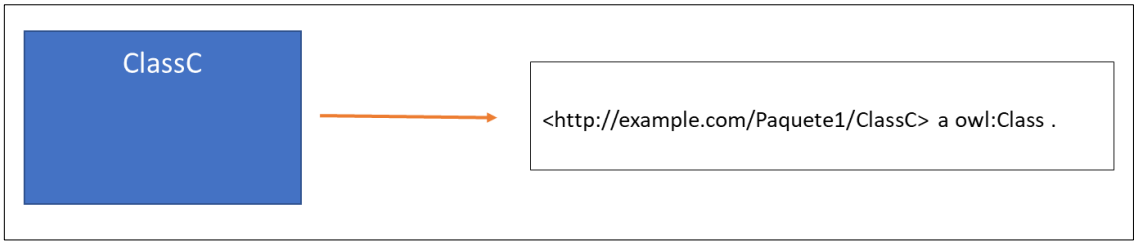


Figura 7.6: Mapeo de Clase a RDF

El mapeo del atributo desde MOF a RDF se presenta en la Figura 7.7 .

Atributo:

Figura 7.7: Mapeo de atributo a RDF.

El mapeo de una asociación desde MOF a RDF se muestra en la Figura 7.8.

Asociación:

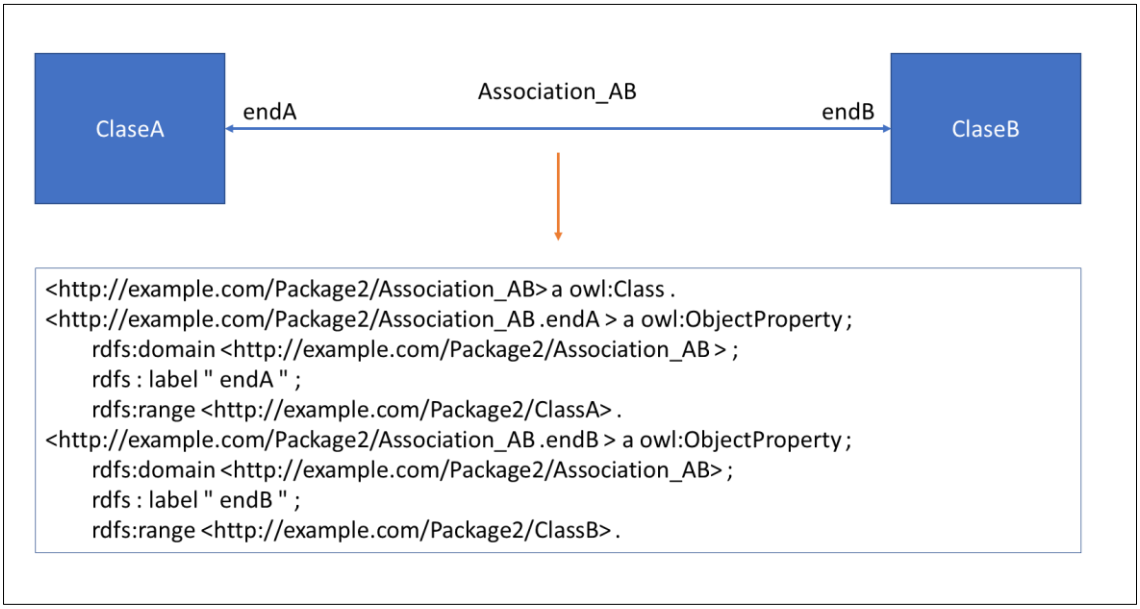


Figura 7.8: Mapeo de asociación a RDF

La propuesta desde este trabajo es el uso de la reificación aplicada a nivel de los ejemplares (ver Listado 7.1):

```
AtributoAs atrib1 valAtrib1 ;
            .....
            atribn valAtribn .


S_endA a rdf:Statement ;
        rdf:subject objetoB ;
        rdf:predicate endA ;
        rdf:object objetoA ;
        atributos AtributoAs .

S_endB a rdf:Statement ;
        rdf:subject objetoA ;
        rdf:predicate endB ;
        rdf:object objetoB .
```

Listado 7.1: Asociación a RDF con reificación.

7.2. Transformación de RDF/OWL a UML

En la actualidad, todas las comunidades que conforman la ingeniería de software han adoptado ampliamente la orientación a objetos en sus procesos de requisitos, análisis y desarrollo. Los lenguajes de propósito general utilizados para el desarrollo, como Java, C#, C++ e incluso lenguajes interpretados como Python, son

principalmente orientados a objetos. Esto implica que al manipular grafos RDF, se requiere un modelado en orientación a objetos de RDF como dominio. OMG ha estandarizado este modelo para lograr la portabilidad de los grafos RDF entre diferentes aplicaciones, lo que facilita la integración y la interoperabilidad.

Este proceso implica metamodelar un paradigma utilizando otro paradigma, en este caso, se trata de metamodelar el lenguaje RDF con el lenguaje UML.

Ontology Definition Metamodel ODM

En este trabajo, se va a enfocar exclusivamente en el metamodelado de RDF/RDFS, sin abordar, en detalle, la especificación de OMG relacionada con el un metamodelo para OWL.

En esta sección, se presentará el RDF Schema correspondiente al concepto, junto con el diagrama proporcionado por OMG para su mapeo a UML. Si bien OMG modela RDF en UML, no se hace mención explícita de los esquemas RDF de los conceptos ni del lenguaje visual asociado a los grafos utilizados en el modelado con RDF. El objetivo es analizar si el concepto y el metamodelo del concepto se pueden mapear adecuadamente.

OMG, para la transformación de RDF a UML y viceversa usa el lenguaje QVT [21] para dicha transformación. La crítica es que el metamodelo de RDF se ha realizado con el lenguaje UML y el metamodelo de UML se ha realizado en UML para poder llevar a cabo la transformación. En la Tabla 7.1 se muestra el mapeo en la dirección UML a OWL.

RDFS es el lenguaje para metamodelar. Los conceptos de RDFS están definidos con el lenguaje RDF en otras palabras RDF es su propio metalenguaje, de allí se han agrupado los conceptos RDF en el paquete *RDFConcepts* que es importado por el paquete *RDFS* como se muestra en la Figura 7.9. Tomada de la especificación de ODM de OMG.

UML CONCEPT	OWL CONCEPT
Clase UML	Clase OWL/RDF
Asociación UML	Propiedad RDF/OWL con su inversa
Atributo UML	Propiedad datatype OWL.
Subclase UML	rdfs:subClassOf
Restricciones de cardinalidad UML	Restricciones de cardinalidad OWL.

Tabla 7.1: Mapeo de conceptos en la dirección UML a OWL.

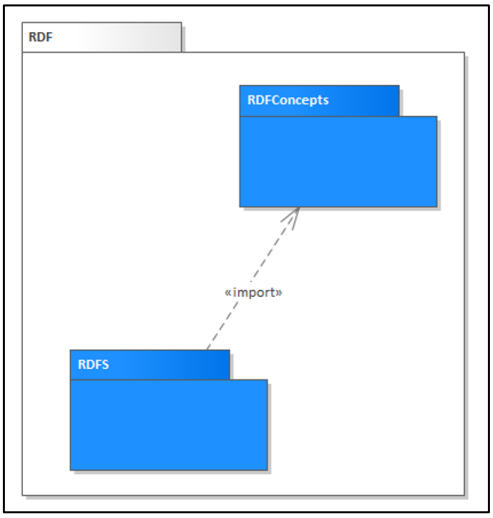


Figura 7.9: La estructura del metamodelo RDF (Fuente: figura 10.1 Ontology Definition Metamodel Version 1.1)

El diagrama de Figura 7.10 proporcionado por OMG especifica que un grafo está compuesto por un conjunto de blankNodes, que no es cierto por el hecho de que un grafo RDF es un conjunto de tripletas como viene definido en la especificación de W3C: "La estructura central de la sintaxis abstracta es un conjunto de tripletas, cada una de las cuales consta de un sujeto, un predicado y un objeto. Un conjunto de tales tripletas se llama un grafo RDF. Un grafo RDF se puede visualizar como un nodo y un diagrama de arcos dirigidos, en el que cada tripleta se representa como un enlace de nodo-arco-nodo.". Sería más coherente que la relación de composición que va de *Graph* a *BlankNode* se cambie de *Graph* a *Triple*. El experto de modelado en este caso debería tener conocimiento profundo tanto en el paradigma de orientación a objetos con UML como en el paradigma ontológico con RDF (el experto de modelado de UML necesita, para la especificación, un experto de dominio en RDF).

Se ha omitido el mapeo estructural del metamodelo en RDF que es RDFS con el metamodelo de UML para RDF.

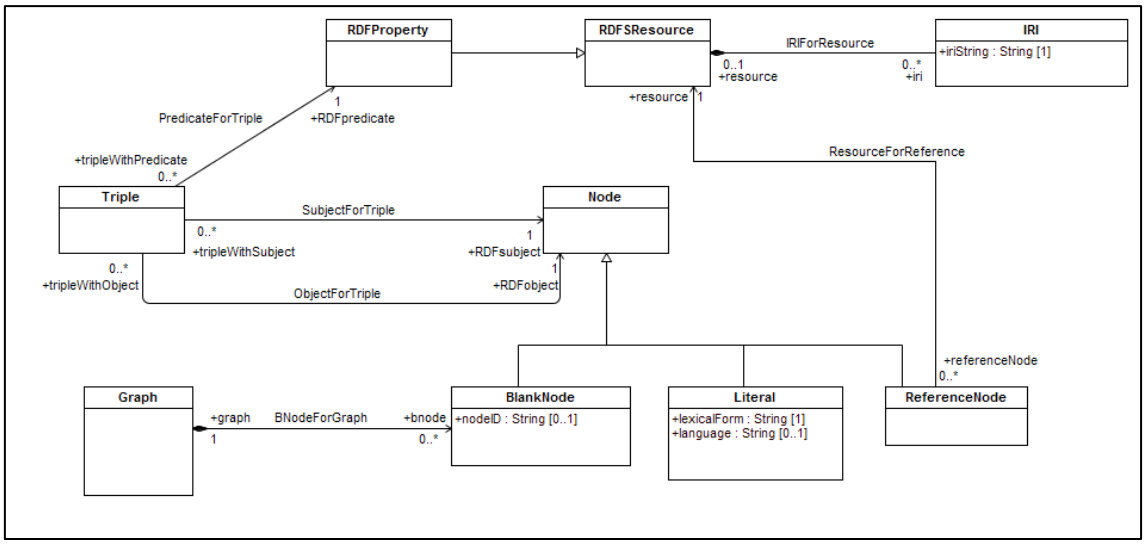


Figura 7.10: El modelo del grafo de datos (Fuente: Figura 10.2 Ontology definition model version 1.1)

El Listado 7.2 contiene el RDF Schema del concepto literal, su grafo RDF está representado en la Figura 7.11 y su correspondiente diagrama UML se muestra en la Figura 7.12.

```
rdfs:Resource rdf:type rdfs:Class .

rdfs:Datatype rdf:type rdfs:Class ;
        rdfs:subClassOf rdfs:Class .

rdfs:Literal rdf:type rdfs:Class .

rdfs:comment rdfs:domain rdfs:Resource ;
        rdfs:range rdfs:Literal .

rdfs:label rdfs:domain rdfs:Resource ;
        rdfs:range rdfs:Literal .
```

Listado 7.2: RDF de rdfs:Literal.

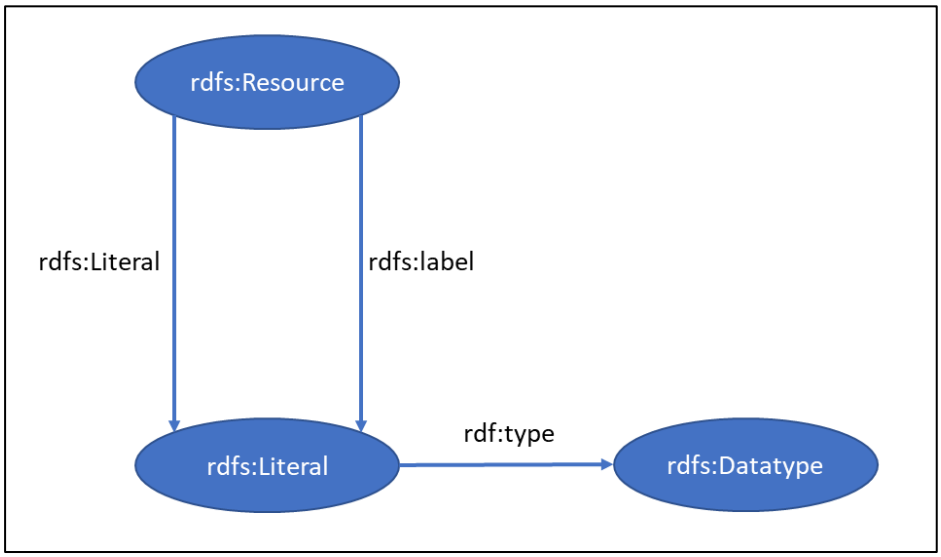


Figura 7.11: Grafo RDF para los literales

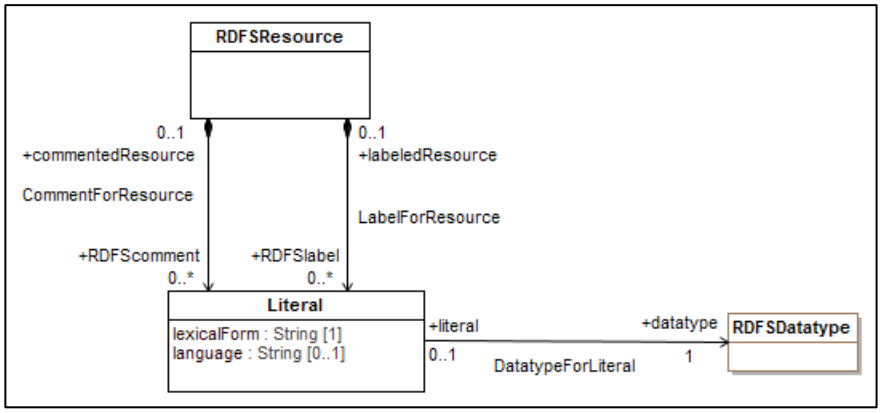


Figura 7.12: Diagrama de literales (Fuente: Figura 10.3 Ontology definition model version 1.1)

El Listado 7.3 contiene el RDF Schema del concepto de reificación, su grafo RDF está representado en la Figura 7.13 y su correspondiente diagrama UML se muestra en la Figura 7.14.

```
rdf:Statement rdf:type rdfs:Class .

rdf:subject rdfs:domain rdf:Statement ;
            rdfs:range rdfs:Resource.

rdf:predicate rdfs:domain rdf:Statement ;
              rdfs:range rdf:property.

rdf:object rdfs:domain rdf:Statement ;
           rdfs:range rdfs:Resource.
```

Listado 7.3: RDFS de reificación rdf:Statement.

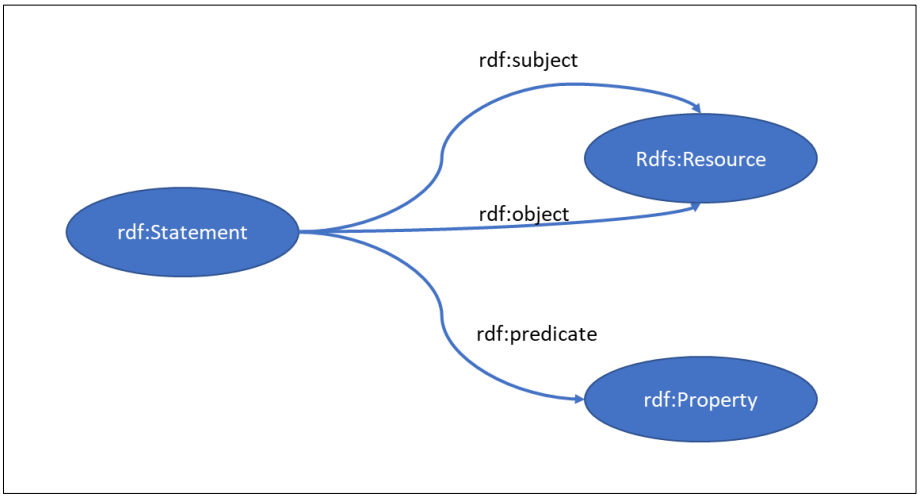


Figura 7.13: Reificación RDF

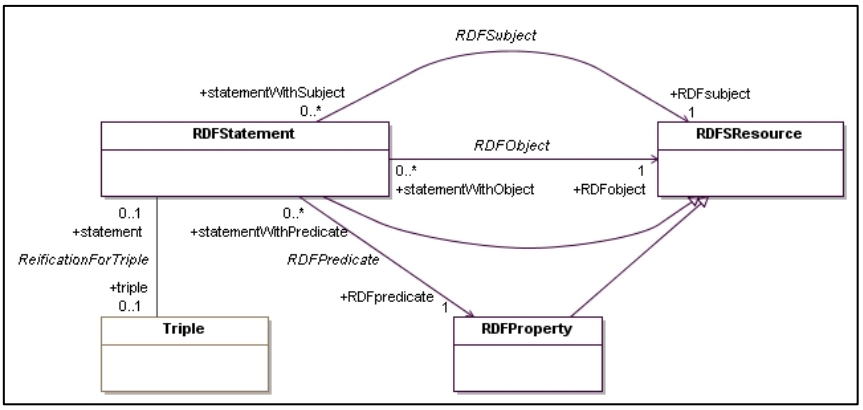


Figura 7.14: El diagrama de reificación (Fuente: Figura 10.4 Ontology definition model version 1.1)

El Listado 7.4 contiene el RDF Schema de los conceptos clase y recurso su grafo RDF está representado en la Figura 7.15 y su correspondiente diagrama UML se muestra en la Figura 7.16.

```
rdfs:Resource rdf:type rdfs:Class .
rdfs:Class rdfs:subClassOf rdfs:Resource .
rdfs:Class rdf:type rdfs:Class .
rdfs:Datatype rdf:type rdfs:Class .
rdfs:Datatype rdfs:subClassOf rdfs:Class .
rdfs:subClassOf rdfs:domain rdfs:Class .
rdfs:subClassOf rdfs:range rdfs:Class .
rdfs:seeAlso rdfs:domain rdfs:Resource .
rdfs:isDefinedBy rdfs:domain rdfs:Resource .
```

Listado 7.4: RDFS de clases, recursos y utilidades.

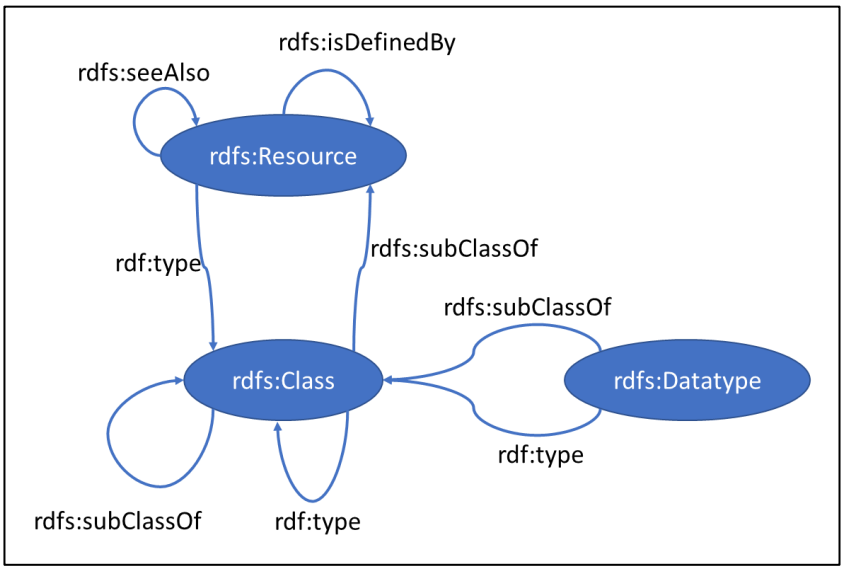


Figura 7.15: Grafo RDF para Resource, Class y DataType.

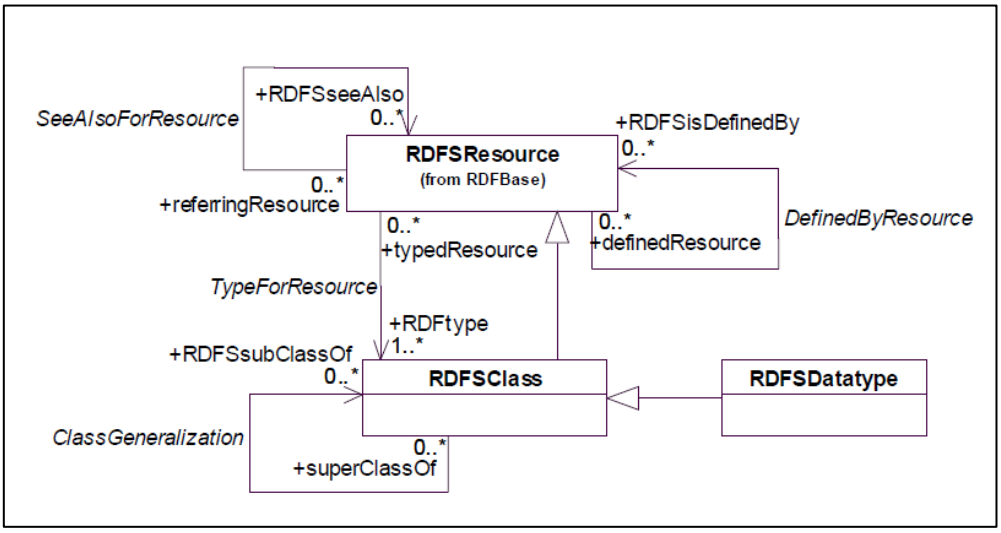


Figura 7.16: Diagrama de clases y utilidades (Fuente: Figura 10.5 Ontology definition model version 1.1)

El Listado 7.5 contiene el RDF Schema del concepto propiedad, su grafo RDF está representado en la Figura 7.17 y su correspondiente diagrama UML se muestra en la Figura 7.18.

```
rdf:Property rdf:type rdfs:Resource;
                rdfs:subclassOf rdfs:Resource;
             rdf:type rdfs:Class;
             rdfs:domain rdfs:Class;
             rdfs:range rdfs:Class.
rdfs:subProperty rdfs:domain rdf:Property ;
             rdfs:range rdf:Property .
```

Listado 7.5: RDFS de la propiedad.

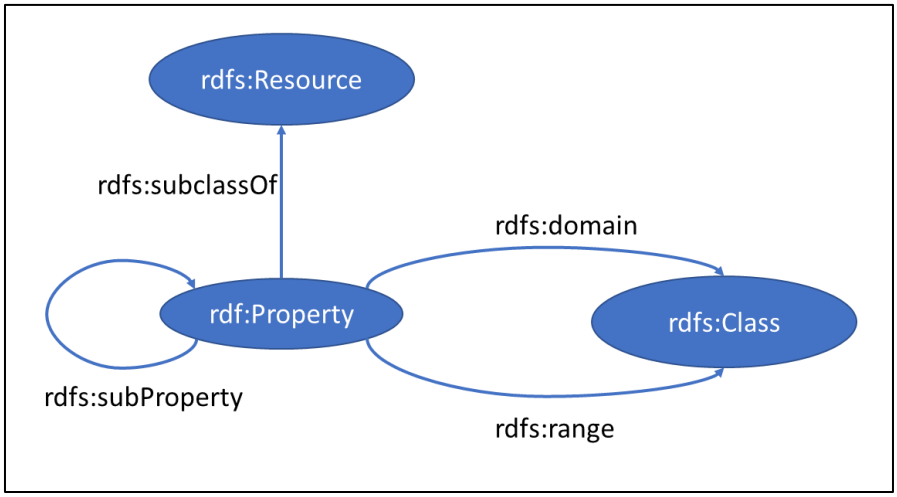


Figura 7.17: Grafo RDF para la propiedad

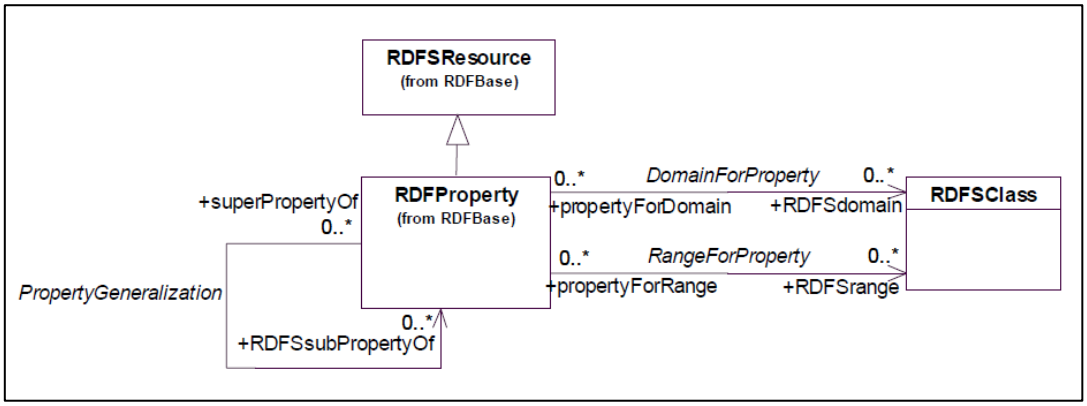


Figura 7.18: El diagrama de propiedades (Fuente: Figura 10.6 Ontology definition model version 1.1).

El Listado 7.6 contiene el RDF Schema de los conceptos contenedor y colección, su grafo RDF está representado en la Figura 7.19 y su correspondiente diagrama UML se muestra en la Figura 7.20.

```
rdfs:Container rdf:type rdfs:Class .
                    rdfs:subClassOf rdfs:Resource .
rdf:List rdfs:subClassOf rdfs:Container .
rdf:Alt rdfs:subClassOf rdfs:Container .
rdf:Bag rdfs:subClassOf rdfs:Container .
rdfs:ContainerMembershipProperty rdf:type rdfs:Class .

rdfs:member rdf:type rdf:Property ;
                rdfs:domain rdfs:Resource ;
```

```
                rdfs:range rdfs:Resource .
rdf:first rdfs:domain rdf:List .
          rdfs:range rdfs:Resource .
rdf:rest rdfs:domain rdf:list;
          rdfs:range rdf:List .
```

Listado 7.6: RDFS de contenedores y colecciones.

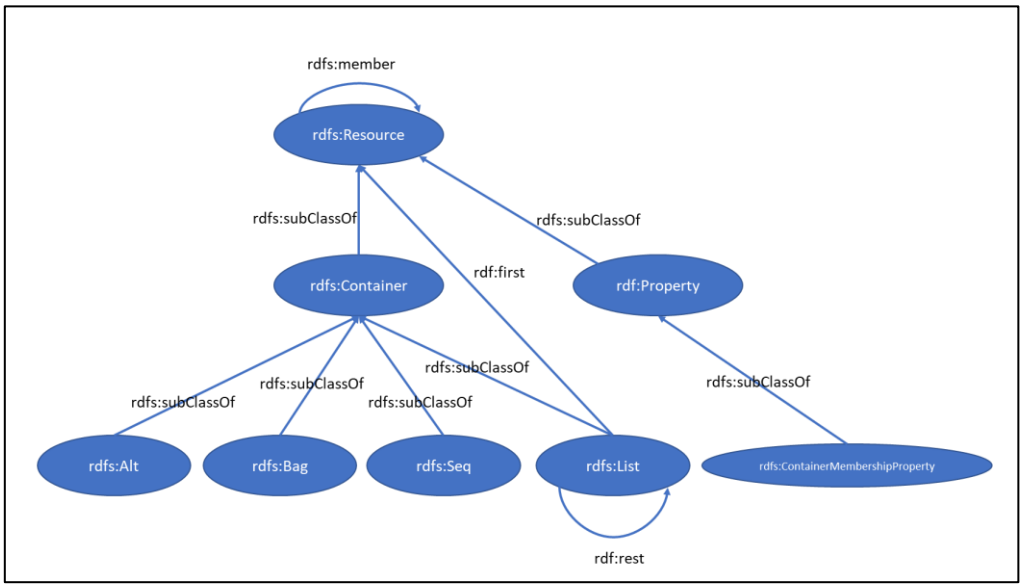


Figura 7.19: Grafo RDF para Contenedores y Colecciones.

El diagrama UML propuesto por OMG para el el grafo de la figura anterior se muestra en la figura siguiente.

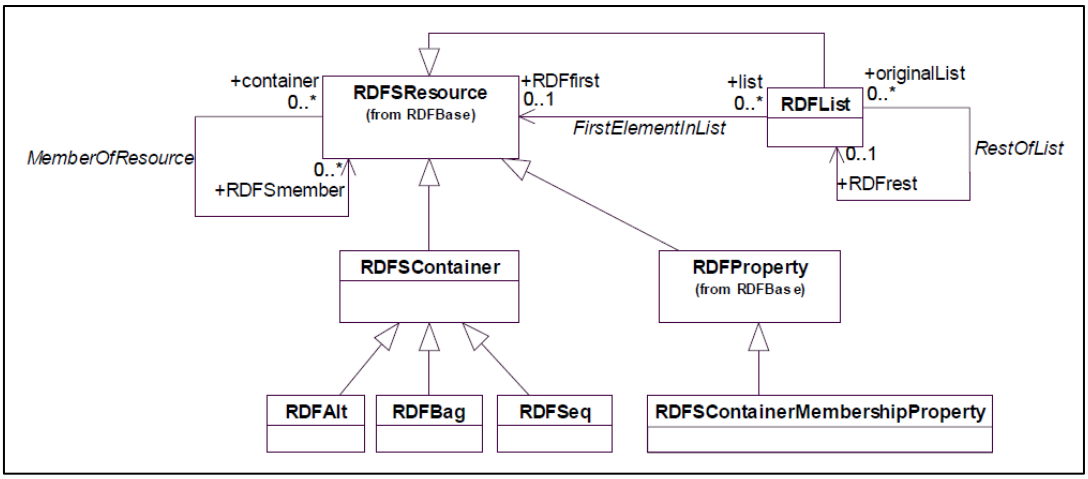


Figura 7.20: Diagrama de contenedores y colecciones (Fuente: Figura 10.7 Ontology definition model version 1.1).

8. Alineamiento de restricciones y transformación OCL a SHACL

8.1. Resumen

Los modelos son elementos de primera clase en la ingeniería dirigida por modelos (MDE). En este paradigma, los enfoques más extendidos y adoptados por la comunidad de desarrollo son el Orientado a Objetos y el ontológico, formalizados usando UML y RDF, respectivamente. Sin embargo, OMG no proporciona un lenguaje

estándar específico para validar modelos UML contra las restricciones definidas en OCL; mientras tanto, W3C ha definido SHACL como un lenguaje de validación estándar. Aunque la transformación entre UML y RDF se puede realizar a nivel estructural, no se ha hecho ningún esfuerzo para transformar OCL a SHACL. Este trabajo aborda la transformación de OCL y reglas basadas en texto a reglas SHACL en el contexto del estándar CGMES *(Common Grid Model Exchange Standard)*, un estándar basado en UML para el dominio de las compañías eléctricas en Europa. Este trabajo presenta varias contribuciones a la comunidad de ingeniería de software. En primer lugar, se resuelve el problema de validación de forma estandarizada. En segundo lugar, se facilita a la ENTSO-E *(Red Europea de Operadores de Sistemas de Transporte de Energía Eléctrica)* la construcción de una ontología asociada al estándar CGMES. En tercer lugar, se permite que los desarrolladores integren dos enfoques complementarios. Finalmente, se promueve la adopción e integración del enfoque ontológico en la comunidad de software.

8.2. **Introducción**

MDE es un paradigma en el cual los modelos son elementos de primera clase a lo largo del ciclo de vida del desarrollo de software. El enfoque Orientado a Objetos es una variante de MDE en la cual los elementos clave para modelar el mundo son la clase y el objeto. UML [22] es un lenguaje de modelado visual que se utiliza para abordar el proceso de modelado con enfoque Orientado a Objetos. OCL [23] define restricciones adicionales sobre los objetos en el modelo UML.

RDF es una tecnología de la web semántica que adopta un enfoque en el cual todo es considerado un recurso con un identificador único. Los modelos se construyen desde la perspectiva de recursos y predicados. En este enfoque, la representación atómica del conocimiento se realiza mediante una tripleta en forma de (sujeto, predicado, objeto), y los modelos se basan fundamentalmente en grafos.

El W3C proporciona SHACL como un lenguaje de validación estándar para validar restricciones o reglas sobre grafos de datos RDF. Además, también puede utilizarse como lenguaje de modelado.

En MDE, el modelado se puede abordar desde diferentes enfoques. Este trabajo se centra en la Orientación a Objetos (UML/OCL) y en el enfoque ontológico (RDF/SHACL), como se muestra en la Figura 8.1.

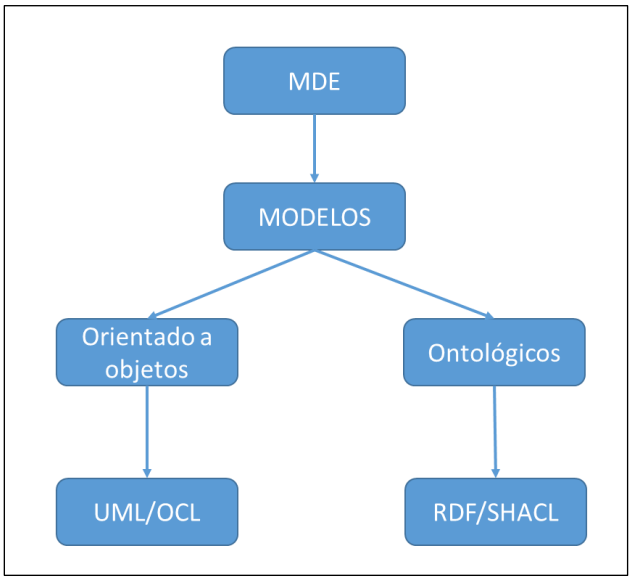


Figura 8.1: Los enfoques Orientado a Objectos y ontológico para MDE.

Independientemente del paradigma de modelado que se adopte, un modelo consta de un metamodelo, reglas e instancias. La transformación de un modelo en el paradigma fuente a otro modelo en el paradigma de destino implica la transformación de los aspectos léxicos, sintácticos y semánticos del paradigma origen al paradigma de destino, como se indica en la Figura 8.2.

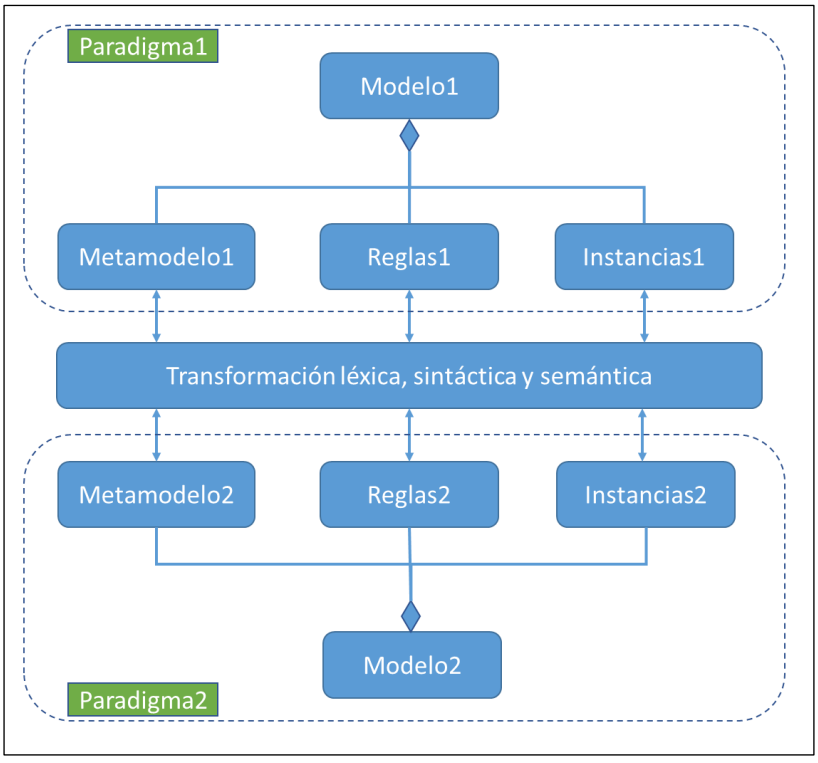


Figura 8.2: Transformación de modelos entre dos paradigmas.

Este trabajo se ocupa de llevar a cabo la transformación entre el paradigma Orientado a Objetos (UML) y el paradigma ontológico (RDF), tal como se muestra en la Figura 8.3.

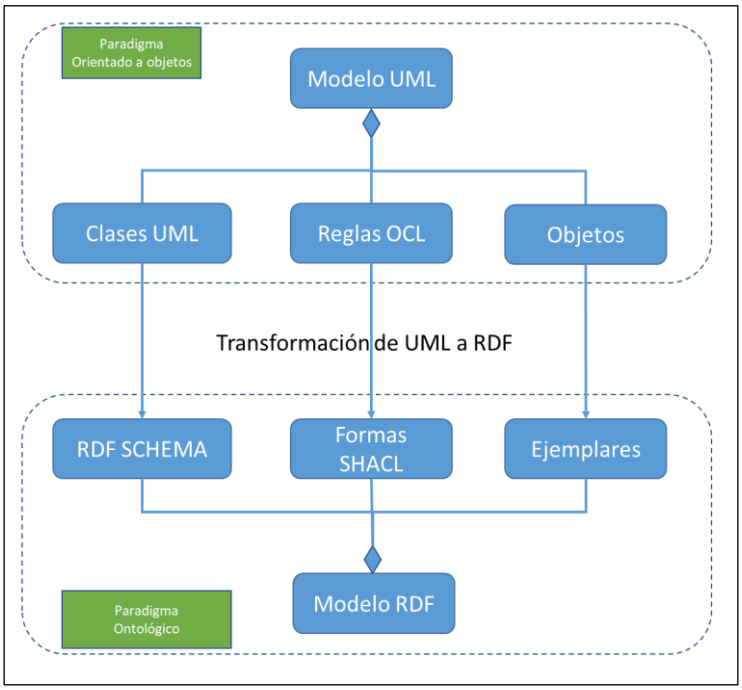


Figura 8.3: Transformación entre modelos UML (Orientación a objetos) y modelos RDF (Ontologicos).

El modelo de información común (CIM) es un estándar basado en UML adoptado por las compañías eléctricas para intercambiar modelos de red entre ellas. Consiste en un conjunto de normas de la Comisión Electrotécnica Internacional (IEC); Tabla 8.1 muestra un subconjunto adoptado en Europa por la ENTSO-E. En el contexto de ENTSO-E, CGMES es un estándar destinado a el intercambio de modelos de sistemas eléctricos entre los Operadores de Sistemas de Transmisión (TSO).

Estándar **IEC**	**Nombre del Estándar**
IEC 61970-552	Formato de intercambio de modelo XML CIM
IEC 61970-301	Modelo común de información (CIM) base
IEC 61970-302	Modelo de información común (CIM) para dinámicos
IEC 61970-452	Perfiles de modelo de red de transmisión estática CIM
IEC 61970-453	Perfil del diagrama de distribución
IEC 61970-456	Perfiles de estado en sistemas de distribución
IEC 61970-457	Perfiles dinámicos
IEC 61968-4	Integración de aplicaciones en compañías eléctricas

Tabla 8.1: Los estándares IEC adoptados por ENTSO-E.

CGMES es un estándar de interoperabilidad basado en el estándar UML/OCL y puede considerarse un enfoque dentro de MDE. Sin embargo, aunque OMG proporciona UML junto con OCL como lenguajes de modelado, carece de un estándar

para validar modelos UML contra restricciones OCL, debido a que UML presenta una limitación en cuanto a la definición de una semántica teórica de modelo formal. Esta limitación también se extiende a OCL, ya que UML carece de una teoría formal de modelo y prueba.

Como resultado, no es factible utilizar ni UML ni OCL para realizar razonamiento automatizado, debido a su falta de una teoría formal que respalde su utilización con ese propósito. Por lo tanto, la validación se lleva a cabo utilizando métodos no estandarizados, como se muestra en la Figura 8.4, lo cual tiene un impacto negativo en factores esenciales del software, como la calidad y productividad, entre otros.

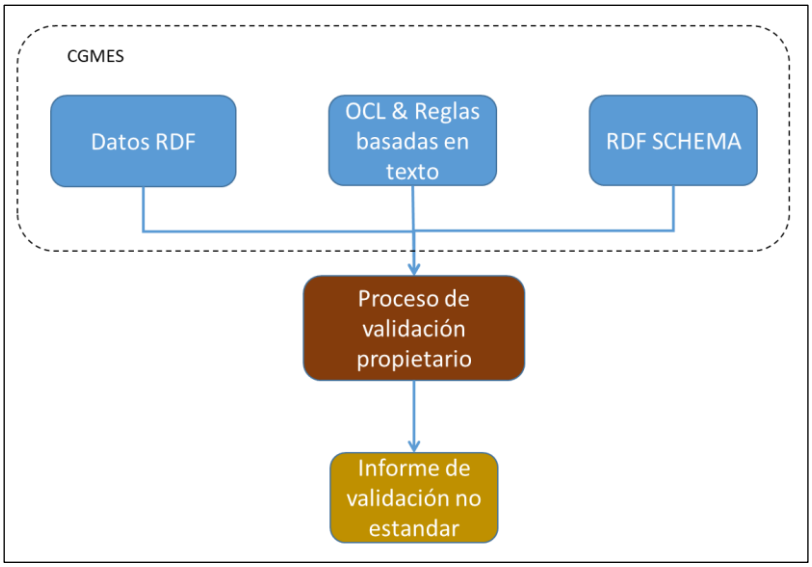


Figura 8.4: El enfoque propietario para el proceso de validación.

La solución propuesta en este trabajo es adoptar SHACL como un lenguaje para validar modelos contra las reglas en el contexto CGMES, ver Figura 8.5. El esfuerzo consiste en (i) la elaboración de una metodología para transformar las reglas OCL y basadas en texto a SHACL (ii) la conversión de todas las reglas OCL de CGMES a SHACL (iii) el desarrollo de una aplicación que permita validar cualquier modelo de TSO, miembro de ENTSO-E, contra las reglas del CGMES.

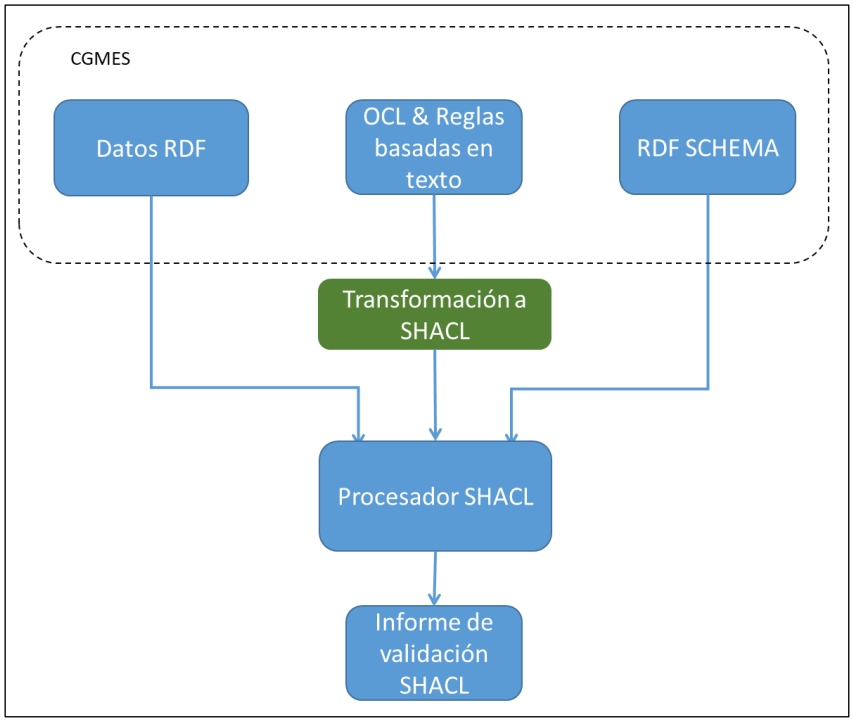


Figura 8.5: La validación con SHACL en el contexto de CGMES.

Este trabajo, basado en la publicación [24], presenta una serie de contribuciones: (i) Una metodología para convertir OCL a SHACL, un enfoque que nunca ha sido abordado anteriormente. (ii) Proporcionar a las partes interesadas de ENTSO-E un modelo ontológico RDF/SHACL para validar sus modelos y guiarlos en el desarrollo de sus propias reglas en SHACL, ya sea mediante la conversión de los modelos existentes o la creación de nuevos modelos. (iii) Desarrollo de una aplicación web para validar el cumplimiento de un modelo específico con las reglas CGMES. (iv) Con esta contribución, se busca cerrar la brecha entre el paradigma orientado a objetos y el paradigma ontológico, en línea con los esfuerzos actuales de diferentes comunidades para promover la adopción de la ingeniería ontológica a través de tecnologías de la web semántica como [25]. La audiencia objetivo de este trabajo son los expertos en el dominio de los sistemas de potencia y compañías eléctricas y los desarrolladores de software involucrados en el contexto de la validación de modelos de red.

El trabajo se divide en nueve partes: En la sección 9.3 se presenta una revisión del estado del arte. En la sección 9.4 se describen los documentos del estándar CGMES. La sección 9.5 explora las razones para adoptar SHACL como un lenguaje de validación. La sección 9.6 muestra una regla básica de modelado en SHACL. La sección 9.7 describe la metodología para transformar las reglas OCL y CGMES a formas SHACL. En la sección 9.8 se describe la transformación de OCL a SHACL. La sección 9.9 presenta una aplicación web para validar modelos CGMES. Finalmente, la sección 9.10 contiene las conclusiones y los trabajos futuros.

8.3. **Revisión del estado del arte**

En las últimas décadas, los dos enfoques más prometedores para el modelado son el orientado a objetos y el ontológico. La comunidad de software ha adoptado UML/OCL como enfoque para el modelado orientado a objetos. Por otro lado, las tecnologías de la web semántica como RDF/OWL/SHACL [26] se han adoptado para el modelado ontológico.

Esta sección se divide principalmente en dos partes: (i) una presentación de publicaciones relacionadas con el modelado orientado a objetos y (ii) otra que incluye trabajos relacionados con el modelado ontológico.

A continuación, se enumeran las publicaciones relacionadas con el enfoque UML/OCL: En [27], se presentan estrategias de automatización de reglas OCL. Este trabajo proporciona una metodología semiautomática para la evolución de las reglas OCL en respuesta a cambios en los diagramas de clase. En [28], se desarrolla una herramienta basada en la web con un lenguaje de nivel de abstracción más alto para definir reglas sin necesidad de conocimientos profundos de la sintaxis y semántica de OCL. En [29], se utiliza la lógica de primer orden para transformar UML/OCL y validar modelos UML. En [30], se crea un modelo lógico en torno a UML/OCL que tiene en consideración las precondiciones, las postcondiciones, los estados y las transiciones para validar y verificar modelos UML, teniendo en cuenta todos los posibles estados y transiciones. En [31], Se propone una conversión de modelos UML/OCL a programación lógica con restricciones, con el objetivo de facilitar la validación, el razonamiento y la verificación, poniendo especial atención en los invariantes OCL. En [32], se crea un *framework* con un repositorio de modelos y un modelo de datos común para abstraer e integrar fuentes heterogéneas, como XML y ontologías, y visualizarlos de manera homogénea.

En cuanto al enfoque ontológico, en el estado del arte se han encontrado metodologías para validar reglas.

A continuación, se enumeran las publicaciones relacionadas con el enfoque ontológico: En [33], en lugar de validar datos RDF contra un esquema, un modelo se valida con datos RDF. El propósito es validar la credibilidad del modelo. La validación se realiza utilizando heurísticas de puntuación de axiomas basadas en la teoría de la posibilidad. En [34] se utiliza el SWRL para el intercambio de datos aplicado al dominio del agua usando WDTF (*Water Data Transfer Format*) para la definición de las restricciones de integridad y se ha usado el lenguaje OWL para modelar este dominio. A pesar de que OWL y SWRL consideran la suposición de mundo abierto (OWA), los autores han utilizado OWL/SWRL para la validación de modelos. Se realiza la validación transformando axiomas en consultas sobre el dominio lógico y transformando estas consultas en SPARQL, que son ejecutables por la máquina. Sin embargo, se considera en el contexto de este trabajo que SHACL es adecuado en el contexto de validación. Como se puede observar en el estado del arte, existen varios enfoques de validación no estandarizados [33], [34] son trabajos llevados a cabo en el contexto de la web semántica. En 2017 el lenguaje SHACL ha sido publicado como un estándar de validación RDF en el dominio de modelado de ontologías. Hasta dónde llega conocimiento obtenido desde la investigación en esta tesis, no se ha encontrado ningún trabajo previo que aborde la transformación de OCL a SHACL.

En [35], se describen varios enfoques de validación de RDF. Uno de ellos es SHACL, el primer lenguaje de validación estándar para grafos RDF. En [36], se pone a disposición

una herramienta de validación en línea para SHACL. En [37], Los autores han transformado el modelo estándar XAPI *(Experience API)* a una ontología, y han utilizado SHACL para validar las restricciones definidas en la XAPI estándar. Se ha utilizado JSON para serializar datos. En [38], los autores han transformado la Iniciativa de Modelado de Información Clínica (CIMI) a ontologías, usando SHACL para modelar y definir reglas con fines de validación. En [39] se crea una ontología colaborativa a partir de flujos de información provenientes de diferentes fuentes, tanto públicas como privadas, en el dominio de sensores ambientales. Se realiza un mapeo semántico, junto con un enriquecimiento datos. Los datos se validan mediante SHACL. Para ello, los autores han desarrollado un *framework* llamado LSane. En [40], la evaluación de la calidad de un par de bases de conocimiento (KB) se realiza en función de la completitud y la consistencia mediante el uso de un *framework* basado en la web semántica. Esta evaluación se ha realizado mediante consultas SHACL y SPARQL. Dado que estos bases de conocimiento están basados en RDF y son públicos, se ha realizado un análisis estadístico. La principal contribución de este trabajo es la evaluación de calidad de bases de conocimiento sujetas a constante evolución. En [41], se ha realizado el esfuerzo donde se ha usado SHACL para modelar y validar Los modelos CGMES, pero limitado a la validación del esquema RDF sin abordar las reglas OCL ni las reglas CGMES basada en texto. [42] es un libro que explora cómo las tecnologías de la web semántica pueden ser utilizadas para crear aplicaciones inteligentes, y a su vez, establece un puente entre la comunidad industrial y la comunidad de la web semántica.

El dominio de aplicación de este trabajo es el intercambio de modelos de red entre las compañías eléctricas. Como se muestra en los párrafos anteriores, hay una necesidad desde diferentes dominios de la transformación de los modelos de dominio a un enfoque ontológico con RDF/OWL y SHACL. Este trabajo se centra en la tarea más tediosa para la transformación entre diferentes enfoques: transformar las reglas OCL a las formas SHACL. Para transformar UML a RDF o OWL, OMG especifica esta transformación en el metamodelo de definición de ontología (ODM) y en el mapeo de MOF a RDF, que es una nueva especificación beta. Sin embargo, no se ha encontrado ningún trabajo. hasta ahora sobre la transformación entre las reglas OCL y SHACL. En este trabajo se presenta una metodología para: (i) la conversión de restricciones OCL a formas SHACL teniendo en cuenta los elementos léxicos, sintácticos y semánticos claves en ambas gramáticas, (ii) la transformación de las reglas basadas en texto a SHACL ha sido abordada, tanto de forma manual como semiautomática, dada la ambigüedad del lenguaje natural. Además, utilizando esta metodología, se han modelado e implementado todas las reglas CGMES, incluyendo aquellas basadas en OCL y en texto. Adicionalmente, se ha desarrollado una aplicación basada en microservicios para la validación de los modelos de red.

8.4. **Documentos del estándar CGMES**

``Especificación de intercambio de modelo de red común (CGMES) es una especificación técnica (TS) de IEC basada en una familia de normas IEC CIM. Fue desarrollado para cumplir con los requisitos para los intercambios de datos entre TSOs

en las áreas de desarrollo del sistema y operación del sistema'' [43]. ENTSO-E CGMES proporciona un conjunto de documentos para la especificación del estándar. Los documentos clave que son de interés para este trabajo se muestran en la Tabla 8.2. QoDCRules.xsd es un archivo XSD para definir el modelo de reglas, véase la Figura 8.6.

Documento	Propósito
Quality_of_CGMES_Datasets_and_Calculations.pdf	Descripción de las reglas
QoDCRules.xsd	Modelo de reglas
QoDCRules level1.xml a level7.xml	Estructura del informe de validación
Ficheros RDFS	RDFS por cada perfil
Ficheros OCL	OCL por cada perfil
Documentos de los estándares IEC para la serialización	Ver Tabla 9-1

Tabla 8.2: Documentos CGMES elementales.

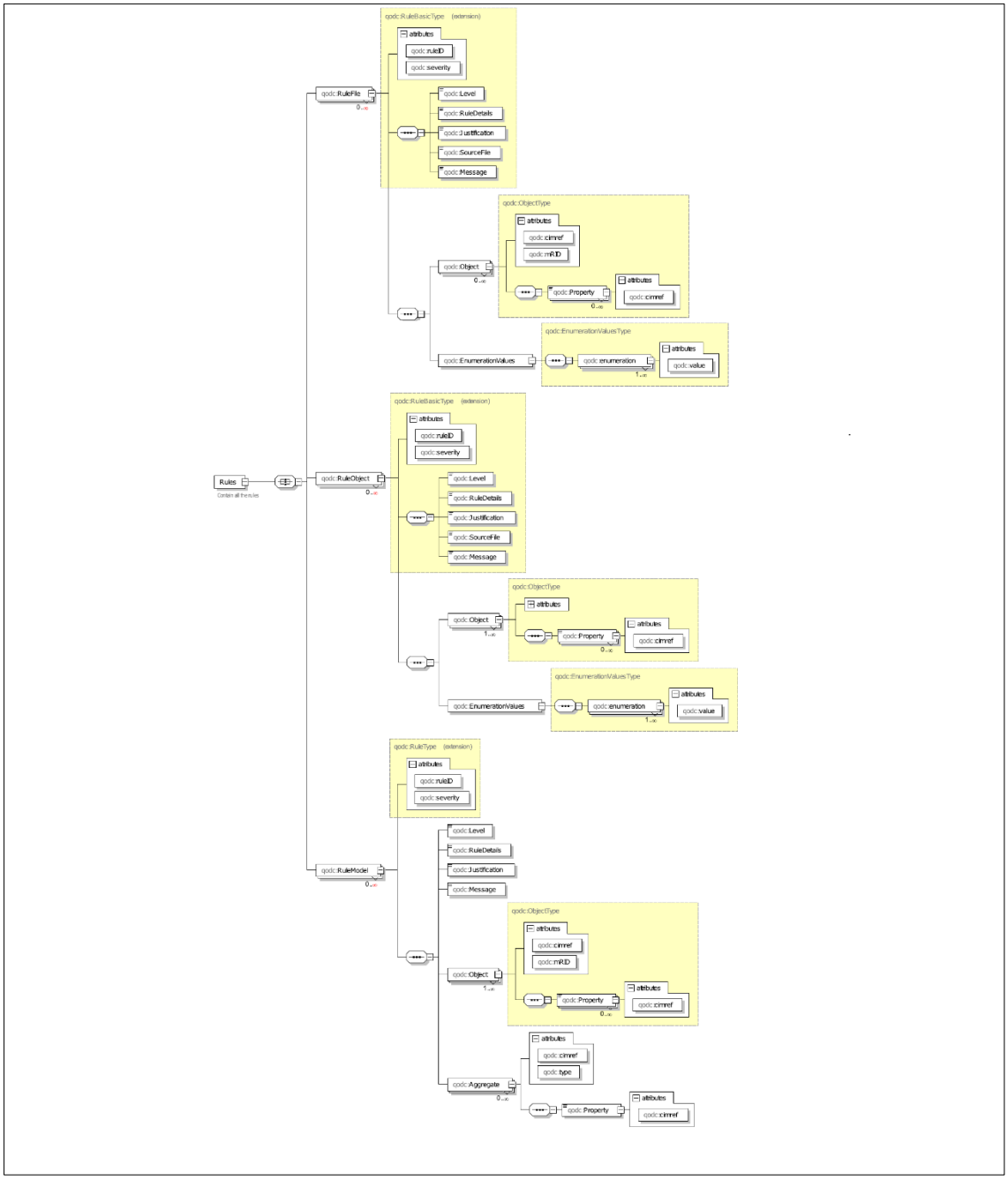


Figura 8.6: Modelo de reglas CGMES.

Los documentos y archivos proporcionados por el estándar CIM CGMES se han clasificado de la siguiente forma.

- QoDCRules level1.xml a level7.xml definen cómo los informes de validación deben producirse para los siete niveles.
- Archivos RDFS y OCL para los modelos correspondientes a EQ (Equipment), DL (Diagram layout), DY (Dynamics), BD-EQ (Boundary equipment), SV (State Variables), SSH (Steady state hypothesis), TP (Topology), BD-TP (Boundary TP) entre otros.
- Dos estándares IEC utilizados por CGMES para la serialización de UML a RDF que son: (i) IEC 61970-552 [44] define la estructura del documento, el

encabezado, la sintaxis y los metadatos. (ii) IEC 61970-501 [45] que define el mapeo de conceptos entre CIM (UML) y RDF.

8.5. **Por qué SHACL**

En el contexto del modelado, especialmente en la validación de modelos para asegurar la calidad y la interoperabilidad durante el intercambio de modelos, se reconoce la necesidad de contar con un lenguaje de validación estándar. [46]. W3C proporciona SHACL para permitir la Validación del modelo contra las reglas. Sin embargo, ningún lenguaje estándar es proporcionado para la validación de los modelos definidos en UML (Orientado a Objetos). Un aspecto clave de MDE es la transformación de modelos. Por lo tanto, en el presente caso se presenta la necesidad de transformar modelos UML/OCL a RDF/SHACL para llevar a cabo la validación con las tecnologías de la web semántica. Un caso particular es el estándar CIM, cuyos fundamentos se basan en técnicas Orientadas a Objetos (UML) [47]. Por lo tanto, los modelos son diagramas de clases y objetos, y con el propósito de serialización de modelos, el estándar CIM ha adoptado el lenguaje RDF del W3C. Es necesario señalar una característica de suma relevancia: la ``ejecutabilidad'' del modelo, que se puede definir de la siguiente manera:

Un modelo M formalizado en un lenguaje L bajo un paradigma P es ejecutable si existe un procesador que permita llevar a cabo tareas de razonamiento sobre el modelo M para un objetivo específico sin ningún modelo de transformación. (por ejemplo, P = Orientación a objetos, P = Ontológico, L = RDF; L = UML).

Dado que UML es un lenguaje visual, los modelos formalizados con él no son directamente ejecutables; por lo tanto, es necesaria una transformación a otro lenguaje que permita la ejecutabilidad. Sin embargo, los modelos definidos en RDF son directamente ejecutables en el sentido de que suportan razonamiento (ver Figura 8.7).

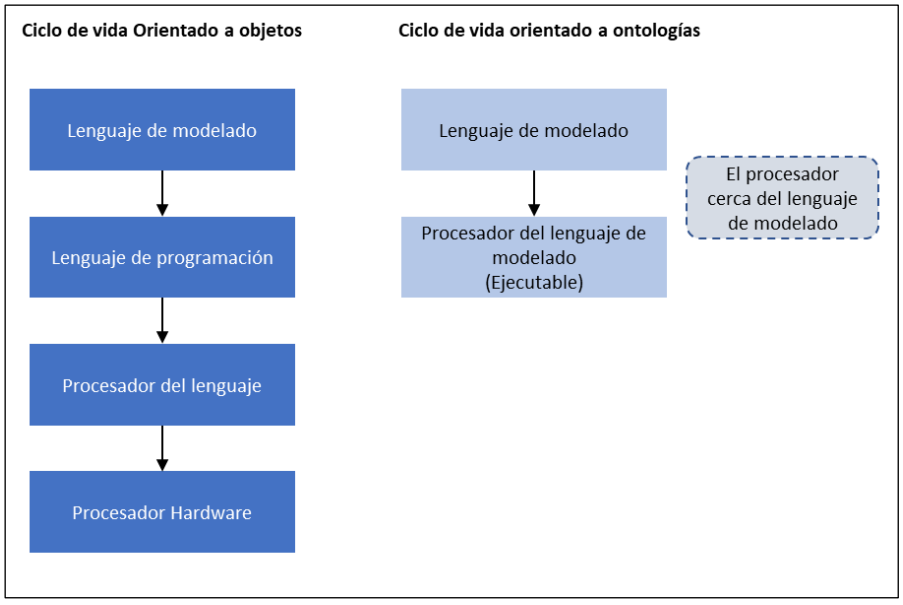


Figura 8.7: Ciclo de vida Orientado a Objetos VS Ciclo de vida Ontológico.

ENTSO-E ha creado perfiles CIM para especificar el estándar CGMES. Estos perfiles son principalmente diagramas de clases, junto con reglas adicionales escritas en OCL o basadas en texto. Con esta especificación, queda claro que el modelo no es directamente ejecutable. Una tarea crucial como la validación en este contexto se puede realizar a través de: (i) La transformación del modelo a otro paradigma que permite la ejecutabilidad del modelo. (ii) Usar un lenguaje de propósito general, pero las reglas basadas en texto y OCL requieren un esfuerzo considerable para ser implementadas, lo que negativamente afecta la calidad del software y a la productividad. (iii) Uso de EMF *(Eclispe modeling framework)* con OCL, pero el informe de validación devuelto carece de información elemental ya que contiene sólo una referencia al objeto y al invariante; dado que OCL, en esencia, no es un lenguaje de validación.

W3C, en respuesta a la necesidad de la comunidad de un lenguaje estándar para la validación, ha proporcionado SHACL para este objetivo. Los argumentos para adoptar SHACL son: (i) Incorporar SHACL como lenguaje de validación estándar para el estándar CGMES para resolver el problema de validación. (ii) La creación de una ontología como parte de CGMES, representando la parte estructural en RDFS y las reglas en SHACL. Por lo tanto, se proporciona un conocimiento reutilizable y compartido (razonamiento, procesamiento, consultas) que es legible tanto por la máquina como por los humanos, a través de las tecnologías de web semántica. (iii) Los modelos RDF/SHACL son directamente ejecutables, lo que se traduce en un considerable ahorro de tiempo y esfuerzo en la codificación en un lenguaje de programación de propósito general, y en una mejora de la calidad del proceso y del producto, ya que este enfoque está más alineado con MDE [48] [49] [50] que UML/OCL. Además, en este trabajo se han realizado dos comparaciones referentes a: (i) La presencia de los enfoques de la Orientado a Objetos y ontológico en aspectos claves de modelado en el estándar CGMES, (ver Tabla 8.3). (ii) Características que ofrecen los lenguajes OCL y SHACL, que se muestran en la Tabla 8.4.

Aspectos de modelado CGMES	Paradigma Ontológico	Paradigma Orientado a Objetos
Modelado estructural	No forma parte del estándar CGMES	UML
Serialización	RDF proporciona un identificador único.	• XMI Solo para diagrama de clases • Serialización de diagramas de objetos con relaciones entre objetos no contemplada en (UML/XMI) • XML no permite enlaces entre dos elementos que nos sean padre e hijo.
Validación	SHACL/SPARQL (No adoptado todavía)	Las reglas OCL son parte del estándar CGMES.
Definición de reglas	Las reglas no están definidas con SHACL como parte del estándar CGMES.	OCL y reglas basadas en texto.
Suporte de razonamiento	Si.	No.

Aspectos de modelado CGMES	Paradigma Ontológico	Paradigma Orientado a Objetos
Meta modelado	RDF Schema o OWL (pero de forma no voluntaria, producto de la necesidad de serializar)	Diagrama de clases.
Ejemplares	RDF	Diagrama de objetos.
Estándares de validación	SHACL (pero no está adoptado todavía como lenguaje para la validacion)	UML/OCL es un estándar para modelar, pero no para validar.

Tabla 8.3: La implicación de los paradigmas orientación a objetos y ontológico en diferentes aspectos del modelado en CGMES.

Característica	OCL	SHACL
Razonamiento	No soportado	Soportado
Informe de validación	Solo proporciona invariante y objeto	Proporciona el ID de la regla junto con el ID del objeto y sh:message.
Severidad	No esta soportado	sh:warning sh:info sh:violation
Dependencias de restricciones	No están soportadas	Dependencia entre reglas soportada
Restricciones inter-modelo	No soportado	Soportado
Flexibilidad en la definición del contexto	Limitado a la clase	Targets basados en SPARQL “sh:target”
URI/ID	No forma parte del paradigma Orientado a objetos	Es parte del paradigma RDF

Tabla 8.4: SHACL VS. OCL.

8.6. **Modelado básico de una regla en SHACL y constructos básicos**

Esta sección pretende ilustrar los elementos esenciales de SHACL desde los puntos de vista arquitectónico, sintáctico y semántico. En el mundo RDF, todo es un recurso. En RDF, el modelo más elemental es una tripleta (sujeto, predicado, objeto) que corresponde a un grafo, (ver Figura 8.8).

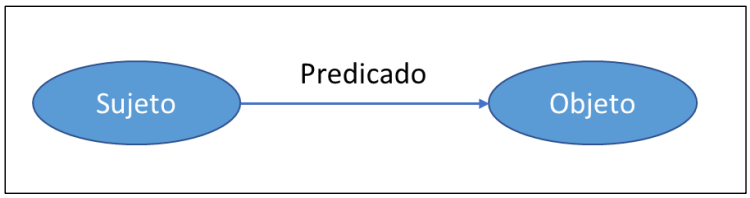


Figura 8.8: El grafo RDF correspondiente a la tripleta (sujeto, predicado, objeto).

SHACL es un lenguaje para validar modelos de datos contra restricciones. Cabe señalar que tanto los datos como las reglas están representados en el lenguaje RDF, y, por tanto, los dos son grafos. El procesador SHACL tiene dos entradas, datos RDF y un conjunto de reglas. Una vez ejecutado el proceso de validación se devuelve un informe de validación que a su vez está definido en RDF. El informe contiene los nodos del grafo de datos donde las reglas han sido violadas, (ver Figura 8.9).

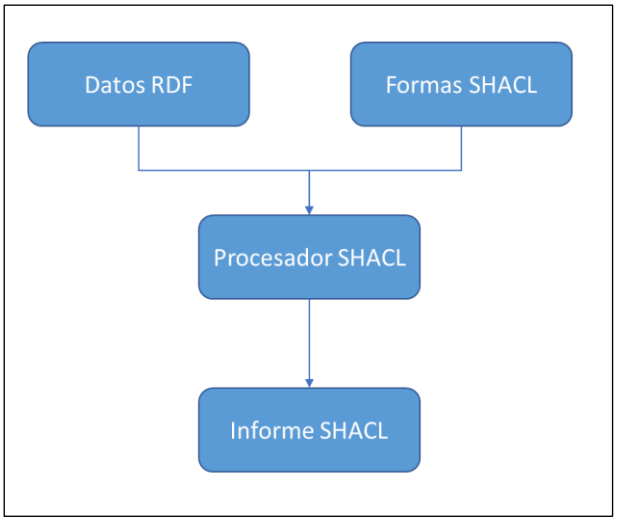


Figura 8.9: Arquitectura del procesador SHACL.

El procesador SHACL recorre el grafo, visitando los nodos foco especificados en la regla para evaluar ésta sobre ellos; en caso de incumplimiento, la información sobre los identificadores de los elementos donde se violan las reglas se añade al informe de validación junto con información adicional como un mensaje de retroalimentación indicado en la regla, (ver Figura 8.10).

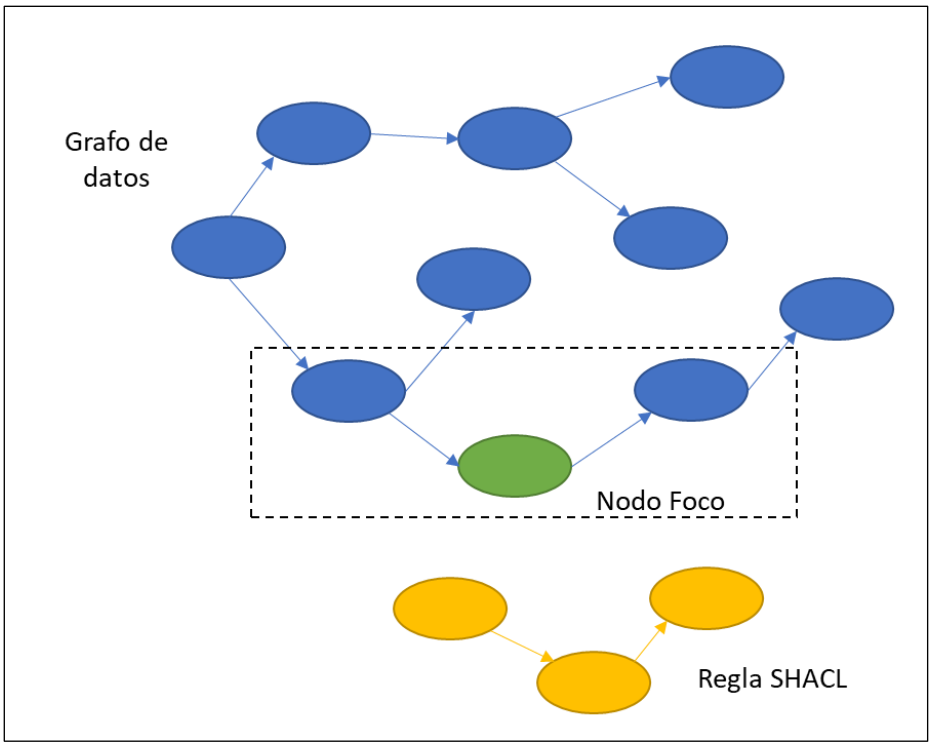


Figura 8.10: Recorrido del procesador de SHACL para la validación de reglas.

W3C ha dividido la especificación SHACL en SHACL-CORE, que contiene los constructos básicos, y SHACL-SPARQL, que proporciona constructos para una

expresividad más avanzada, basada en SPARQL. El siguiente ejemplo ilustra una regla simple, su implementación con SHACL-CORE, y el resultado del informe de validación. Los datos RDF se describen en el Listado 8.1.

```
:Peter a :Person;
        :idCode "123456789A" ;
        :idCode "123456789B" .
```

Listado 8.1: Ejemplo de RDF de datos.

Donde el objeto *:Peter* es de tipo *:Persona*, se declara *(:Peter a :Person;)* con dos códigos de identificación *"123456789A"* y *"123456789B".* El grafo correspondiente al anterior RDF se muestra en la Figura 8.11.

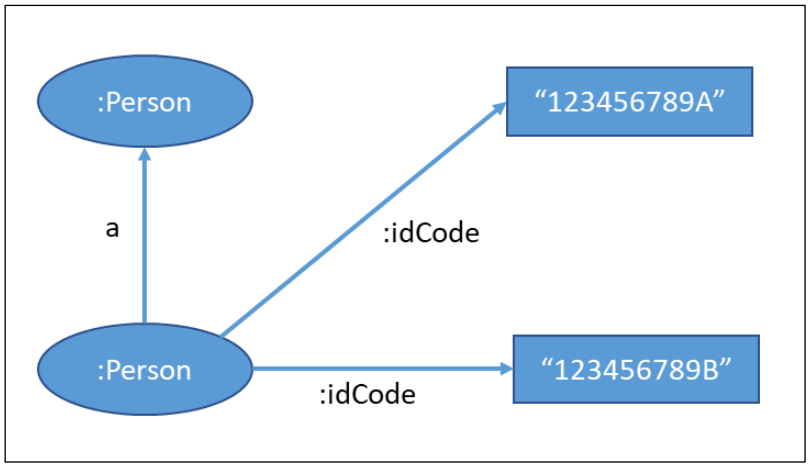


Figura 8.11: El grafo correspondiente al código RDF del listado 9.1.

En este ejemplo de regla, todas las instancias del tipo *:Persona* deben tener un valor único para *:idCode*. El código SHACL que define la regla se muestra en el Listado 8.2.

```
# El Identificador de la regla es :uniqueCodePerson
: uniqueCodePerson a sh:PropertyShape;

                # La clase del nodo foco es :Person
                sh:targetClass :Person;

                # Mensaje de no conformidad.
                sh:message "Solo un valor para :idCode está permitido";

                # La restricción se aplica al predicado :idCode
                sh:path :idCode;

                # El número máximo de valores para :idCode tiene que ser 1
```

```
        sh:maxCount 1;

        # El mínimo número de valores para: idCode tiene que ser 1
        sh:minCount 1.
```

Listado 8.2: La regla SHACL para la unicidad de idCode.

El Listado 8.3 muestra el informe de validación obtenido una vez que el procesador SHACL se ha ejecutado con los datos de entrada correspondientes al Listado 8.1 (datos RDF) y al Listado 8.2 (formas SHACL). En el informe de validación de RDF representado en el Listado 8.3, el RDF de los datos incumple la regla porque hay dos *idCode* para la misma persona. El informe muestra las coordenadas del incumplimiento en términos de los identificadores del nodo y de la regla donde se ha producido la violación junto a información adicional, como la severidad y un mensaje que contiene una descripción textual de los motivos del incumplimiento.

```
[ a sh:ValidationReport ;
sh:conforms false ;
sh:result       [ a      sh:ValidationResult ;
        sh:focusNode  :Peter ;
        sh:resultMessage      "Solo un valor para :idCode está permitido" ;
        sh:resultPath  :idCode ;
        sh:resultSeverity       sh:Violation ;
        sh:sourceConstraintComponent      sh:MaxCountConstraintComponent ;
        sh:sourceShape :uniqueCodePerson
        ]
] .
```

Listado 8.3: Informe de validación.

8.7. **Metodología de transformación de OCL y reglas CGMES a SHACL**

Los paradigmas orientación a objetos y ontológico están formalizados por UML/OCL y RDF/SHACL respectivamente. La transformación de UML/OCL a RDF/SHACL que se muestra en Figura 8.12 implica los siguientes pasos:

1. Los diagramas de clase se transforman en RDF/OWL asignando conceptos UML a conceptos RDF.
2. Las restricciones OCL se transforman en su correspondiente forma SHACL.
3. Las reglas basadas en texto se definen como reglas SHACL.
4. Los objetos UML se convierten en datos RDF.

El esfuerzo en este sentido consiste en transformar OCL y reglas basadas en texto a SHACL.

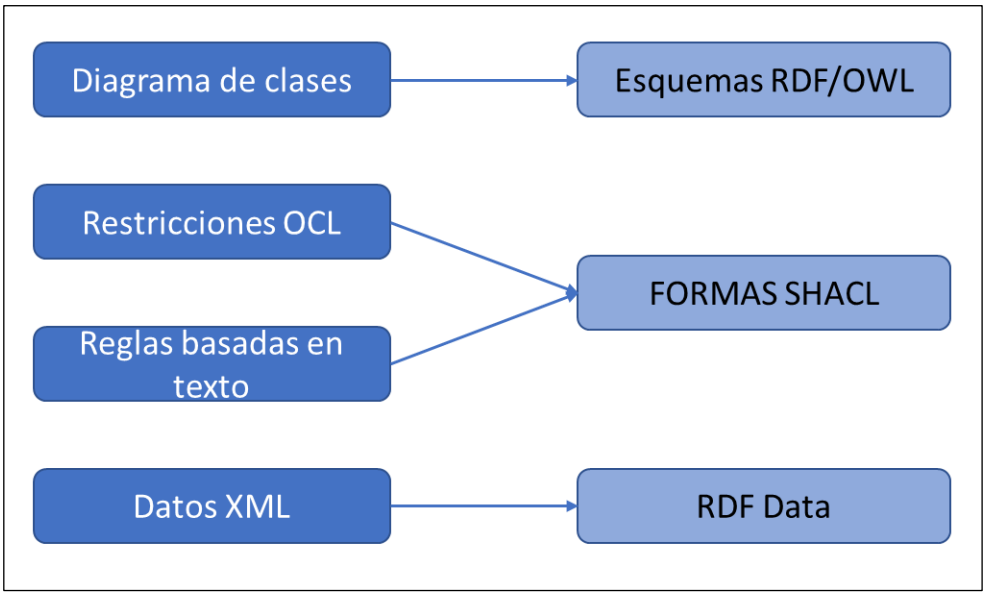


Figura 8.12: Transformación desde la Orientación a Objeto (UML+OCL) y reglas basadas en texto a (RDF+SHACL)

8.7.1 **OCL a SHACL**

Esta sección tiene como objetivo ilustrar cómo realizar la transformación de OCL a SHACL presentando las ideas clave que facilitan la transformación de UML a las tecnologías de la web semántica aplicadas al dominio de los sistemas de potencia (CGMES). Se deben tener en cuenta algunos requisitos durante la actividad de la transformación de OCL a SHACL tales como los siguientes:

a. Cómo se ejecuta el procesador SHACL recorriendo el grafo de datos y encajando el grafo de restricciones.

b. La tarea principal es producir tripletas a partir de los componentes de las expresiones OCL.

Un invariante en OCL se define de la forma siguiente. ``*Context ClassName inv: invariantName oclExpression*'' define una restricción identificada por *invariantName*, y donde *ClassName* representa el nombre de la clase de los objetos a los cuales la condición booleana indicada en *oclExpression* es aplicada. Se puede representar un invariante OCL y una forma SHACL por las clases UML como se muestra en la Figura 8.13. En la Figura 8.14 se presenta el mapeo entre *Invariant* y *ShaclRule*. El proceso de transformación de OCL a SHACL consiste en las siguientes etapas:

1. Extraer los conceptos principales del invariante OCL y construir con ellos la estructura inicial de la regla SHACL, como se ilustra en el algoritmo *Ocl2Shacl()* que se muestra en el Listado 8.4. Como OCL se usa para definir restricciones en los diagramas de clases UML, y SHACL es un lenguaje para validar datos RDF contra restricciones, el algoritmo *Ocl2Shacl()* necesita como parámetro *CIM_UML_RDF_TABLE*, una tabla que contiene la asignación de los conceptos de dominio entre UML y RDFS. El mapeo, que es parte inherente del estándar CGMES, se muestra en la Tabla 8.5 .

2. El mapeo entre los conceptos OCL y SHACL se encuentra en la observación de que el elemento primario o principal de una expresión OCL es la navegabilidad, y parte del objeto cuyo tipo se indica por defecto en la declaración del contexto de la restricción OCL, o explícitamente, como *Class.allInstances().count()* donde *Class* es cualquier clase en el diagrama de clases que representa el modelo. La idea principal o tarea es la creación de tripletas (sujeto predicado objeto) a partir de expresiones OCL de la forma *"a.x"* que representa la navegabilidad a partir de del objeto *"a"* en el que se aplica la regla a la característica x. El algoritmo *oclExpression2Sparql()* ilustra una manera de alto nivel sobre cómo se puede realizar esta tarea (ver Listado 8.5). La Tabla 8.5 muestra la semántica y el mapeo sintáctico entre OCL y SHACL. El Listado 8.6 representa un extracto de la gramática OCL, definida por OMG, y La Figura 8.15 muestra su correspondiente sintaxis abstracta donde el elemento primario tiene la siguiente forma: *exp ::= a opNav b (params)?* (donde *opNav* es *'.'* o *'->'*)

Figura 8.13: Clases UML para el invariante OCL y la forma SHACL.

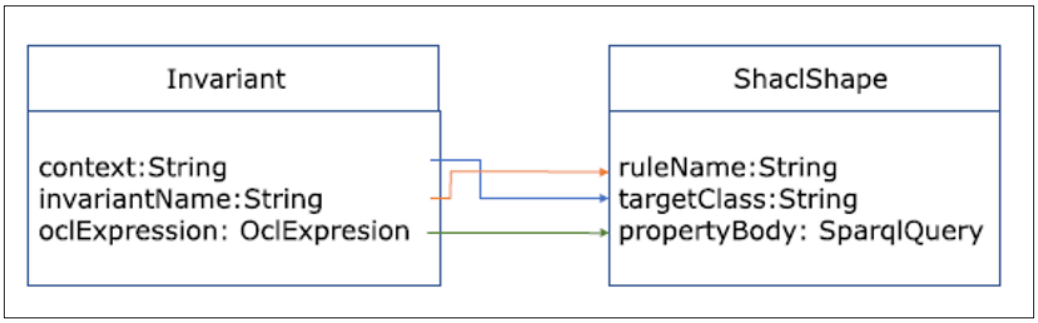


Figura 8.14: Mapeo entre el invariante OCL y la forma SHACL.

```
Ocl2Shacl(ocl_invariant,CIM_UML_RDF_TABLE, schemaDomain)
        # Initial structural transformation
        ruleName<- invariantName;
        targetClass<-context
        # oclExpresion to propertyBody transformation
        shaclShape<-
                "qodc:"+ruleName + " a sh:NodeShape;"
                # uml2rdf(umlConcept) is a simple function that returns
                # the rdf concept corresponding to uml concept
                # from table CIM_UML_RDF_TABLE
                "sh:targetClass " +  uml2rdf(context) " ;" +
                "sh:sparql [\n" +
                        " a sh:SPARQLConstraint ;\n" +
                                oclExpression2Sparql(oclExpression,
                                CIM_UML_RDF_TABLE, schemaDomain) +
                "]."
End
```

Listado 8.4: El algoritmo de transformación de los invariantes OCL a SHACL.

Concepto UML	Concepto CIM-RDF
……	……
IEC61970::Base::Wires::PowerTransformer.name	cim:IdentifiedObject.name
……	……

Tabla 8.5: uml2rdf, un extracto de la tabla de mapeo de los conceptos del dominio entre UML y RDF en CIM.

```
oclExpression2Sparql(oclExpression, CIM_UML_RDF_TABLE, schemaDomain)
.....
        # OCL to RDF transformation using navigability.
        foreach a_i  in a_1.a_2.....a_i.a_i+1.....a_n-1.a_n
                # create triple as follows
                ?a_i        a_i.a_i+1     ?a_i+1
        end foreach
.....
# For example a transformation of an expression like
#  a_1.a_2.....a_i.a_i+1.....a_n-1.a_n > 500 is:
# FILTER (?a_n > 500)

        FILTER (boolean condition with the corresponding pattern elements)

End
```

Listado 8.5: El algoritmo de transformación de la expresión OCL a consulta SPARQL.

```
OclExpressionCS ::= CallExpCS
CallExpCS ::= FeatureCallExpCS
FeatureCallExpCS ::= OperationCallExpCS
FeatureCallExpCS ::= PropertyCallExpCS
FeatureCallExpCS ::= NavigationCallExpCS

OperationCallExpCS ::= OclExpressionCS[1] simpleNameCS OclExpressionCS[2]
        | OclExpressionCS '->' simpleNameCS '(' argumentsCS? ')'
        |OclExpressionCS '.' simpleNameCS '(' argumentsCS? ')'
        | simpleNameCS '(' argumentsCS? ')'
        |OclExpressionCS '.' simpleNameCS isMarkedPreCS '(' argumentsCS? ')'
        |simpleNameCS isMarkedPreCS '(' argumentsCS? ')'
        | pathNameCS '(' argumentsCS? ')'
        | simpleNameCS OclExpressionCS
        | OclExpressionCS '.' pathNameCS '::' simpleNameCS '(' argumentsCS?
             ')'
        | OclExpressionCS '.' pathNameCS '::' simpleNameCS isMarkedPreCS '('
             argumentsCS? ')'

PropertyCallExpCS ::= OclExpressionCS '.' simpleNameCS isMarkedPreCS?
        | simpleNameCS isMarkedPreCS?
        | pathNameCS
        | OclExpressionCS '.' pathNameCS '::' simpleNameCS isMarkedPreCS?

NavigationCallExpCS ::= PropertyCallExpCS
        | AssociationClassCallExpCS

AssociationClassCallExpCS ::= OclExpressionCS '.' simpleNameCS ('['
       argumentsCS ']')? isMarkedPreCS?
        | simpleNameCS ('[' argumentsCS ']')? isMarkedPreCS?
argumentsCS[1] ::= OclExpressionCS ( ',' argumentsCS[2] )?
simpleNameCS ::= NameStartChar NameChar*
        | '_' #x27 StringChar* #x27
simpleNameCS[1] ::= simpleNameCS[2] WhiteSpaceChar* #x27 StringChar* #x27

NameStartChar ::= [A-Z] | "_" | "$" | [a-z]
| [#xC0-#xD6] | [#xD8-#xF6] | [#xF8-#x2FF]
| [#x370-#x37D] | [#x37F-#x1FFF]
| [#x200C-#x200D] | [#x2070-#x218F] | [#x2C00-#x2FEF]
| [#x3001-#xD7FF] | [#xF900-#xFDCF] | [#xFDF0-#xFFFD]
| [#x10000-#xEFFFF]
```

Listado 8.6: Extracto de la gramática OCL, especificada por OMG.

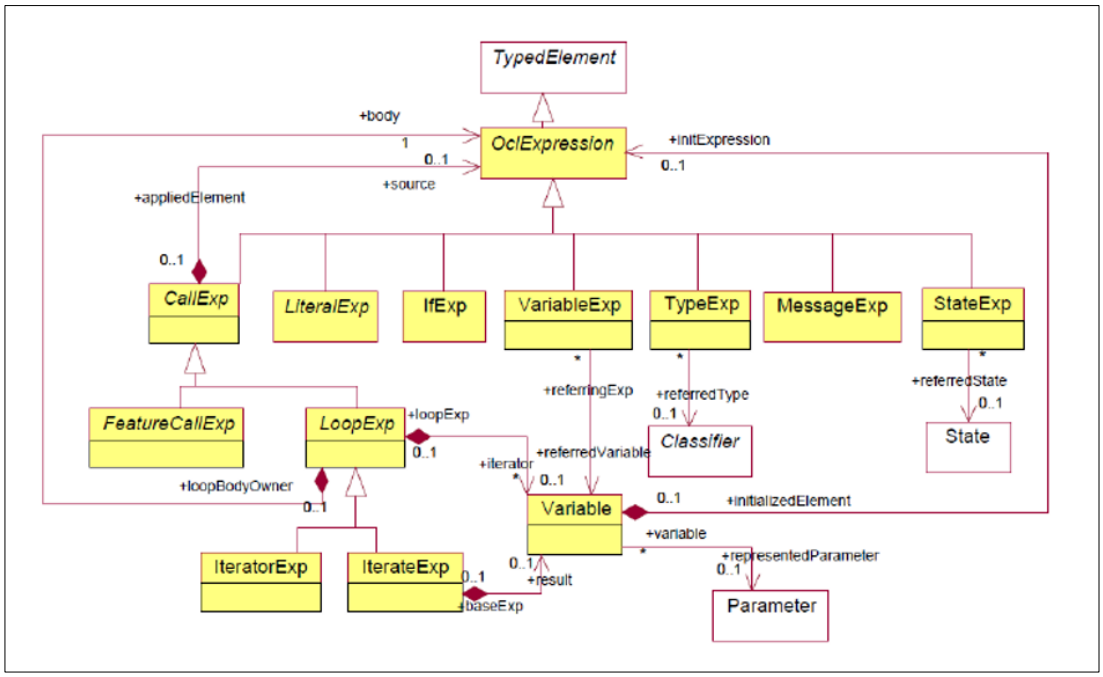


Figura 8.15: La estructura básica del metamodelo kernel de sintaxis abstracta para expresiones (OMG).

8.7.2 **Transformación de las reglas basadas en texto a SHACL**

Los parámetros del algoritmo son una regla basada en texto y un esquema de un dominio; la regla de texto se especifica en lenguaje natural en el contexto del dominio, en este caso, los sistemas de potencia; el esquema del dominio es el metamodelo del dominio definido en RDF. Los conceptos en la regla basada en texto se traducen a conceptos CIM, después se conectan haciendo uso del esquema del dominio para construir el patrón que corresponde a la regla original. La conversión del concepto de dominio en lenguaje natural a su correspondiente en CIM/XML puede ser realizada de formas: (i) manual, con la colaboración del experto del dominio o (ii) semiautomática, utilizando las tecnologías de procesamiento del lenguaje natural (PNL) en dos pasos. En primer lugar, se realiza un procesamiento basado en la similitud para determinar los conceptos del esquema RDF involucrados y el experto en el dominio interviene para seleccionar el más adecuado. Segundo, el patrón correspondiente a la regla se construye buscando los subgrafos que interconectan los conceptos que se encuentran en el paso previo e involucrando al experto en el dominio para seleccionar el patrón más apropiado. De acuerdo con estos dos enfoques, *textBasedToShaclManual()* describe el método manual (ver el Listado 8.7), y *textBasedToShaclSemiAutomatic()* describe el método semiautomático (ver el Listado 8.8).

```
textBasedToShaclManual(text_rule,schemaDomain)
        TBR = {is a set of the concepts in the Text-Based Rule}
        CC = {CIM Concepts initially empty}
        RC = {set of pair of related concepts, initially empty}
        Foreach c in TBR
                # dc is the domain concept defined in RDF schema.
                # rdfsConcept is a manual task performed by the model expert
                # to find the RDF concept that corresponds to textual
                # concept c.

                dc= rdfsConcept (c)
                CC->add(dc)
        End Foreach

        # constructPattern is a modeler manual task which consists of
        # searching for a subgraph that interconnects the concepts in CC.

        pattern = constructPattern(CC, schemaDomain)

        # getOperationsInTextBasedRules is a modeler manual task which
        # consists of extracting from text_based the operations and their
        # associated operands involved in the rule.

        OO = {(Operation,{Operands})} = getOperationsInTextBasedRules(
              text_rule)

        # createSHACLShape is a modeler manual task which consists of
        # building SHACL shape from the pattern and the operations stated
        # in text-based rule

        SHACLShape = createSHACLShape (pattern,OO)
End textBasedToShaclManual
```

Listado 8.7: El algoritmo manual de transformación de reglas basadas en texto a SHACL.

```
textBasedToShaclSemiAutomatic(text_rule,schemaDomain)

        # First, similarity-based processing is performed to determine the
        # RDF schema concepts involved in the rule SC = { is a set of
        # RDF schema concepts}

        SC = getSimilarities(text_rule, schemaDomain);

        # The domain expert intervenes to select the most appropriate match
        # with the task selectAppropriateConcepts

        CC = selectAppropriateConcepts(SC);

        # Second, the pattern corresponding to the rule is built by searching
        # for a subgraph  that interconnect the concepts found in the
        # previous step using any Pathfinding algorithm over schemaDomain
        # graph

        SG=getSubGraph(CC, schemaDomain);

        # With SG, the modeler builds the SHACL shape

End textBasedToShaclSemiAutomatic.
```

Listado 8.8: El algoritmo semiautomático de transformación de reglas basadas en texto a SHACL.

8.8. Implementación de la transformación de OCL a SHACL

8.8.1 Arquitectura de la aplicación

La arquitectura de la aplicación del conversor de OCL a SHACL se muestra en la Figura 8.16.

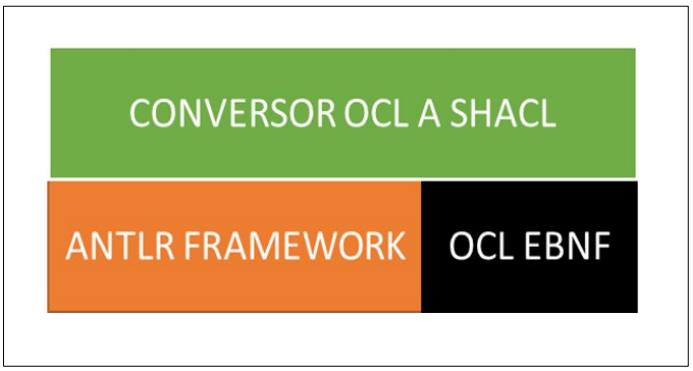


Figura 8.16: Arquitectura del conversor de OCL a SHACL.

ANTLR es un *framework* utilizado para el desarrollo de procesadores de lenguajes, y en este caso se ha empleado para la conversión de la sintaxis de un lenguaje a la sintaxis de otro. ANTLR proporciona un analizador léxico, un analizador sintáctico y un marco de trabajo para implementar la semántica del lenguaje. Se le suministra una gramática escrita en EBNF y, a partir de esta, genera automáticamente el analizador léxico y sintáctico. El esfuerzo recae en la implementación de la semántica del lenguaje.

8.8.2 Gramática de OCL

La especificación de OCL está en [23] y la gramática que se ha tomado para el actual trabajo se muestra en el Listado 8.6.

8.8.3 Sintaxis de SHACL

La sintaxis de SHACL está definida en la especificación de W3C correspondiente a [46].

8.8.4 Mapeo de conceptos

El mapeo de los conceptos para la estructura más externa de OCL consta de lo siguiente:

- El nombre asociado a la palabra reservada *inv* se asigna como nombre a las reglas SHACL.
- La clase de la palabra reservada *Context* le corresponderá el objeto del predicado *sh:targetClass*.
- La expresión booleana del invariante es convertida a su correspondiente restricción de SHACL.

8.8.5 Ejemplo de transformación de OCL de CGMES a SHACL

La aplicación, para la transformación de OCL a SHACL, toma un fichero OCL como entrada y genera un fichero SHACL. El Listado 8.9 contiene extracto de un fichero OCL del estándar

CGMES que se proporciona a la aplicación como parámetro y esta genera el código SHACL que se muestra en el Listado 8.10.

```
package TC57CIM

context IEC61970::Base::Core::IdentifiedObject
-- R.4.10.11. Description length restriction (optional)
inv equipmentDescriptionLength: self.description = null or self.description->size() <= 256
-- R.4.10.11. Energy Ident Code length restriction (optional)
inv equipmentEnergyIdentCodeEicLength: self.energyIdentCodeEic = null or
self.energyIdentCodeEic->size() = 16
-- R.4.10.11. Name length restriction
inv equipmentNameLength: self.name->size() <= 32
-- R.4.10.11. ShortName length restriction (optional)
inv equipmentShortNameLength: self.shortName = null or self.shortName->size() <= 12
```

Listado 8.9: Extracto de un fichero OCL del estándar CGMES.

```
@base <http://iec.ch/TC57/2013/CIM-schema-cim16#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix cim: <http://iec.ch/TC57/2013/CIM-schema-cim16#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix qodc: <http://entsoe.eu/CGMES2_4_15/QoCDC/3/0#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

qodc:equipmentDescriptionLength a sh:NodeShape;
        sh:targetClass cim:IdentifiedObject;
        sh:or (
                sh:property [
                        sh:path cim:IdentifiedObject.description;
                        sh:maxCount 0;
                ]
                sh:property [
                        sh:path cim:IdentifiedObject.description;
                        sh:maxLength 256;
                ]

        )
.

qodc:equipmentEnergyIdentCodeEicLength a sh:NodeShape;
        sh:targetClass cim:IdentifiedObject;
        sh:or (
                sh:property [
                        sh:path cim: IdentifiedObject.energyIdentCodeEic;
                        sh:maxCount 0;
                ]
                sh:property [
                        sh:path cim:IdentifiedObject.energyIdentCodeEic;
                        sh:minCount 16;
```

```
                        sh:maxCount 16;
                ]

        )
.

qodc:equipmentNameLength a sh:NodeShape;
        sh:targetClass cim:IdentifiedObject;
        sh:property [
                sh:path cim:IdentifiedObject.name;
                sh:maxLength 32;
        ]
.

qodc:equipmentShortNameLength a sh:NodeShape;
        sh:targetClass cim:IdentifiedObject;
        sh:or (
                sh:property [
                        sh:path cim:IdentifiedObject.shortName;
                        sh:maxCount 0;
                ]
                sh:property [
                        sh:path cim:IdentifiedObject.shortName;
                        sh:maxLength 12;
                ]

        )
.
```

Listado 8.10: Código SHACL generado automaticamente desde OCL del Listado 9.9.

8.9. **Desarrollo de la plataforma WEB de validación de las reglas CGMES**

En este trabajo se ha desarrollado una aplicación WEB de tipo Modelo Vista Controlador (MVC) y un conjunto de microservicios para validar modelos del dominio de los sistemas de potencia definidos con el estándar CIM. El lenguaje de marcado de hipertexto 5 (HTML5) es utilizado para implementar la parte de la vista; se representa la capa del modelo a través de RDF/RDFS/SPARQL/SHACL; la parte del controlador consta de un conjunto de clases de Java que gestionan las peticiones HTTP y que llevan a cabo la realización de las tareas de validación que invocan al procesador de SHACL. La Figura 8.17 muestra la arquitectura de capas de la plataforma desarrollada.

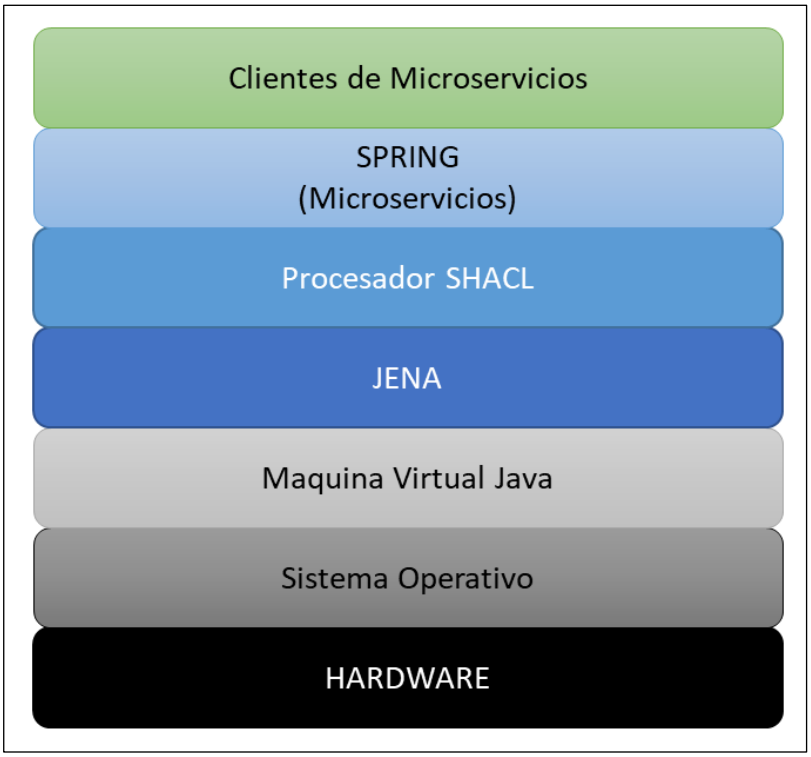


Figura 8.17: Arquitectura de capas de la aplicación desarrollada.

Una arquitectura más detallada de la aplicación que presenta los componentes y la comunicación entre ellos se muestra en la Figura 8.18.

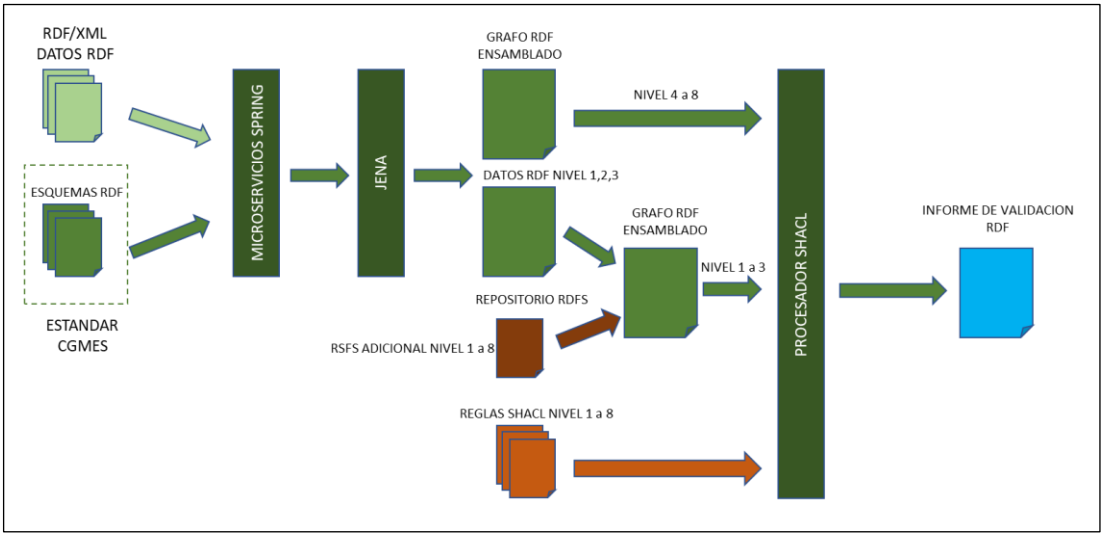


Figura 8.18: Arquitectura detallada de la aplicación.

Para fines de validación, el sistema utiliza RDF/XML y RDFS como datos de entrada, representando modelos del dominio de los sistemas de potencia. Los niveles de validación se han agrupado en tres categorías: (i) Un microservicio se encarga de la validación de archivos RDF en los niveles 1, 2 y 3. (ii) Un segundo

microservicio se ocupa de validación de nivel 4 (OCL). (iii) Finalmente, un tercer microservicio trata de los niveles 5 a 8. La razón de esta separación es que los niveles 1, 2 y 3 se ocupan de la nomenclatura de archivos, la corrección sintáctica de XML, espacios de nombres, encabezados y empaquetado de archivos. El nivel 4 se ocupa de las reglas OCL definidas por el estándar. Los niveles 5 a 8 se ocupan de la coherencia entre perfiles, reglas de negocio relacionadas con el flujo de carga, la consistencia del modelo IGM *(Individual grid model)* y la plausibilidad del CGM *(common grid model)*. Se proporciona un microservicio para el proceso de ensamblaje (*Assembly*), que consiste en producir un grafo a partir de los cuatro archivos (Perfiles) que representan un IGM (EQ, TP, SV y SSH junto con el *Boundary set)* y los RDFS correspondientes. La interfaz de usuario de la aplicación se muestra en la Figura 8.19.

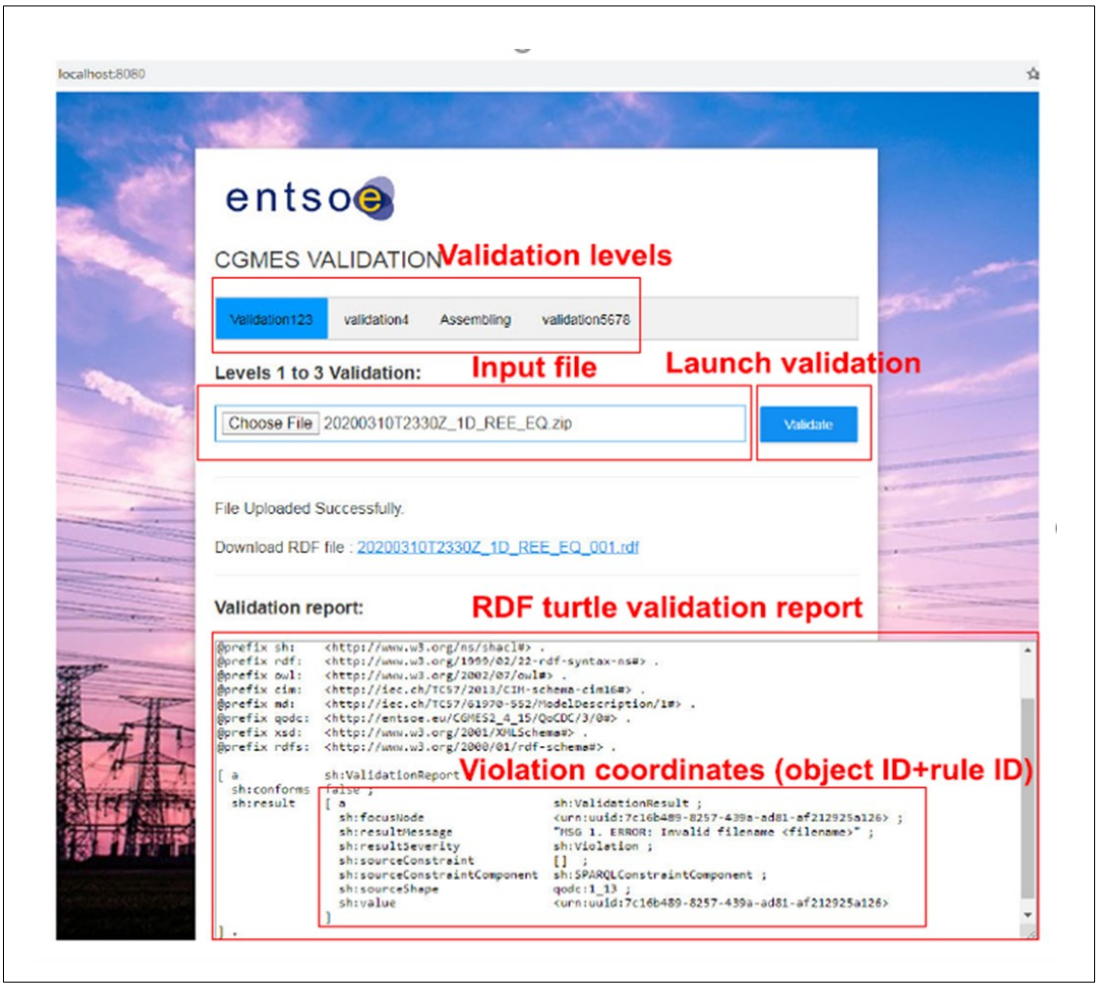


Figura 8.19: Interface de usuario de la plataforma de validación.

Un objetivo adicional de esta aplicación consiste en evaluar los siguientes aspectos: (i) La contribución de adoptar RDF/SHACL dentro del enfoque MDE. (ii) El impacto de utilizar modelos directamente ejecutables en el desarrollo de software. (iii) La mejora en escalabilidad, mantenibilidad, agilidad y otros aspectos del proceso de desarrollo de software.

Para desarrollar una aplicación en el contexto de sistemas de potencia utilizando este enfoque, se requiere lo siguiente: (i) Experiencia en CIM, UML, RDF/S, SPARQL y SHACL. (ii) Realizar el mapeo de conceptos expresados en lenguaje natural a conceptos CIM. (iii) Comprender el modelo y la semántica de una regla. (iv) Navegar dentro del metamodelo (RDFS) y los modelos (RDF) del dominio. (v) Construir patrones de grafos que correspondan a las reglas y traducirlos en consultas SPARQL, que posteriormente se integrarán en SHACL.

El desarrollo de la aplicación para la validación de los modelos CGMES, ha requerido el uso de tecnologías software como microservicios Spring, una biblioteca de validación SHACL (librería JENA), HTML5 y una librería de JavaScript.

8.10. Conclusión y trabajos futuros.

En este trabajo, se ha propuesto una solución basada en un estándar para abordar el problema de validación en los modelos definidos en UML/OCL, adoptando SHACL como lenguaje de validación. Con este propósito, se ha especificado una metodología para la conversión de reglas OCL y reglas basadas en texto. Dado que CGMES se basa en UML/OCL, se requiere una transformación de reglas a RDF/SHACL.

Los argumentos para adoptar SHACL como lenguaje de validación han sido presentados a través de una comparación entre los dos enfoques (UML/OCL vs. RDF/SHACL) considerando criterios esenciales para la tarea de validación del modelo, tales como informe de validación, razonamiento, entre otros.

Los conceptos básicos y constructos de SHACL han sido presentados como un primer acercamiento a los fundamentos semánticos y sintácticos de este lenguaje de validación dentro del dominio del modelado.

La metodología para la transformación de las reglas OCL y basadas en texto a SHACL ha sido presentada como algoritmos. Para la conversión de OCL a SHACL se ha tomado en consideración los elementos claves en términos sintácticos y semánticos en el contexto de las dos gramáticas. La conversión de las reglas basadas en texto ha sido abordada desde la perspectiva lingüística hacia la sintaxis y semántica de SHACL.

En este trabajo se han implementado todas las reglas, definidas por CGMES, en SHACL. También se ha desarrollado una aplicación web con microservicios Spring para validar los ocho niveles especificados en el estándar CGMES que están disponibles para la comunidad de usuarios, tanto de sistemas de potencia como de ingeniería de software. Las aportaciones presentadas en este trabajo son: (i) Transformar OCL y reglas basadas en texto a SHACL, una tarea nunca abordada anteriormente, a través de una metodología que permite una implementación de las reglas. (ii) Proporcionar a ENTSO-E, a la comunidad de los sistemas de potencia y las partes interesadas una ontología que les permite validar sus modelos. Además, puede servir como guía para la implementación de sus propias reglas. (iii) Poner a disposición de la comunidad del dominio de los sistemas de potencia, una aplicación web desarrollada en el contexto de este trabajo para validar los modelos CGMES contra los ocho niveles de validación especificados en la norma. (iv) Proporcionar un puente para la brecha entre los

paradigmas orientado a objetos y ontológico abordando la conversión de OCL a SHACL, ya que se está realizando un esfuerzo considerable de diferentes comunidades para fomentar la adopción de la ingeniería ontológica a través de las tecnologías de la web semántica. (v) Se ha abordado el alineamiento entre los lenguajes de las reglas asociadas al aspecto estructural de ambos paradigmas. Este trabajo se ha centrado en la versión del estándar CGMES que actualmente es adoptado por los miembros de la ENTSO-E. Una limitación menor de este trabajo es la optimización de las reglas expresadas en SHACL, en términos de rendimiento para el procesador SHACL, y la implementación de nuevas reglas SHACL para nuevas versiones del estándar CGMES a corto plazo. Para avanzar en esta investigación, está planeado desarrollar un lenguaje especifico de dominio (DSL) en el dominio de los sistemas de potencia, permitiendo a los expertos escribir reglas de texto para ser transformadas en SHACL automáticamente.

9. Alineamiento comportamental y el enfoque para abordar la especificación del comportamiento

9.1. Resumen

Todo proceso de ingeniería, especialmente el software, involucra dos aspectos complementarios: estructural y comportamental. El comportamiento es, en esencia, la transformación estructural asociada al sistema. Como lenguaje para el paradigma orientado a objetos, el lenguaje UML ofrece constructos para ambos aspectos, por ejemplo, diagramas de clase para el aspecto estructural y diagramas de actividad para el aspecto comportamental. Sin embargo, los modelos obtenidos no son directamente ejecutables ni ofrecen las ventajas del enfoque como caja de cristal, como producto en su ciclo de vida, y tampoco soportan el razonamiento. Por otro lado, cuando se aborda la ingeniería de software con ontologías, solo se proporcionan constructos para aspectos estructurales para desarrollar un modelo directamente ejecutable, gracias a su capacidad de razonamiento. Sin embargo, no existen constructos o enfoques para la especificación o definición del comportamiento en este paradigma. Esta carencia aparece principalmente en las primeras etapas del proceso de ingeniería de software, donde no existen constructos similares a, por ejemplo, el diagrama de actividad en el dominio orientado a objetos. OMG ya abordó la transformación entre los dos paradigmas en términos estructurales a través del metamodelo de definición de ontología (ODM) de UML a RDF y OWL. Sin embargo, no se ha abordado la transformación de los constructos comportamentales orientados a objetos en sus correspondientes ontológicos, porque no están definidos en el paradigma ontológico. Este trabajo aborda la definición del comportamiento en el paradigma ontológico y la transformación de los constructos del comportamiento entre los dos paradigmas. La base fundamental de la especificación del comportamiento es el concepto de flujo, y la base de éste es la transformación del modelo estructural en un sentido evolutivo. Por tanto, una vez definido el comportamiento en el dominio de la ontología, los artefactos obtenidos a lo largo del ciclo de vida son directamente ejecutables, y su

validación y testeo son automáticos. Con este enfoque, el ciclo de vida se reduce a un proceso de modelado. Así, el proceso de ingeniería de software resultante mejora características como la agilidad, la simplicidad, la productividad y el formalismo. La audiencia objetivo de este trabajo es la comunidad de ingeniería de software, especialmente en el paradigma de la ingeniería dirigida por modelos (MDE) abordado desde perspectivas ontológicas y orientadas a objetos. La evaluación del enfoque propuesto se ha realizado en el dominio de las compañías eléctricas y sistemas de potencia, resolviendo el problema del flujo de validación para el proceso de interoperabilidad especificado por el estándar CGMES *(Common Grid Model Exchange Standard)*.

9.2. **Introducción**

En cualquier proceso de ingeniería, para obtener un artefacto específico, es necesario construir sus componentes estructurales y de comportamiento. Estos dos aspectos son interdependientes, ya que uno se construye sobre el otro. Para lograr un comportamiento determinado, se requiere una estructura adecuada, y viceversa (ver Figura 9.1).

En el ámbito de la ingeniería de software, específicamente en el paradigma orientado a objetos, el lenguaje UML desempeña un papel fundamental en las etapas iniciales del ciclo de vida del desarrollo. Su aplicación principal se encuentra en la fase de diseño y especificación de requisitos, en la cual se emplean diferentes constructos para abordar los aspectos estructurales del sistema, tales como diagramas de clases y objetos, junto con el uso de reglas del lenguaje OCL.

Asimismo, se utilizan otros constructos orientados al comportamiento, como máquinas de estado y diagramas de actividad, con el fin de representar y comprender las interacciones dinámicas del sistema. No obstante, es importante tener en consideración que los artefactos generados mediante el uso del UML no son directamente ejecutables, ni poseen capacidades de razonamiento. En su lugar, se trata de representaciones visuales y estructuradas que facilitan la comunicación y comprensión entre los diferentes actores involucrados en el proceso de desarrollo de software.

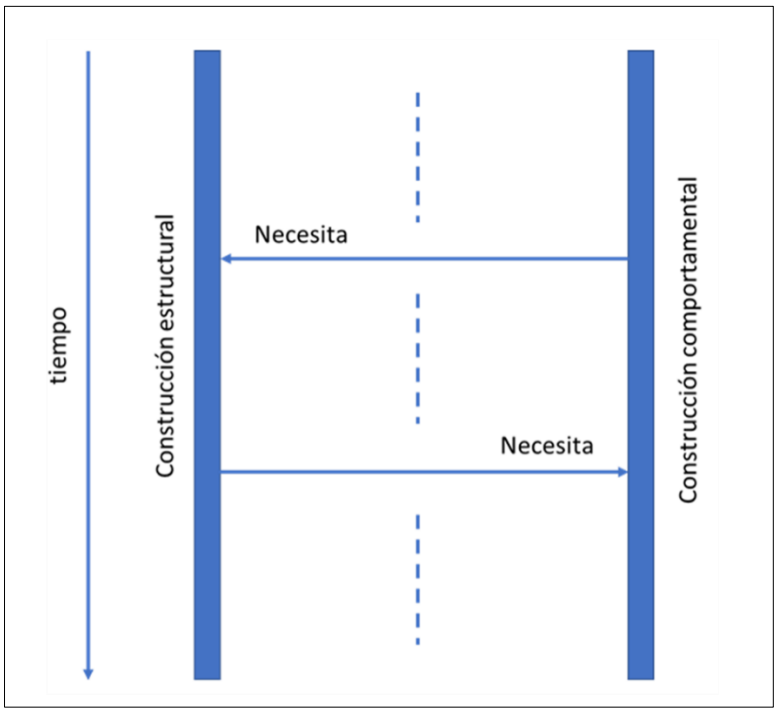


Figura 9.1: Dependencia de los procesos de construcción estructural y comportamental

Por otro lado, en el paradigma ontológico se utilizan las tecnologías de web semántica proporcionadas por W3C, como RDF, OWL, SPARQL y SHACL, para el aspecto estructural, obteniendo así artefactos directamente ejecutables dada la capacidad de razonamiento que tienen implícitamente. Sin embargo, no se proporciona ningún lenguaje para especificar el comportamiento, similar a lo que ofrece UML, como el diagrama de actividad o la máquina de estado. Por lo tanto, no se dispone de un lenguaje con enfoque ontológico para el proceso de ingeniería del software, especialmente en las primeras fases del ciclo de vida, para la especificación y definición del comportamiento.

El enfoque propuesto se ha centrado en la ontología proporcionada por W3C, la cual se basa en las tecnologías ampliamente utilizadas de la web semántica, tales como RDF, OWL, SHACL y SPARQL. Estas tecnologías han sido adoptadas de manera extensa debido a su amplio uso en diversos contextos.

Hasta dónde llega el conocimiento en esta investigación, en el estado del arte no existe un lenguaje específicamente diseñado para la especificación del comportamiento en el contexto del paradigma ontológico. Para abordar la especificación de requisitos relacionados con el comportamiento de un sistema en el paradigma ontológico, se ha seguido el enfoque de envolver los modelos ontológicos en modelos orientados a objetos y posteriormente realizar funciones en dichos modelos, o bien interactuar directamente con operaciones SPARQL CRUD.

El enfoque propuesto para abordar esta brecha se basa en un lenguaje que se fundamenta en los siguientes constructos: tarea *(Task)*, condición previa *(precondition)*, modelo *(inputModel/outputModel)* y condición posterior *(postCondition)*. La semántica asociada con este lenguaje establece que una tarea se

ejecuta cuando se cumple una determinada condición previa en un modelo específico, y que un modelo de salida puede estar disponible después de su ejecución. La validez de la postcondición debe ser verificada con relación a los modelos.

La orquestación de la ejecución de los grafos/flujo, que representan el comportamiento, se logra mediante el procesador de flujo, el cual es un componente central de este enfoque. De esta manera, el comportamiento se representa a través de un flujo en el que los nodos corresponden a las tareas, tal como se muestra en la Figura 9.2 y en la Figura 9.3. Dichas figuras representan el núcleo del enfoque, su representación visual como lenguaje y la transición entre dos tareas.

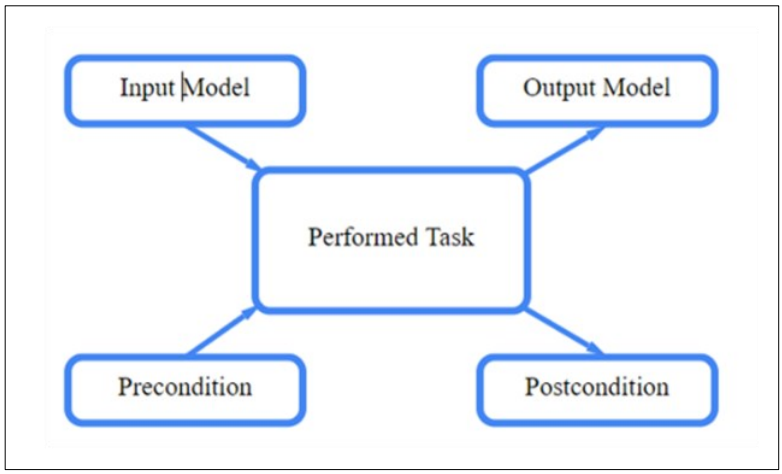


Figura 9.2: El Constructo visual básico para el enfoque comportamental

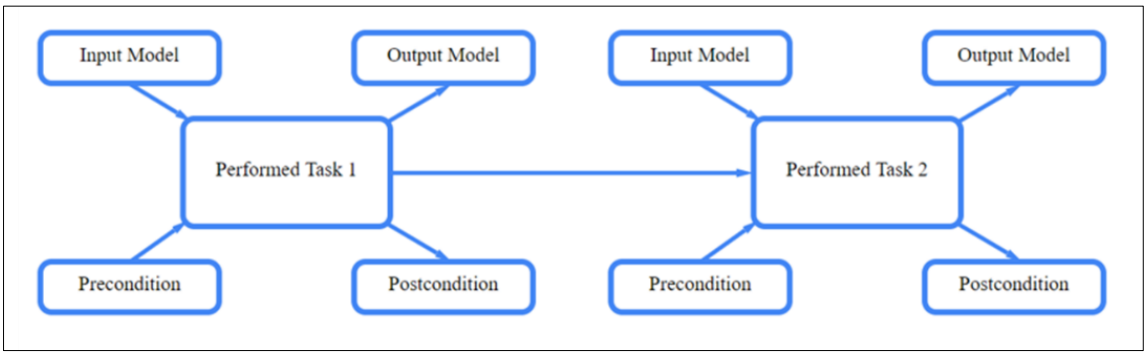


Figura 9.3: Los elementos clave del enfoque con dos elementos y una transición

Se ha llevado a cabo una transformación de los constructos comportamentales básicos proporcionados por UML en el paradigma orientado a objetos hacia los constructos del enfoque propuesto. Esta transformación permite la interoperabilidad entre sistemas heterogéneos y se considera una evaluación del enfoque.

La evaluación se ha realizado sobre CGMES, un estándar que consiste en un modelo UML que representa los elementos del dominio de las compañías eléctricas y un conjunto de reglas. Las reglas se clasifican en ocho niveles y se especifican en lenguaje natural y en OCL, cuya transformación a SHACL ha sido abordada en la sección

8.7. La evaluación en esta sección consiste en aplicar el lenguaje propuesto al proceso de validación abordado como comportamiento.

Un modelo de datos generado por una entidad específica, como un TSO *(Operador del Sistema de Transmisión),* está sujeto a ocho procesos de validación correspondientes a los ocho niveles de validación especificados en QOCDC *(Quality of CGMES Datasets and Calculations)*, un conjunto de reglas proporcionadas por el estándar CGMES *(Common Grid Model Exchange Standard)*.

La comunidad de ingeniería de software, los ingenieros de modelado de ontologías, la comunidad dedicada a la definición de estándares y las partes interesadas involucradas en CIM CGMES para compañías eléctricas se beneficiarán de este enfoque de modelado de comportamiento.

En la sección 9.3 se presenta la bibliografía relacionada con el estado del arte del enfoque propuesto. La sección 10.4 aborda el enfoque de modelado comportamental en el dominio ontológico y proporciona un ejemplo didáctico para ilustrar su aplicación. A continuación, en la sección 9.5, se lleva a cabo la transformación de los constructos básicos de UML a los constructos del enfoque propuesto. En la sección 9.6 se modela, con el lenguaje propuesto, el flujo de validación del estándar CGMES. Por último, la sección 9.7 contiene las conclusiones y las direcciones futuras de investigación.

9.3. Revisión del estado del arte

En el estado del arte, no se ha encontrado ningún trabajo que permita hacer que los constructos UML sean ejecutables directamente a través de un procesador con capacidad de razonamiento, y que utilice un enfoque de caja de cristal.

Para las primeras fases de la ingeniería de software, especialmente la especificación de requisitos con ontologías, existen *frameworks* y herramientas para el aspecto estructural. Sin embargo, la comunidad se limita a utilizar el modelado estructural de las ontologías con Protégé, Jena, AllegroGraph, StarDog, RDF4J entre otros, y el comportamiento se implementa envolviendo las ontologías en tecnología orientada a objetos o a través de constructos SPARQL para operaciones CRUD. Por lo tanto, se necesita un lenguaje visual de alto nivel que las partes interesadas puedan usar desde diferentes contextos y procedencias, tanto técnicos como no técnicos.

El modelado de comportamiento en orientación a objetos con UML cubre la fase de especificación de requisitos, como diagramas de interacción, diagramas de actividad y máquinas de estado. Sin embargo, no se proporcionan artefactos que sean directamente ejecutables con capacidad de razonamiento; incluso si el código se puede producir automáticamente, se necesita una fase de refinamiento posterior.

Hasta dónde llega el conocimiento en esta investigación, en el estado del arte no se encuentra disponible ningún lenguaje visual como UML o una adaptación del proceso de software con ontologías para las primeras fases.

Los trabajos encontrados en las investigaciones de revisión bibliográfica relacionadas con ontologías suelen centrarse en el desarrollo de ontologías para dominios específicos. Sin embargo, no se ha encontrado ningún esfuerzo de investigación académica que aborde de manera integral el proceso de desarrollo de software utilizando ontologías para abordar tanto los aspectos estructurales como los comportamentales. Además, las soluciones existentes son ad-hoc y carecen de la característica de reusabilidad de procesos.

Los siguientes trabajos de investigación están relacionados con los conceptos del enfoque propuesto, aunque todos están desarrollados desde una perspectiva orientada a objetos.

Los criterios para clasificar los trabajos relacionados, que son útiles para la presente investigación, son los siguientes: i) Ejecutabilidad directa [ED], ii) El uso de BPMN para abordar flujos y orquestación en un dominio determinado [FBPMN], iii) La aplicación del enfoque ontológico tanto a un dominio estructural como comportamental [OADD], iv) La transformación entre diferentes paradigmas y lenguajes [TR], v) La validación de modelos en diferentes niveles de abstracción [VAL], vi) Modelado ontológico [MOD-ONT], vii) Modelado orientado a objetos [MOD-OO], viii) *Frameworks* que abordan estos criterios [FRMK], ix) Revisiones bibliográficas en las áreas correspondientes a estos criterios [LR]. La Tabla 9.1 ilustra la clasificación de la revisión bibliográfica según el conjunto de criterios descritos anteriormente.

Work ref.	ED	FBPMN	OADD	TR	VAL	FRMK	LR	MOD-ONT	MOD-OO
[42]								X	
[43]			X					X	
[44]			X					X	
[45]			X						
[46]			X					X	
[47]			X					X	
[48]			X					X	
[49]			X					X	
[50]			X					X	
[51]			X					X	
[52]			X					X	
[53]			X					X	
[54]							X		
[55]								X	
[56]						X	X		
[57]							X		X
[58]							X		X
[59]		X		X					
[70]		X							X
[71]		X		X					
[72]		X		X					
[73]		X							
[74]				X	X	X			
[75]				X	X	X			
[76]				X				X	
[77]									X
[78]									X

Tabla 9.1: Clasificación de la revisión bibliográfica con respecto a los criterios de interés para el enfoque propuesto.

Los siguientes son esfuerzos que aplican ontologías a un dominio específico:

- En [51], se propone un motor de base semántica para ejecutar procesos de negocio basados en la selección de servicios web y su orquestación en tiempo de ejecución. Utiliza una ontología diseñada para ajustarse a los requisitos de los usuarios y optimizar la selección de servicios desde un repositorio.
- En [52], se modelan e implementan interfaces adaptativas de teléfonos móviles a través de un enfoque ontológico con tecnologías de la web semántica como SPARQL y OWL, sin utilizar SHACL.
- En [53], se desarrolla una ontología para el diagrama de secuencia UML, sin proporcionar constructos específicos para el comportamiento en el desarrollo con ontologías.
- En [54] , se aplica una ontología al dominio de la gestión de crisis del medio ambiente natural para prevenir desastres, abordando el IoT con tecnologías de la web semántica. Utiliza BPMN para el flujo de trabajo, y el motor de flujo se implementa con Activiti. En este enfoque, las reglas se diseñan e implementan separadamente de la definición del flujo, lo que genera un problema de escalabilidad.
- En [55], se realiza una aplicación de una ontología al dominio químico, pero no se utiliza la tecnología de la web semántica. En su lugar, se utiliza la generación automática de código utilizando grafos para las ontologías.
- En [56], se integra una ontología en el dominio de los flujos de trabajo científicos.
- En [57], se ha desarrollado una ontología para áreas específicas correspondientes a ciertas partes de las compañías eléctricas y sistemas de potencias, sistemas multiagente y los servicios web.
- En [58], se ha desarrollado una ontología para los dominios de flujo de ríos y mitigación de inundaciones. La herramienta utilizada en el proceso de construcción de la ontología es Protégé, que proporciona únicamente la definición estructural del modelo.
- En [59], se integran ontologías con requisitos y servicios web.
- En [60], se utiliza una ontología para modelar eventos en el dominio de las compañías eléctricas y sistemas de potencia de la red nacional india. Sin embargo, en el enfoque de esta sección, el evento no se considera un concepto elemental, sino más bien la ejecución de una tarea que se realiza cuando se cumple una condición previa en un modelo dado.
- En [61] , se propone un enfoque basado en ontologías para generar control automático distribuido a partir del diseño de automatización de subestaciones. En el contexto de este sistema descentralizado se ha utilizado la transformación de ontologías como parte del enfoque MDE.

- En [62], se presenta una plataforma para la autorización de datos vinculados *(linked data)* utilizando un lenguaje para políticas de protección de datos enlazados. El lenguaje de la política se ha implementado sobre SPARQL.

Los siguientes trabajos son revisiones de literatura y encuestas sobre los criterios relacionados con el enfoque del trabajo de esta sección:

- En [63], se realiza una encuesta sobre la orquestación de servicios de red (NSO), revisando los antecedentes históricos, los proyectos de investigación relevantes, las tecnologías habilitadoras y los esfuerzos de estandarización.
- En [64], se lleva a cabo una compilación y presentación de los resultados de la investigación y los beneficios potenciales de aplicar ontologías para abordar tres desafíos en la ingeniería de software: i) dificultad para comunicar y compartir información; ii) gestión eficaz de las fases de desarrollo de software; y iii) técnicas y entornos de desarrollo para apoyar la producción de software semántico a través de un enfoque interdisciplinario.
- En [65] , se realiza una encuesta sobre las suites de gestión de procesos de negocio.
- En [66], se realiza una revisión bibliográfica sistemática sobre enfoques y herramientas de modelado para sistemas embebidos.
- En [67], se presenta una revisión bibliográfica sobre la aplicación de ontologías a la ingeniería de sistemas. También se aborda la transformación entre UML y constructos de ontología en el estado del arte. El trabajo señala que mapear todos los constructos de ambos paradigmas es imposible, pero se considera que el mapeo para todos los constructos no es necesario, ya que los paradigmas no son equivalentes y su utilidad radica en las diferencias semánticas entre ellos.

Los siguientes trabajos están relacionados con el uso de BPMN junto con criterios como el modelado y la transformación:

- En [68], se trata de mejorar la productividad relacionada con la implementación de servicios a través de M2M *(Model to Model)* y M2T *(Model to Text)*, sin considerar los siguientes criterios: i) ejecutabilidad directa, ii) enfoque de caja de cristal *(Glass box)*, iii) capacidad de razonamiento y iv) uso de tecnologías de la web semántica.
- En [69], se especifica la composición y orquestación de microservicios en BPMN, utilizando coreografías basadas en eventos para su ejecución. El nivel de abstracción se ha elevado a través de constructos BPMN.
- En [70], se realiza la abstracción para encapsular el procesamiento de flujo de eventos como un conjunto de funciones de negocio, utilizando cadenas de procesos orientados a eventos (EPC) y BPMN 2.0. Se mapean las transformaciones a la representación de procesos ejecutables con un motor de ejecución e integrándolo con software de terceros.
- En [71], se modela el comportamiento de IoT dentro del contexto de BPMN para procesos de negocio, realizando una transformación a código *Callas*, un lenguaje de programación para sensores ejecutado a través de una máquina virtual para dispositivos IoT.

- En [72], se integra BPMN con el enfoque basado en procesos PDA *(Process-Driven Approach)*. Los procesos definidos con BPMN se hacen ejecutables gracias a un motor de procesos encargado de orquestar el flujo definido por BPMN.

Los siguientes trabajos se basan en la transformación:

- En [73], se ha desarrollado un marco para la verificación formal en el diseño de sistemas embebidos utilizando la lógica computacional en árbol (CTL), OCL para el SystemVerilog y un motor de transformación para obtener el código de bajo nivel a partir del de más alto nivel definido con lenguaje de modelado de sistemas (SysML) y UML.
- En [74], se presenta un marco para especificar el diseño y la verificación de requisitos mediante el desarrollo de un perfil basado en UML y SysML para representar la semántica de SystemVerilog desde el bajo nivel de abstracción de RTL hasta el alto nivel de abstracción proporcionado por UML. Como parte del esfuerzo, se ha desarrollado un motor de transformación para generar código RTL (nivel de transferencia de registro) junto con el código de aserción SystemVerilog (SVA).
- En [75], se propone un proceso de transformación del modelo independiente de computación (CIM) al modelo independiente de plataforma (PIM) y la validación de modelos con un enfoque basado en ontologías.

Los siguientes trabajos abordan el modelado, los criterios de comportamiento y la evaluación de lenguajes de modelado:

- En [76], se presenta un marco para el enfoque de desarrollo dirigido por comportamiento (BDD) mediante la introducción de perfiles UML.
- En [77], se propone un modelo de calidad para evaluar los lenguajes de modelado de procesos de software, permitiendo caracterizarlos y compararlos.

Los trabajos que abordan la ejecutabilidad directa se basan en la generación automática de código sin capacidad de razonamiento. En el caso de lograr la ejecutabilidad directa, esta se realiza en términos de caja negra. BPMN se usa en algunos trabajos para modelar el flujo de comportamiento en dominios específicos, como la orquestación de servicios web. Se necesita un motor BPMN para que los modelos BPMN sean ejecutables, pero se implementa como un artefacto de caja negra sin capacidad de razonamiento. BPMN está destinado a la interoperabilidad humana, aunque se ha forzado a ser ejecutable en la máquina a través de un motor BPMN.

Muchos trabajos aplican el enfoque ontológico a un dominio particular para obtener modelos, pero solo en términos estructurales. Sin embargo, una vez obtenida la ontología, los modelos adquieren capacidad de razonamiento. Hay trabajos que tratan sobre la transformación entre diferentes lenguajes de distintos niveles de abstracción considerando diferentes paradigmas. Se han considerado los trabajos

consistentes en marcos que incluyen criterios de interés para el enfoque propuesto; esto es útil para presentar el enfoque propuesto como un marco y una herramienta. Dados los criterios de interés del enfoque propuesto, en que trabajo se han considerado encuestas y revisiones bibliográficas.

En la presente investigación no se encontró trabajo alguno relacionado con el modelado de comportamiento con ontologías o un lenguaje visual que permita la ejecución directa de un flujo.

9.4. **Especificación del comportamiento desde el enfoque ontológico**

Dado que no se dispone de un lenguaje de alto nivel para las fases de especificación de requisitos, lo cual es útil para los interesados tanto técnicos como no técnicos para el modelado del comportamiento del proceso de ingeniería de software con un enfoque ontológico, este trabajo tiene como objetivo proporcionar un enfoque metodológico respaldado por un lenguaje de modelado visual para abordar el proceso de especificación del comportamiento en el campo ontológico.

El comportamiento se puede representar como un flujo. Un flujo se puede describir como un grafo. Los constructos básicos del flujo son tarea, transición, evento, decisión, paralelismo y sincronización, como se representa en la Figura 9.4. BPMN y el diagrama de actividad, entre otros, se especifican con este enfoque para las primeras fases del ciclo de vida de la ingeniería de software, como la especificación de requisitos.

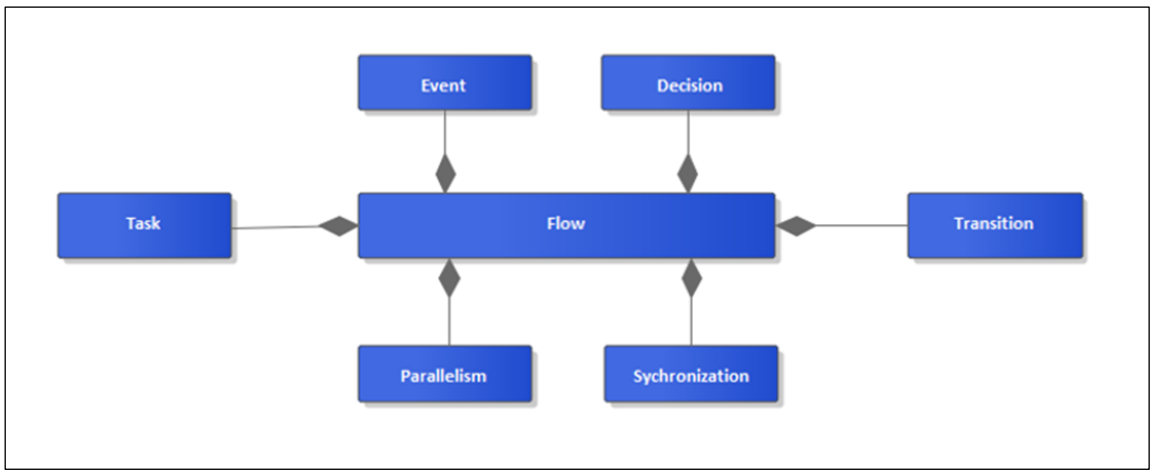


Figura 9.4: Constructos básicos de un flujo.

Las siguientes son definiciones de los conceptos básicos implicados en el comportamiento.

- La tarea es una transformación realizada en los modelos.

- La transición indica el camino desde la ejecución de una tarea hasta la ejecución de otra.
- evento = (modelo anterior, condiciones, transformación, modelo actual, tal que modelo anterior <> modelo actual).
- La decisión es la opción que se ejecutará una vez que una condición haya sido validada contra un modelo.

La idea principal del enfoque en este trabajo surge de la observación de que ciertas condiciones deben cumplirse en un conjunto específico de recursos para ejecutar una algo específico. Aunque pueda parecer trivial, es una definición rigurosa que ofrece suficientes conceptos para desarrollar un lenguaje visual directamente ejecutable.

El comportamiento, en su esencia, es la transformación de un modelo. Lo que se va a ejecutar es la tarea, modelada con el paradigma ontológico directamente ejecutable. Primero, la información estructural y de comportamiento se definirán utilizando un modelo ontológico. Luego, las condiciones serán modeladas e implementadas con reglas ontológicas para validar el modelo de información. El concepto de decisión en el enfoque de esta tesis se basa en determinar si el modelo de información cumple con las reglas que definen las condiciones.

Definición: Una transición es un camino de una tarea a otra una vez finalizada la primera, que es esencialmente una transformación de un modelo.

Transición = <tarea1, tarea2, terminada(tarea1), iniciada(tarea2)>

El enfoque de esta tesis en contexto del modelado comportamental consta de los siguientes constructos básicos: tarea, modelo, condición previa, condición posterior y transición entre tareas (flechas), como se ilustra en la Figura 9.5.

La idea básica consiste en ejecutar una tarea específica cuando se cumple una condición particular sobre un conjunto de datos específicos (modelo). El desarrollo del enfoque que se realiza con las tecnologías de la web semántica es el siguiente.

Los modelos se definen utilizando los lenguajes RDF o OWL. Los modelos pueden ser simples o complejos; en este caso, estarán construidos por un grupo de referencias a grafos que representan modelos. La condición previa se construye utilizando un conjunto de reglas SHACL.

La tarea se puede modelar usando el lenguaje SPARQL para realizar las transformaciones que están detrás de las operaciones CRUD.

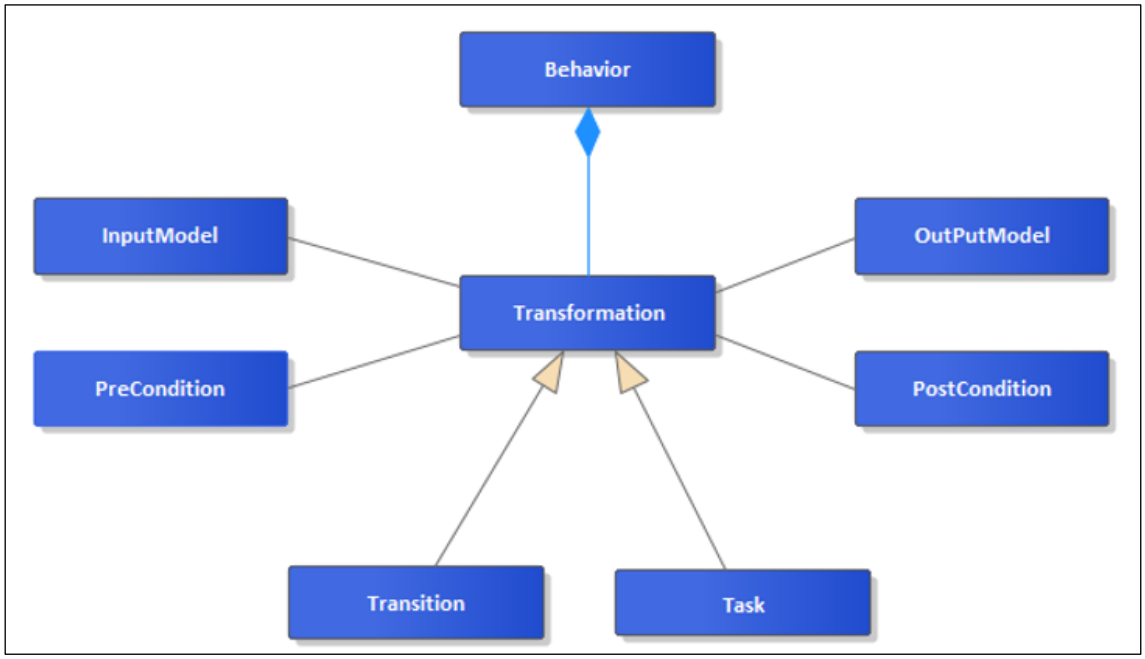


Figura 9.5: Modelo del comportamiento con los constructos básicos.

La semántica del enfoque propuesto se muestra en la Figura 9.6.

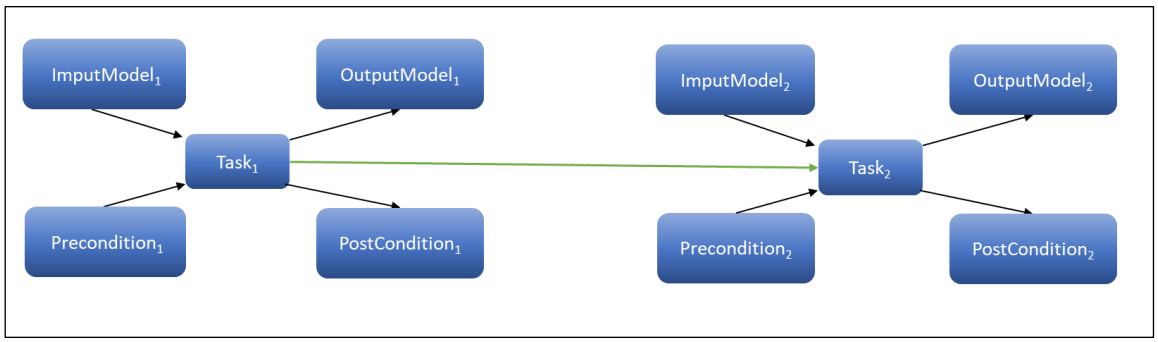


Figura 9.6: El caso básico para el enfoque presupuesto.

Para ejecutar la tarea *Task₁*, *InputModel₁* debe ser válido con respecto a *precondition₁* y *Postcondition₁* debe cumplirse en los modelos de interés para los requisitos. Una vez que se ejecuta *Task₁*, se ejecuta *Task₂* si el *InputModel₂* es válido con respecto a *Precondition₂*. De acuerdo con el enfoque propuesto, el procesador de flujo está a cargo de orquestar y ejecutar el modelo de flujo que representa el comportamiento. La semántica que representa el flujo de comportamiento será ejecutada por el procesador de flujo que en este trabajo se ha implementado a través de un algoritmo en Java.

La transición (flecha verde) tiene un modelo de entrada, un modelo de salida, una condición previa, una condición posterior y una tarea T, como se muestra en la Figura 9.7. Después de ejecutar *Task1*, *Task₂* se ejecuta solo si el modelo de entrada cumple la condición previa de la transición y se ha ejecutado la tarea de transición T.

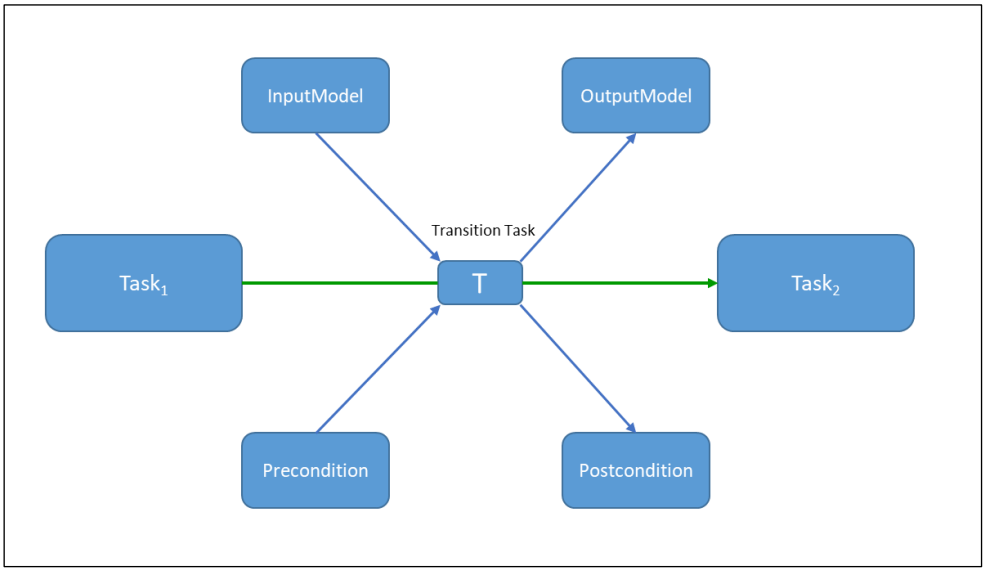


Figura 9.7: Modelo de la transición.

La arquitectura del procesador de flujo se muestra en la Figura 9.8.

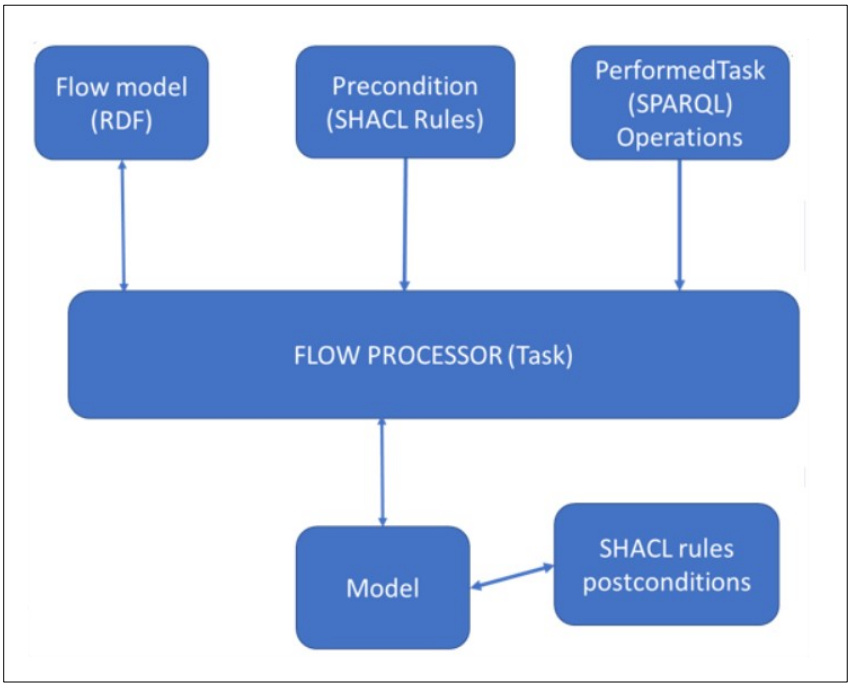


Figura 9.8: Arquitectura del procesador de flujo.

El flujo del procesador de flujo que se muestra en la Figura 9.9 y la Figura 9.10 ilustra los conceptos básicos y las acciones para ejecutar el flujo. El proceso se ejecuta en cada nodo del grafo que representa el flujo. La flecha roja indica la iteración. Llegado a una tarea, cuando el resultado de la validación de su modelo de entrada con respecto a su condición previa tiene un valor falso, se pueden adoptar las siguientes opciones para configurar el comportamiento del procesador de flujo. (i) Queda

permanentemente pendiente hasta que el resultado de la validación sea verdadero para pasar a la siguiente tarea, tal como se presenta en la Figura 9.9. (ii) Ignora las tareas salientes del nodo actual y nunca vuelve a ejecutar la tarea actual, como se presenta en la Figura 9.10.

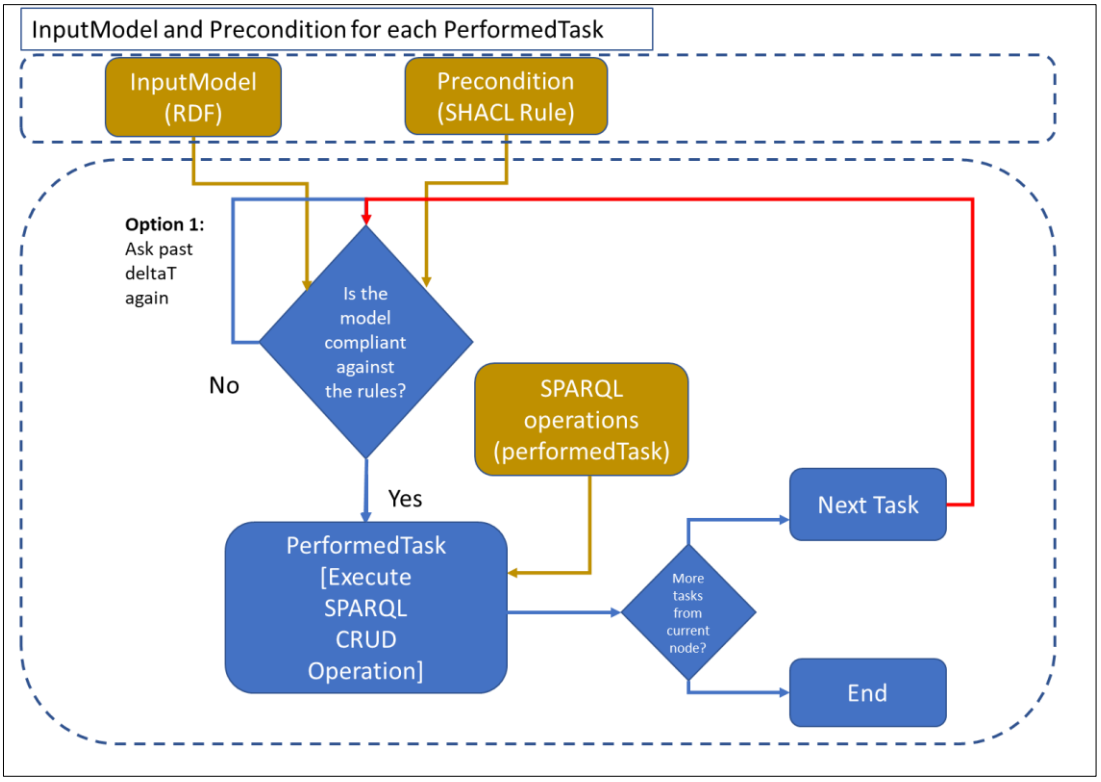


Figura 9.9: Flujo del procesador de flujo para la opción 1.

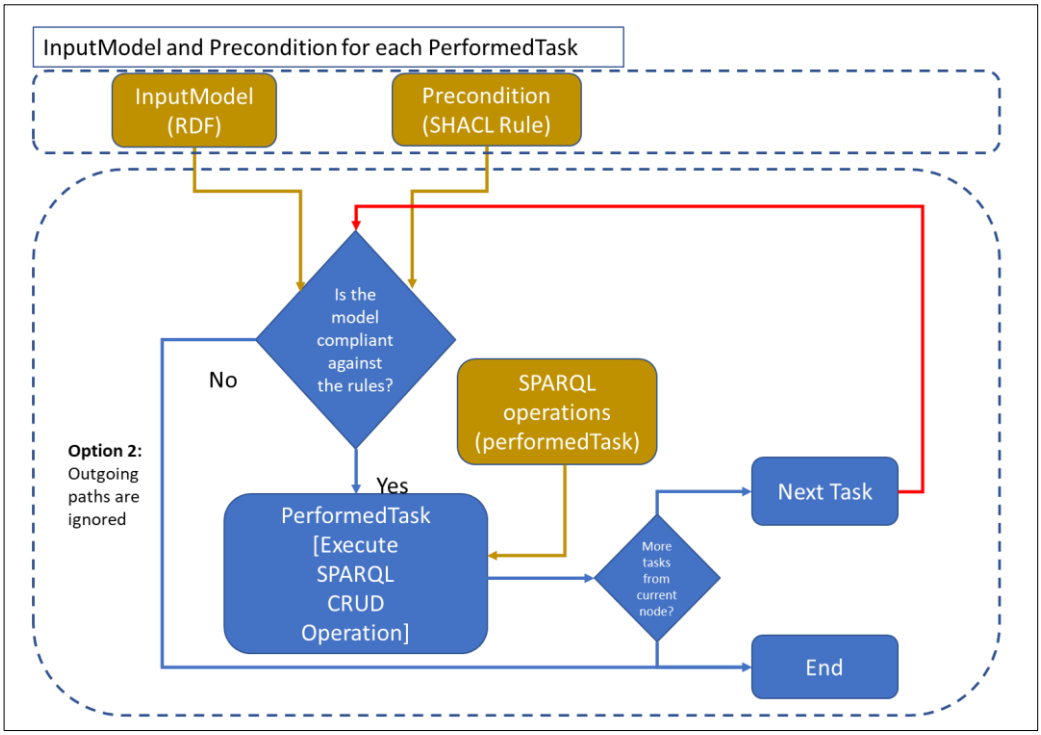


Figura 9.10: Flujo del procesador de flujo para la opción 2.

El algoritmo que corresponde al procesador de flujo se muestra en el Listado 9.1.

La Figura 9.11 muestra la clase *Node* para implementar el enfoque propuesto.

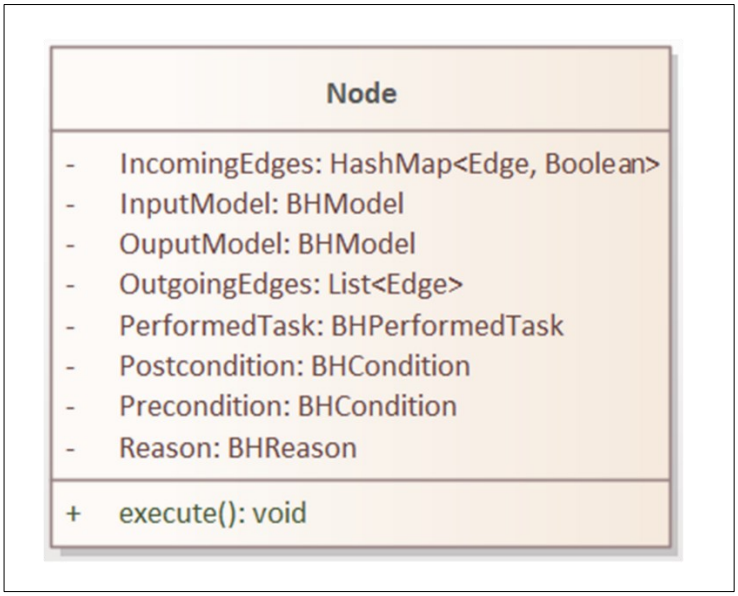


Figura 9.11: Clase Node.

El grafo detrás del flujo se ejecuta visitando cada nodo. Un nodo puede ser de los siguientes tipos:

(i) StartNode; en este caso se obtienen y ejecutan sus aristas de salida y sus correspondientes nodos de destino;

(ii) GatewayNode; En este caso, *InputModel* se valida contra la *PreCondition* de GatewayNode y se obtienen EdgeYes, EdgeNo, NodeYes y NodeNo. Si la validación es correcta, se ejecuta el NodeYes; de lo contrario, se ejecuta NodeNo; finalmente, todos los nodos de tipo SyncNode que son accesibles desde la ruta que no se ejecutará, son notificados para que no se consideren por el SyncNode.

(iii) PerformedTask; este escenario ejecuta la tarea correspondiente a este nodo, y se obtienen las aristas salientes y sus correspondientes nodos de destino. Si el nodo de destino no es un SyncNode, se ejecuta; de lo contrario, se notifica al SyncNode que esta arista entrante ya ha llegado a este nodo y si han llegado todos los IncomingEdges, se ejecuta el SyncNode.

(iv) SyncNode; se obtienen las aristas de salida y sus correspondientes nodos de destino. Para cada nodo: si no es de tipo SyncNode, se ejecuta; en caso contrario (si es de tipo SyncNode), si han llegado todos los IncomingEdges, se ejecuta el nodo.

v) EndNode; No se hace nada ya que no debe tener aristas salientes.

```
# Execute the graph  behind the flow
function executeGraph() {
    sn = getStartNode();
    execute(sn);
}

function execute(node) {
   # If the current Node is equals to start node
    if(nodeType==StartNode){
      # The outgoing edges are obtained from the current node
      # which represent concurrent paths.
       oe = getOutgoingEdges();
      # Each node is executed concurrently
       foreach edge in oe {
          node = getNextNode(edge)
           execute(node);
           }
      }
      # If the current node is a gateway node
      else if(nodeType==GateWayNode){
            # The outgoing edges are obtained from the current node
            oe = getOutgoingEdges();
            # The input model is validated against the gateway node
            validationResult = validate(getPrecondition(),getInputGraph())
            # Edge yes, edge no, node yes and node no are obtained
            nodeYes = getNodeYes();
            nodeNo = getNodeNo();
            edgeYes = getEdgeYes();
            edgeNo = getEdgeNo();
```

```
            # If the validation  confoms.
            if(validationResult){
                    # All reachable sync nodes from the path that is not traversed are
notified not to consider the
                    # resulting incoming edges.
                    desableSyncPaths(edgeNo);
                    # Node Yes is executed.
                    execute(nodeYes);
            }
            # Else the inverse is executed
            else{
                  desableSyncPaths(edgeYes);
                 execute(nodeNo);
            }
      }

      # If the current node is a performed task
      else if(nodeType==PerformedTask){
           # The task corresponding to this node is executed
           executeTask(getTask());
           # The outgoing edeges are obatained from the current node
           oe = getOutgoingEdges();
           foreach edge in oe {
               # The node connected to the edege is obtained
               node = getNextNode(edge)
               # if the obtained node is a SyncNode
               if(getType(node)==SyncNode){
                     # This Edge is registered as arrived
            node.hasArrived(edge,true);
                     # If all edges that should arrive have arrived the syncNode is
executed
                     # (go to next nodes in the graph)
            if(allEdgesHaveArrived(node)){
                  execute(node);
            }
                }
                else{
                      execute(node);
                }
            }
      }
     # If the current node is a sync node
      else if(nodeType==SyncNode){
         # The outgoing edeges are obatained from the current node
         oe = getOutgoingEdges();
         foreach edge in oe {
                # The node connected to the edege is obtained
```

```
            node = getNextNode(edge)
             if(getType(node)==SyncNode){
              node.hasArrived(edge,true);
              # If all edges that should arrive have arrived the syncNode is executed
               if(allEdgesHaveArrived(node)){
              execute(node);
        }
         }
         else{
        execute(node);
         }
       }
    }
    # If the current node is end node nothing is done.
    else if(nodeType==EndNode){
    }

}
```

Listado 9.1: Algoritmo del procesador de flujo.

El modelo de procesador de flujo con el enfoque propuesto se ilustra en la Figura 9.12. Donde: (1) representa el procesador de flujo, que permite la ejecución directa. (2) denota el modelo de flujo que representa el proceso detrás del comportamiento a ejecutar, que es un grafo RDF. (3) son las reglas SHACL que establecen las condiciones sintácticas para la gramática del lenguaje visual, que define el flujo. (4) es el modelo RDF que contiene la información requerida sobre el estado de ejecución del flujo en términos estructurales. Finalmente, (5) son las condiciones en SHACL que deben cumplirse una vez que se ejecuta el flujo.

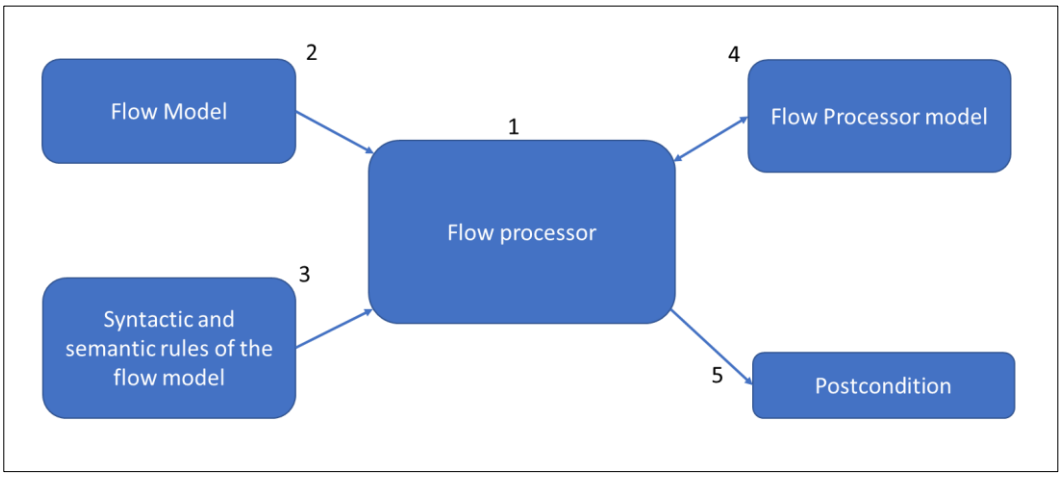


Figura 9.12: modelo del procesador de flujo definido con el enfoque propuesto.

El Listado 9.2 muestra un extracto del modelo ontológico del flujo.

```
@prefix bh: <http://www.bh.org/bh#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

bh:Node rdf:type rdfs:Class.
bh:StartNode rdfs:SubClass bh:Node.
bh:EndNode rdfs:SubClass bh:Node.
bh:PerformedTask rdfs:SubClass bh:Node.
bh:Gateway rdfs:SubClass Node.
bh:SyncNode rdfs:SubClass Node.
bh:Transition rdf:type rdfs:Class.
bh:Condition rdf:type rdfs:Class.
bh:Model rdf:type rdfs:Class.
bh:Task rdfs:SubClass bh:Node.

bh:inputModel rdf:type rdf:Property;
        rdfs:domain bh:Model;
        rdfs:range bh:Node.
bh:precondition rdf:type rdf:Property;
        rdfs:domain bh:Condition;
        rdfs:range bh:Node.
bh:ouputModel rdf:type rdf:Property;
        rdfs:domain bh:Node;
        rdfs:range  bh:Model.
bh:postCondition rdf:type rdf:Property;
        rdfs:domain bh:Node;
        rdfs:range bh:Condition.
```

Listado 9.2: Modelo ontológico del flujo.

'@prefix bh: <http://www.bh.org/bh#> . ' es el prefijo que representa el espacio de nombres del contexto de modelado de comportamiento.
En la Figura 9.13 se muestra una instancia del modelo ontológico del flujo, que representa todos los constructos involucrados en el enfoque. Este flujo ha sido extraído desde la plataforma desarrollada.

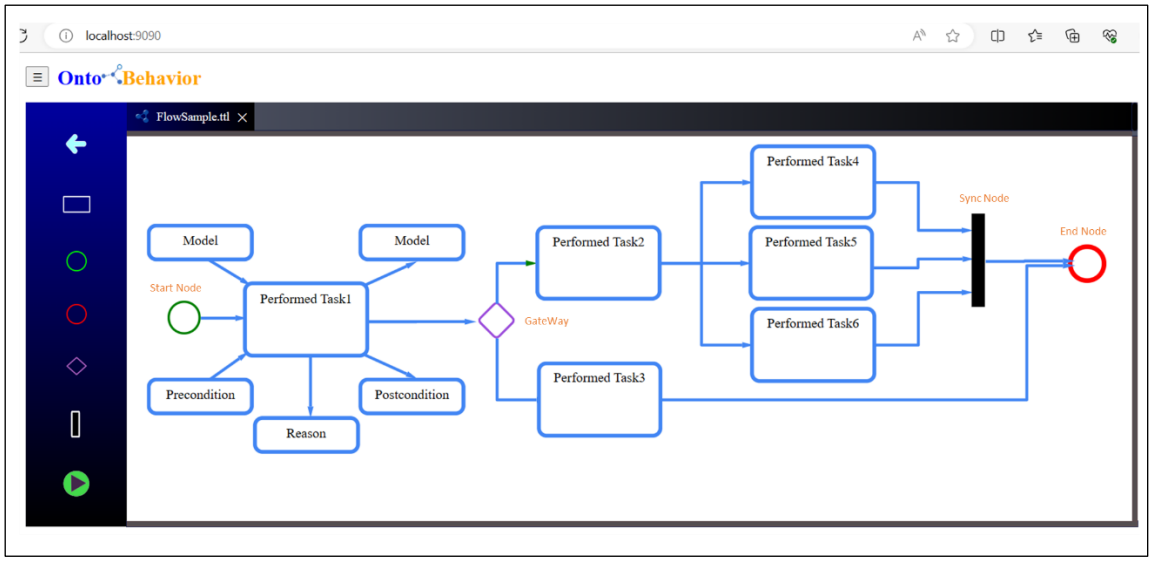


Figura 9.13: Un ejemplar de flujo con los constructos propuestos.

En el Listado 9.3 se muestra un extracto de la definición en SHACL de las condiciones previas que representan las reglas sintácticas y semánticas del modelo de flujo.

```
# A bh:transition represented by an arrow must have one and only one bh:from node.
bh:br1 a sh:NodeShape;
        sh:targetClass bh:Transition;
        sh:property[
                sh:path bh:from;
                sh:minCount 1;
                sh:maxCount 1
        ].
# A bh:transition represented by an arrow must have one and only one bh:to node.
bh:br2 a sh:NodeShape;
        sh:targetClass bh:Transition;
        sh:property[
                sh:path bh:to;
                sh:minCount 1;
                sh:maxCount 1
        ].
```

Listado 9.3: Reglas sintácticas definidas en SHACL para el enfoque propuesto.

La Figura 9.14 muestra un ejemplo simple de dos tareas que se ejecutan secuencialmente extraído de la plataforma desarrollada.

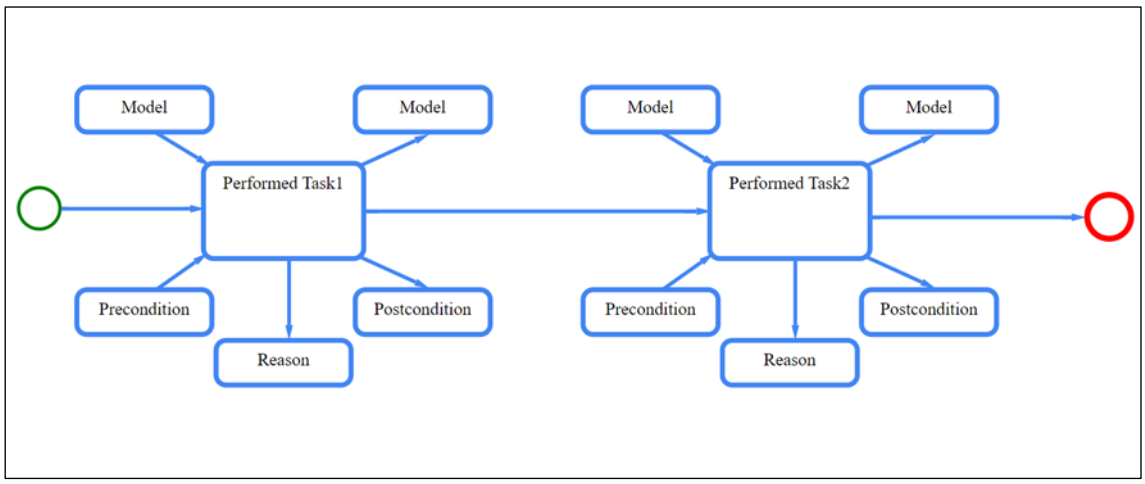


Figura 9.14 . Un ejemplar de flujo simple.

El RDF correspondiente al flujo anterior se ilustra en el Listado 9.4. Los identificadores de los objetos RDF que corresponden a elementos visuales son los valores de los atributos de los elementos SVG de cada concepto visual en el diagrama.

```
<urn:uuid:bfbeadd9-4d2e-4106-860a-8aa78f7ebf05> rdf:type bh:StartNode.
<urn:uuid:d4b3c780-9943-4f58-81c8-806a3eb15dfb> rdf:type bh:EndNode.
<urn:uuid:f3b62a31-404c-4f13-889c-0dc65a7c262c> rdf:type bh:Task;
        bh:inputModel <urn:uuid:0a7d2786-13bc-4e5b-b3ae-7302b7686845>;
        bh:precondition <urn:uuid:beee3615-d927-46a6-88b4-2d9c26e26351>;
        bh:outputModel <urn:uuid:d2cd035d-cc32-41b1-8805-4b65a8791ac5>;
        bh:postcondition <urn:uuid:c900f09b-d9ea-42bd-bf4d-d04d603cf86f>.
<urn:uuid:ef54264b-57ef-4194-ab26-3cc2fe944e9b> rdf:type bh:Task;
         bh:inputModel <urn:uuid:cec40576-2609-415d-8204-c8ccb5d16e37>.
        bh:precondition <urn:uuid:6f82e849-3882-443d-be8d-888144692326>.
        bh:outputModel <urn:uuid:9cb6b3c6-2074-43c3-abaa-c2dd2983f128>.
        bh:postcondition <urn:uuid:532b5424-b67e-4d1e-8462-0747ec15f99d>.
<urn:uuid:d5499124-71f9-4eec-b505-4db1eb6af660> rdf:type bh:Transition;
        bh:from <urn:uuid:bfbeadd9-4d2e-4106-860a-8aa78f7ebf05>;
        bh:to <urn:uuid:f3b62a31-404c-4f13-889c-0dc65a7c262c>.
<urn:uuid:6e2ad94d-db61-421b-8211-a5fbd710b3f4> rdf:type bh:Transition;
        bh:from <urn:uuid:f3b62a31-404c-4f13-889c-0dc65a7c262c>;
        bh:to <urn:uuid:ef54264b-57ef-4194-ab26-3cc2fe944e9b>.
<urn:uuid:ba621958-22b6-4f86-9a0c-615452dd3295> rdf:type bh:Transition;
        bh:from <urn:uuid:ef54264b-57ef-4194-ab26-3cc2fe944e9b>;
        bh:to <urn:uuid:d4b3c780-9943-4f58-81c8-806a3eb15dfb>.
```

Listado 9.4: Un extracto del RDF que representa el modelo de la figura 10.14.

9.5. **Ejemplo de aplicación del enfoque al modelo de presa de agua**

Este ejemplo ilustra cómo abordar un problema específico de un dominio particular con el paradigma ontológico utilizando el enfoque propuesto: modelo, condición previa, condición posterior y tarea realizada. El modelo físico que representa el tanque de agua se muestra en la Figura 9.15.

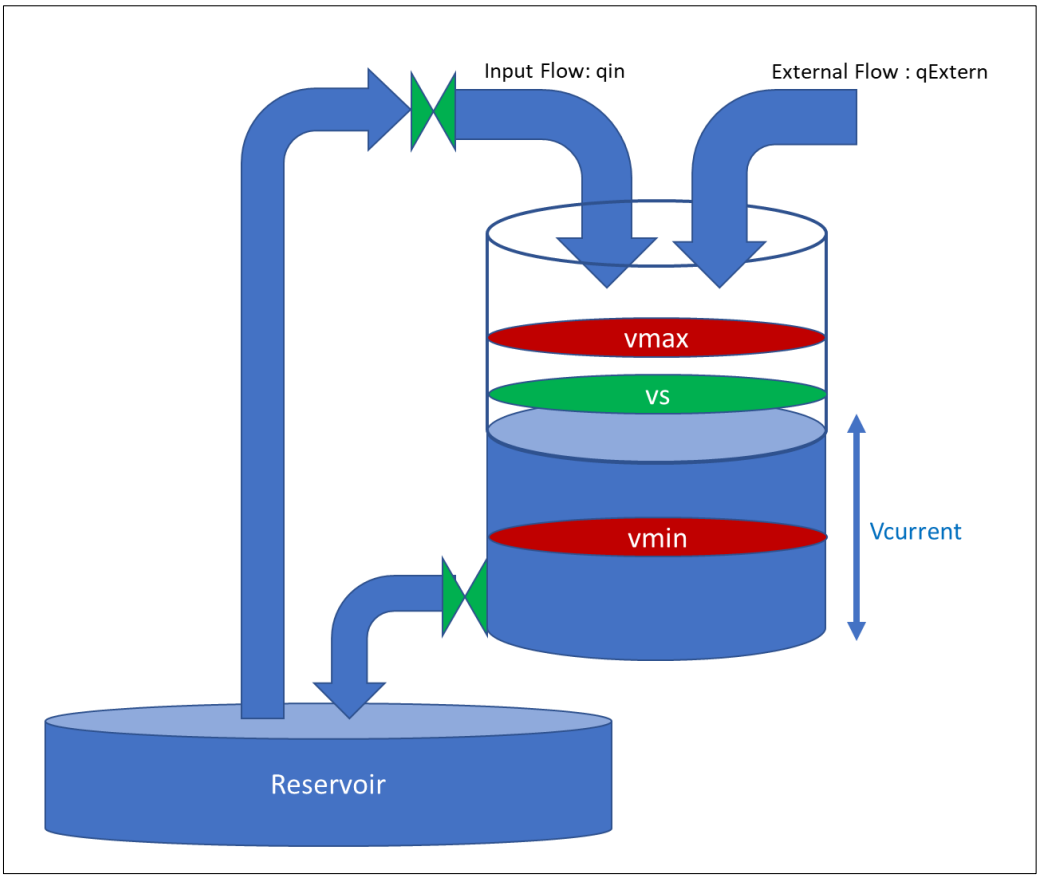


Figura 9.15: Modelo físico de la presa.

El objetivo es controlar el volumen de agua "vcurrent", que se regula mediante "vs", teniendo en cuenta que el volumen mínimo "vmin" y el volumen máximo "vmax" son fijos. La válvula " valve1" controla el caudal de entrada al tanque y la válvula " valve2" controla el caudal de salida. El usuario decide qué consigna asignar, y el sistema reacciona para mantener la "vcurrent" en el valor de "vs" actuando sobre las dos válvulas.

Los escenarios que se han considerado son los siguientes:

- Escenario 1: Abrir la válvula 2 y cerrar la válvula 1 si vc > vs.
- Escenario 2: Cerrar la válvula 2 y abrir la válvula 1 si vc < vs.
- Escenario 3: Cerrar la válvula 2 y cerrar la válvula 1 si vc == vs.

De acuerdo con el enfoque propuesto para un modelo de tanque de agua simple como una presa, el modelo de comportamiento se presenta en la Figura 9.16, extraído de la plataforma desarrollada.

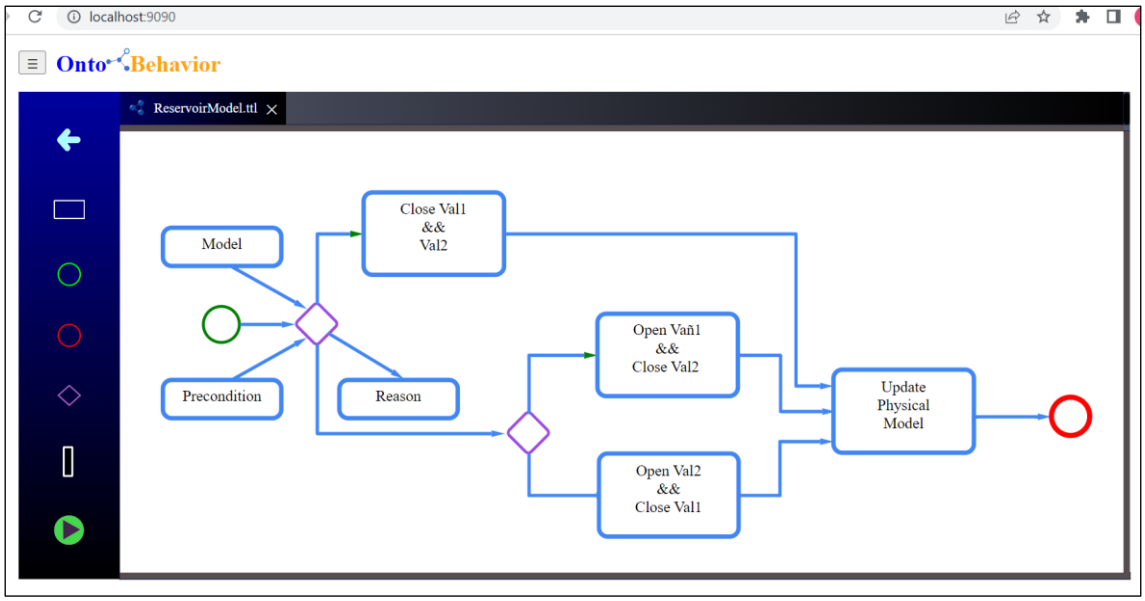


Figura 9.16: Modelado de comportamiento para un tanque de agua simple.

El modelo RDF asociado a este flujo se muestra en el Listado 9.5.

```
@prefix bh: <http://www.bh.org/bh#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<urn:uuid:d074e4d6-c11b-439f-b58c-d5cf02ea2ba7> rdf:type bh:StartNode.
<urn:uuid:793842e0-0e9c-403f-9b49-999ec6193389> rdf:type bh:EndNode.
<urn:uuid:50070bd9-3c44-4980-89d6-ec246d8162c5> rdf:type bh:Task;
   bh:inputModel <urn:uuid:f84743f3-e94d-4654-9ff1-7a0c53eebd77>;
   bh:precondition <urn:uuid:b69fff5a-197c-4f38-8bd2-941d9ff7612f>;
   bh:outputModel <urn:uuid:b3c6eb67-aa9a-4425-b49e-3cae26174ae5>;
   bh:postcondition <urn:uuid:7828a2e0-0c60-4d09-8530-ee9b78aa2b4f>.
<urn:uuid:65498537-6b14-43d1-831c-f334273b9808> rdf:type bh:Task;
   bh:inputModel <urn:uuid:b5f078d5-5121-4e70-a96f-83e2d563c943>;
   bh:precondition <urn:uuid:fd73af94-7c20-45db-b9f6-218482309107>;
   bh:outputModel <urn:uuid:a7cd2062-3ac0-4859-aa3b-982b7ebe59bc>;
   bh:postcondition <urn:uuid:bfa540fa-b856-415a-98ea-6c4abd8273af>.
<urn:uuid:9a7d6a5a-2bfc-4219-b7cf-30c1f4cbeaa9> rdf:type bh:Task;
   bh:inputModel <urn:uuid:779486b3-b5a6-4a80-bb17-d9ad7657f5c4>;
   bh:precondition <urn:uuid:40fc4cc9-8e53-4e04-ad77-1d3361878a1b>;
   bh:outputModel <urn:uuid:23533149-2378-4af6-9f0c-c6ec4eb7bd67>;
```

```
   bh:postcondition <urn:uuid:3b8c377c-3e04-444d-b0ad-c106feb002ac>.
<urn:uuid:62c8eccd-ae32-44e5-b36c-be3f7a7f897f> rdf:type bh:Task;
   bh:inputModel <urn:uuid:79277958-a906-4ee5-a3dd-284bd75061a8>;
   bh:precondition <urn:uuid:62720a4a-b991-4a60-8d75-5ae5dc32297e>;
   bh:outputModel <urn:uuid:ec79f44d-0824-4c68-8111-b1a6cddc4245>;
   bh:postcondition <urn:uuid:d2bfb8dd-a181-4f92-b319-d48659e5da18>.
<urn:uuid:6bc65f81-9cdd-47e1-a1d6-7deb328610c0> rdf:type
bh:ExlusiveGateWay;
bh:inputModel <urn:uuid:8ae8c965-c90c-4000-8e24-93aed2b8a175>;
   bh:precondition <urn:uuid:8adaf9ca-3d5b-4b02-9835-4e6e0be50890>.
<urn:uuid:e4e4f0f1-4767-4a20-8cbf-4f39d07d97f0> rdf:type
bh:ExlusiveGateWay;
   bh:inputModel <urn:uuid:8ae8c965-c90c-4000-8e24-93aed2b8a175>;
   bh:precondition <urn:uuid:748ff7f4-c535-4a15-8f9b-3742c0a7bc96>.
<urn:uuid:47fbec46-7eb1-484c-87fc-7a0f198008ae> rdf:type bh:Transition;
   bh:from <urn:uuid:d074e4d6-c11b-439f-b58c-d5cf02ea2ba7>;
   bh:to <urn:uuid:6bc65f81-9cdd-47e1-a1d6-7deb328610c0>.
<urn:uuid:73354dbf-d56a-472a-a480-890bdd01d63b> rdf:type bh:Transition;
   bh:from <urn:uuid:6bc65f81-9cdd-47e1-a1d6-7deb328610c0>;
   bh:to <urn:uuid:50070bd9-3c44-4980-89d6-ec246d8162c5>.
<urn:uuid:646eb39d-d10b-4960-898a-1d9dacda82dd> rdf:type bh:Transition;
   bh:from <urn:uuid:6bc65f81-9cdd-47e1-a1d6-7deb328610c0>;
   bh:to <urn:uuid:e4e4f0f1-4767-4a20-8cbf-4f39d07d97f0>.
<urn:uuid:87407197-e313-400d-b328-598cbcab4807> rdf:type bh:Transition;
   bh:from <urn:uuid:e4e4f0f1-4767-4a20-8cbf-4f39d07d97f0>;
   bh:to <urn:uuid:9a7d6a5a-2bfc-4219-b7cf-30c1f4cbeaa9>.
<urn:uuid:89347158-f5a8-4b64-ae21-27491df4d97d> rdf:type bh:Transition;
   bh:from <urn:uuid:e4e4f0f1-4767-4a20-8cbf-4f39d07d97f0>;
   bh:to <urn:uuid:62c8eccd-ae32-44e5-b36c-be3f7a7f897f>.
<urn:uuid:e2af8f9f-fd92-4531-aa58-df5c9638ea3a> rdf:type bh:Transition;
   bh:from <urn:uuid:9a7d6a5a-2bfc-4219-b7cf-30c1f4cbeaa9>;
   bh:to <urn:uuid:65498537-6b14-43d1-831c-f334273b9808>.
<urn:uuid:7bf2b1e3-f2fb-4d40-997f-71df9996519b> rdf:type bh:Transition;
   bh:from <urn:uuid:62c8eccd-ae32-44e5-b36c-be3f7a7f897f>;
   bh:to <urn:uuid:65498537-6b14-43d1-831c-f334273b9808>.
<urn:uuid:e264fd6f-67d8-4190-91e8-01ed3754b9c9> rdf:type bh:Transition;
   bh:from <urn:uuid:50070bd9-3c44-4980-89d6-ec246d8162c5>;
   bh:to <urn:uuid:65498537-6b14-43d1-831c-f334273b9808>.
<urn:uuid:bdb20ef9-9902-4c14-8235-8e9894915adb> rdf:type bh:Transition;
   bh:from <urn:uuid:65498537-6b14-43d1-831c-f334273b9808>;
   bh:to <urn:uuid:793842e0-0e9c-403f-9b49-999ec6193389>.
<urn:uuid:58e36c16-33c6-4aae-a121-c82bb3d8784b> rdf:type bh:Transition;
   bh:from <urn:uuid:65498537-6b14-43d1-831c-f334273b9808>;
   bh:to <urn:uuid:6bc65f81-9cdd-47e1-a1d6-7deb328610c0>.
```

Listado 9.5: El modelo de comportamiento RDF para el tanque de agua.

En aras de la simplicidad, solo se muestra un extracto de los tres escenarios.

El RDF para el modelo de entrada que representa las dos válvulas se muestra en el Listado 9.6.

```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix bh: <http://www.bh.org/bh#> .
#
#
[] rdf:type bh:Model .
#
bh:vc bh:hasValue 0.
bh:vs bh:hasValue 10.
bh:val1 bh:hasValue 1.
bh:val2 bh:hasValue 1.
```

Listado 9.6: RDF para modelo de válvulas.

Un extracto de las condiciones que representan los diferentes escenarios se define en SHACL y se muestra en el Listado 9.7.

```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix bh: <http://www.bh.org/bh#> .
#
#
#
#This rule checks if the current volume vc is the same as the set point volume
vs.
bh:r1 a sh:NodeShape;
 sh:targetNode bh:vc ;
 sh:sparql [
  sh:select """
    ask
    Where{
       $this <http://www.bh.org/bh#hasValue> ?x .
       <http://www.bh.org/bh#vs> <http://www.bh.org/bh#hasValue> ?y.
       FILTER (?x=?y)
    }
    """ ;
 ].
```

Listado 9.7: un extracto de las formas SHACL para la primera condición previa de la puerta de enlace.

La tarea detrás del escenario de cerrar las dos válvulas se modela en el código SPARQL representado en el Listado 9.8.

```
PREFIX sh: <http://www.w3.org/ns/shacl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bh: <http://www.bh.org/bh#>
#
#
WITH  <urn:uuid:8ae8c965-c90c-4000-8e24-93aed2b8a175>
DELETE {
        <http://www.bh.org/bh#val1> <http://www.bh.org/bh#hasValue> 1.
        <http://www.bh.org/bh#val2> <http://www.bh.org/bh#hasValue> 1.
}
insert {
        <http://www.bh.org/bh#val1> <http://www.bh.org/bh#hasValue> 0.
        <http://www.bh.org/bh#val2> <http://www.bh.org/bh#hasValue> 0.
}
WHERE {}
```

Listado 9.8: Código SPARQL para la tarea de cerrar las dos válvulas.

La tarea de actualizar el volumen actual vc tiene en cuenta el estado de las válvulas y la ecuación vc(t) = vc(t-1)+k1*status_val1-k2*status_val2, se encuentra representada en el Listado 9.9. Para simplificar k1 y k2 representan los caudales de cada válvula, y por defecto toman el valor 1. Por tanto, la variación del volumen del tanque por unidad de tiempo es una unidad.

```
PREFIX sh: <http://www.w3.org/ns/shacl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bh: <http://www.bh.org/bh#>
#
#

with <urn:uuid:8ae8c965-c90c-4000-8e24-93aed2b8a175>
delete  {bh:vc bh:hasValue ?cv.}
insert  {bh:vc bh:hasValue ?nv.}
WHERE {
   bh:vc bh:hasValue ?cv.
   bh:val1 bh:hasValue ?vv1.
   bh:val2 bh:hasValue ?vv2.
   bind (<http://www.w3.org/2001/XMLSchema#int>(?vv1) as ?vvd1)
   bind (<http://www.w3.org/2001/XMLSchema#int>(?vv2) as ?vvd2)
   bind (<http://www.w3.org/2001/XMLSchema#int>(?cv) as ?cvd)
```

```
  bind( (?cvd+?vvd1-?vvd2) as ?nv)
  # vc(t) = vc(t-1)+k1*status_val1-k2*status_val2
};
```

Listado 9.9: SPARQL para actualizar el modelo del sistema físico.

La tarea realizad, encargada de actualizar el sistema físico, se modela mediante una ecuación simple: vc(t) = vc(t-1) + k1 * status_val1-k2 * status_val2, donde el tiempo es una variable discreta. vc(t) es el volumen actual en el instante t. vc(t-1) el volumen en el instante anterior (una unidad antes del instante t)

k1 y k2 son las cantidades de agua suministradas o extraídas por las válvulas val1 y val2, respectivamente, en una unidad de tiempo (caudal de val1 y val2), y status_val1 y status_val2 representan si las válvulas están abiertas o cerradas. En este ejemplo, un modelo lineal básico describe la dinámica del sistema. Sin embargo, el modelo puede ser reemplazado por uno más complejo en el que la ecuación diferencial sea de orden superior, manteniendo los datos de instantes anteriores en el modelo RDF y tratando la no linealidad de las válvulas.

9.6. **Conversión de constructos comportamentales de UML al enfoque propuesto**

El enfoque propuesto para el modelado del comportamiento proporciona un conjunto de elementos visuales o grafos con su correspondiente sintaxis en los lenguajes de la web semántica, RDF, SHACL, SPARQL o OWL.

Definición: Una entidad de procesamiento semántico es una entidad con la capacidad de recibir y enviar mensajes y razonar sobre ellos, los cuales se pueden representar como grafos.

UML ofrece una serie de elementos que sirven como un lenguaje para las etapas de especificación, análisis y diseño de sistemas. Sin embargo, cabe destacar que estos elementos no son directamente ejecutables, sino que se implementan mediante lenguajes de programación orientados a objetos de propósito general. Los diagramas esenciales se muestran en la Tabla 9.2.

Modelado	Diagramas UML usados tipos de Modelado
De utilización	Casos de Uso
Conceptual de Dominio	Clases
De Procesos	Actividad
De arquitectura	Componentes, Despliegue, Paquetes
De objetos dinámicos	Comunicación, Estructura compuesta, Resumen de interacción, Secuencia, Maquina de estados, Temporización
Estructural detallado	Clases Objetos

Tabla 9.2: Los diagramas UML básicos.

Diagrama de actividad:

El diagrama de actividades es una representación gráfica de un flujo de trabajo que comprende una serie de elementos, los cuales se encuentran detallados en la Tabla 9.3. Con el propósito de simplificar, en este enfoque se abordan las puertas de enlace exclusivas *(Gateway)* con solamente dos salidas posibles: sí o no. Aunque se enfoca principalmente en los elementos fundamentales para la elaboración de flujos, los demás elementos también pueden ser modelados utilizando el enfoque propuesto. Además, los diagramas de actividad incorporan elementos de control de flujo, como la sincronización, las decisiones y el control de concurrencia.

Los elementos visuales del UML para el aspecto comportamental se han reutilizado en el enfoque del trabajo desarrollado en esta sección, aunque la semántica puede variar para adaptarse al dominio ontológico.

OMG proporciona, entre otros, el BPMN, el diagrama de actividad y la máquina de estados para el aspecto comportamental, que se basan principalmente en grafos, pero no abordan la conceptualización del grafo y simplemente se tratan como diagramas.

El enfoque propuesto asume que todo es un grafo. Por lo tanto, las estructuras, los procesos y los comportamientos son grafos. Si se requiere incorporar un nuevo constructo para abordar un caso de uso específico en el futuro, simplemente se debe asignar un nombre como identificador o símbolo al constructo, junto con su semántica, y posteriormente, su implementación se añadirá al procesador del grafo de comportamiento.

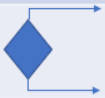
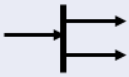
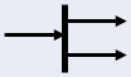
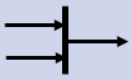
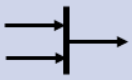
Constructo	Símbolo Uml	Semántica	Constructo Correspondiente en el enfoque
Acción		Representa un paso en una determinada actividad	
Nodo de decisión		Representa una rama condicional	
Flujos de control		Son conectores entre paso del diagrama	
Nodo inicial		Representa el inicio de un flujo	
Nodo terminal		Representa el final de un proceso	
Nodo bifurcación		Ejecución en paralelo	
Nodo Join		Sincronización con objetivo de espera de finalización de todos los aristas entrantes.	

Tabla 9.3: Constructos básicos del diagrama de actividades con mapeo de conceptos al enfoque propuesto.

Máquina de estados:

Los elementos fundamentales de una máquina de estados son los estados, las transiciones, las entradas y las salidas, tal como se ilustra en la Figura 9.17.

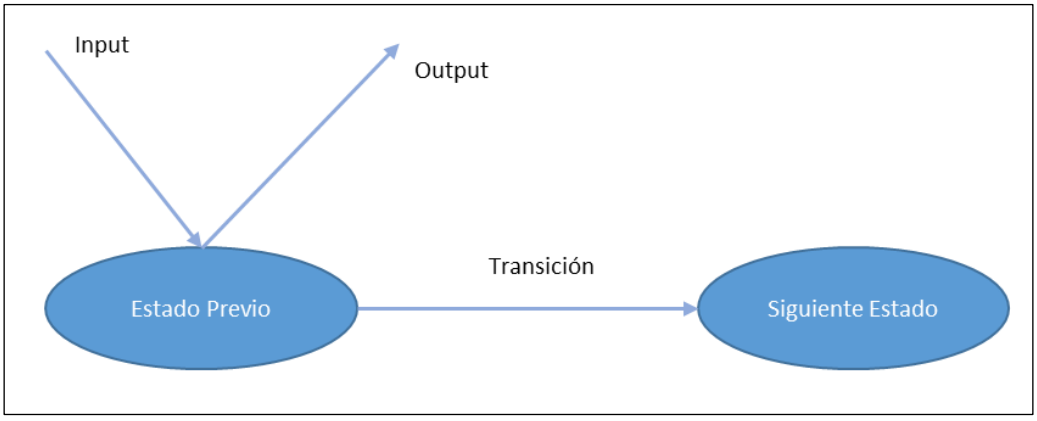


Figura 9.17: Constructos básicos de la máquina de estado.

El mapeo de conceptos entre el enfoque propuesto y la máquina de estados se ilustra en la Figura 9.18.

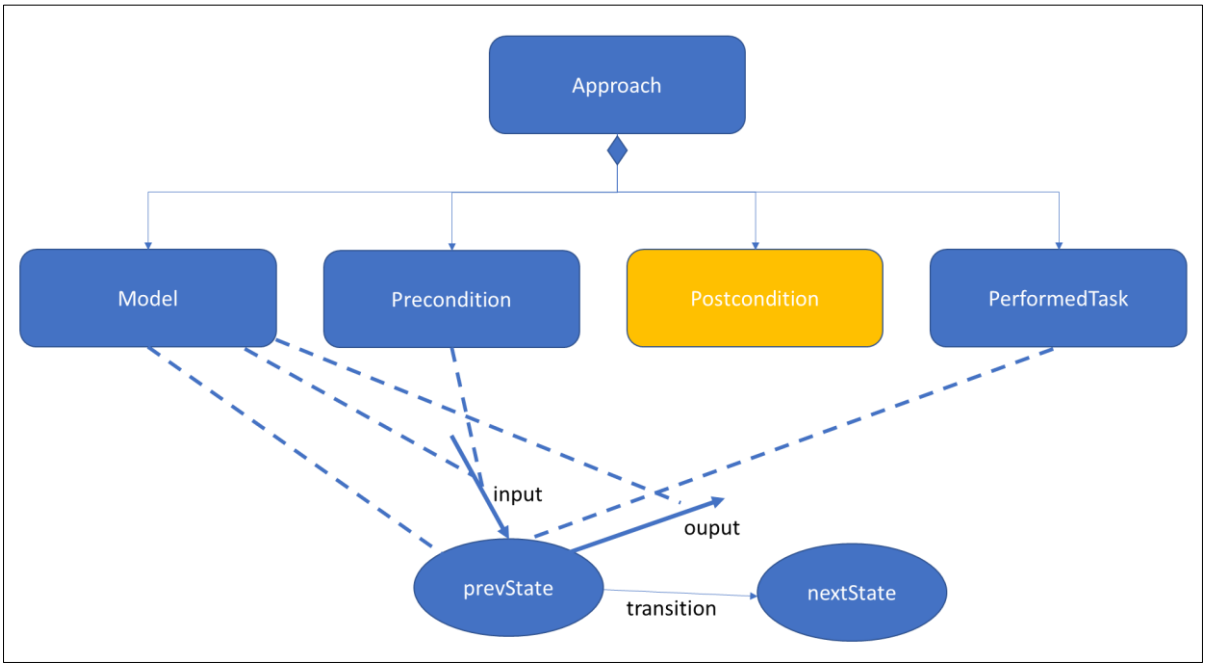


Figura 9.18: Constructos básicos de la máquina de estados con mapeo de conceptos al enfoque propuesto.

La Figura 9.19 puede representar el caso cuando el foco está en el nodo de estado, donde se espera una entrada, se realiza una tarea, se produce una salida y se realiza la transición al siguiente estado.

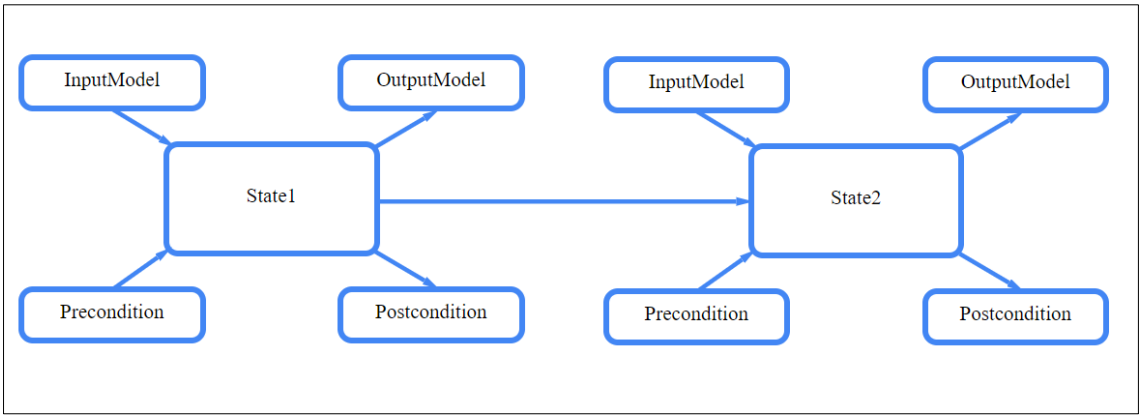


Figura 9.19: El flujo correspondiente para una máquina de estado donde el foco está en el nodo.

La Figura 9.20 puede representar el caso cuando el enfoque está en la transición que comienza desde un estado, se espera un modelo de entrada, se realiza una tarea, se produce una salida y se logra la transición al siguiente estado.

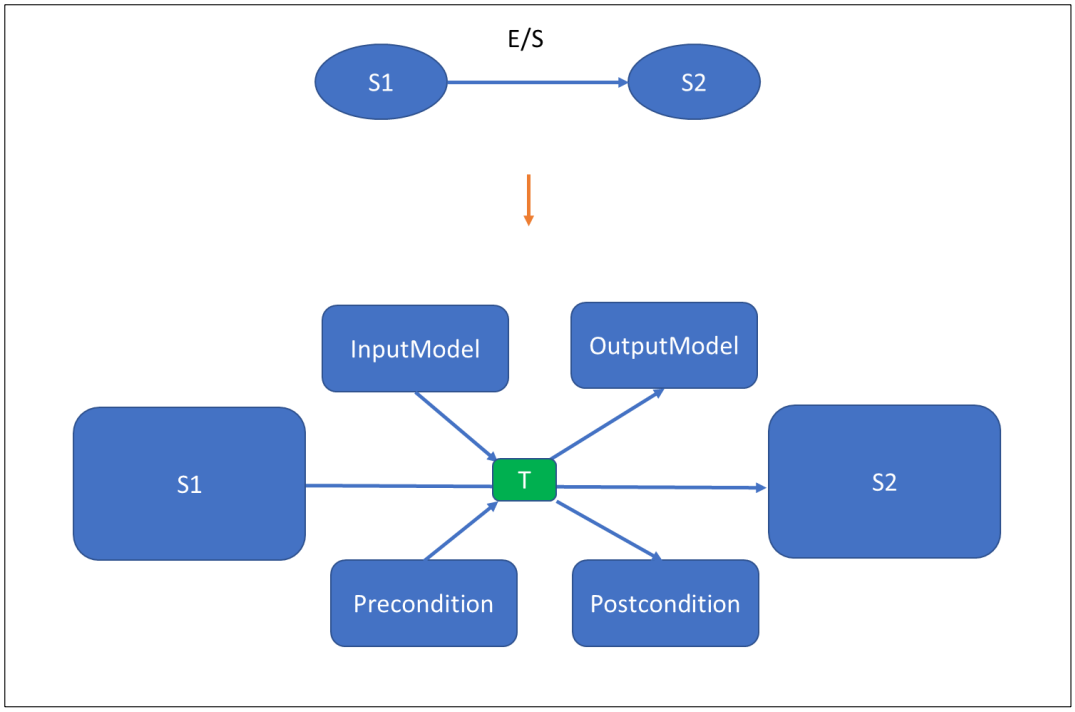


Figura 9.20: El flujo correspondiente para una máquina de estado donde el foco está en la transición.

Como en el enfoque propuesto, la transición tiene explícitamente su modelo de entrada, precondición, modelo de salida y postcondición, el comportamiento de la máquina de estados se puede implementar cuando la entrada y la salida se describen en la transición.

Si la entrada y la salida se describen en el estado, la tarea realizada se utiliza como estado; la entrada de la máquina de estado estará representada por (modelo de entrada, condición previa) y la salida por (modelo de salida, condición posterior).

Diagrama de interacción:

En UML, la interacción se representa principalmente mediante el envío y la recepción de mensajes, que pueden transportar parámetros entre dos componentes u objetos.

En el enfoque propuesto, los mensajes se modelan en función de un metamodelo específico de instancias y la capacidad de razonar sobre ellas. La estructura básica de la interacción se representa en la Figura 9.21 extraída de la plataforma desarrollada.

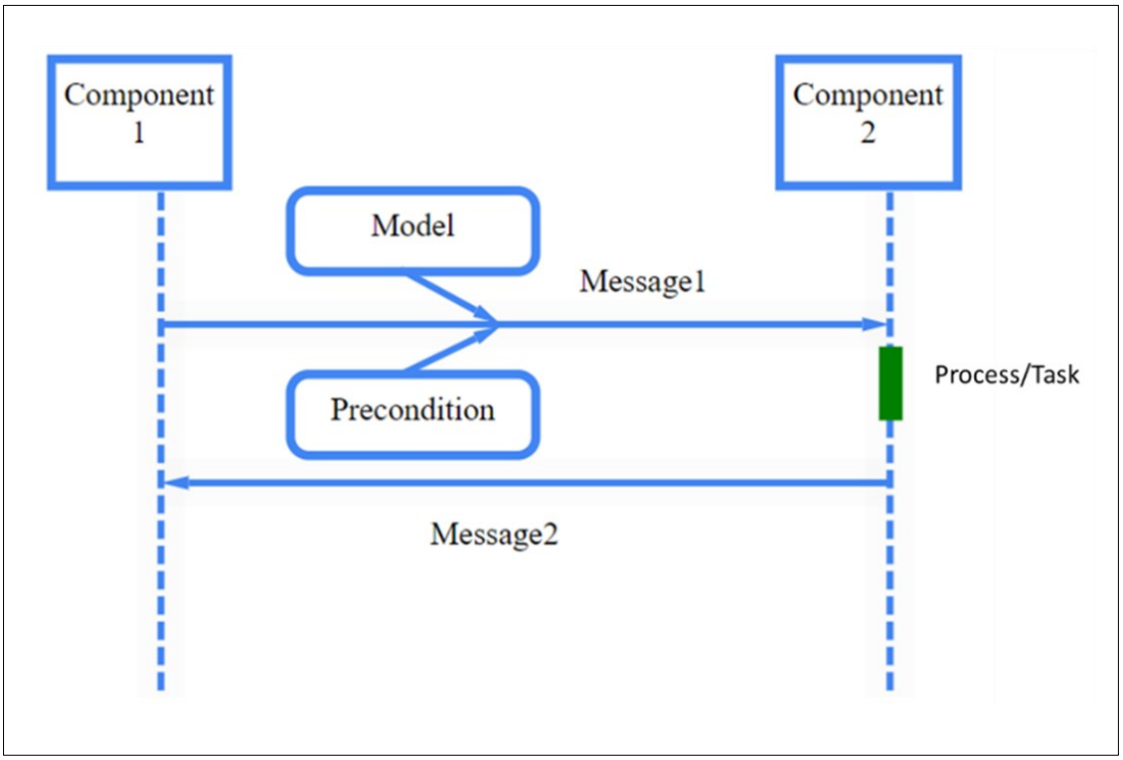


Figura 9.21: el modelo de interacción según el enfoque propuesto.

El grafo RDF que corresponde al diagrama de interacción de la Figura 9.21 se muestra en la Figura 9.22. La clase MP es la superclase que representa ambas clases: Mensaje y Proceso. El grafo de la Figura 9.22 es el metamodelo o RDF Schema del ejemplar representado en la Figura 9.21 .

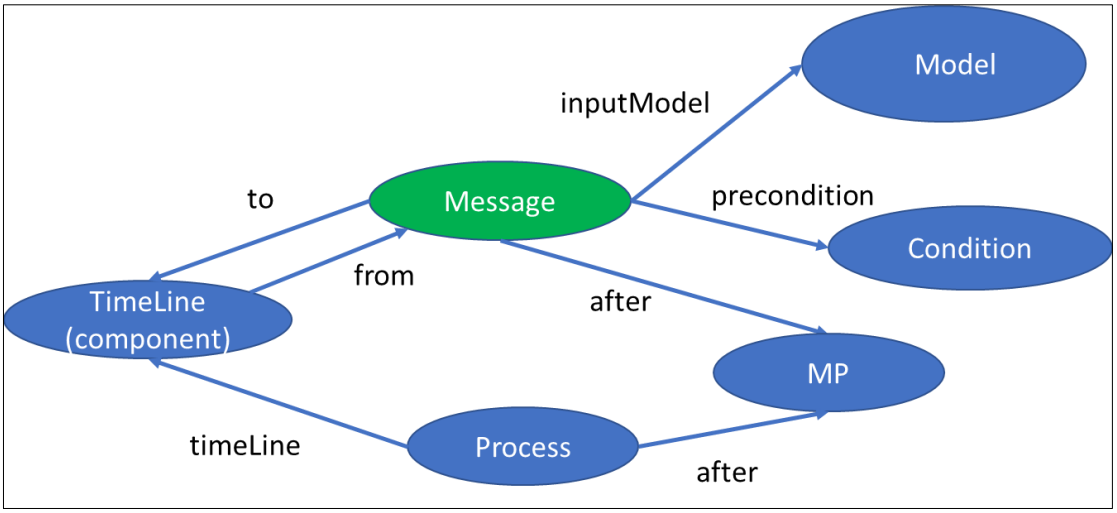


Figura 9.22: El grafo RDF correspondiente al diagrama de interacción de la Figura 9.23.

El RDFS correspondiente a los constructos en los que se basa el diagrama de interacción se muestra en el Listado 9.10.

```
@prefix bh: <http://www.bh.org/bh#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

bh:TimeLine rdf:type rdfs:Class.
bh:Process rdf:type rdfs:Class.
bh:Message rdf:type rdfs:Class.
bh:Model rdf:type rdfs:Class.
bh:Condition rdf:type rdfs:Class.
bh:Flow rdf:type rdfs:Class.


bh:from rdf:type rdf:Property;
        rdfs:domain bh:TimeLine;
        rdfs:range bh:Message.
bh:to rdf :type rdf:Property;
        rdfs:domain bh:Message;
        rdfs:range bh:TimeLine.

bh:name rdf:type rdf:Property;
        rdfs:domain bh:Message;
        rdfs:range xsd:string.
bh:inputModel rdf :type rdf:Property;
        rdfs:domain bh:Message;
        rdfs:range bh:Model.
bh:precondition rdf :type rdf:Property;
        rdfs:domain bh:Message;
        rdfs:range bh:Condition.
bh:hasFlow rdf :type rdf:Property;
        rdfs:domain bh:Process;
        rdfs:range bh:Flow.

bh:timeLine rdf :type rdf:Property;
        rdfs:domain bh:Process;
        rdfs:range bh:TimeLine.

bh:precondition rdf :type rdf:Property;
        rdfs:domain bh:Message;
            rdfs:range bh:Condition.
```

Listado 9.10: Esquema RDF de los constructos del diagrama de interacción.

El RDF correspondiente al componente 2 de la Figura 9.21 se muestra en el Listado 9.11.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix sh:   <http://www.w3.org/ns/shacl#> .
@prefix bh: <http://www.bh.org/bh#> .

<urn:uuid:b02c2e12-96ed-4499-9769-6960bf1b92ed> rdf:type bh:TimeLine .

<urn:uuid:6499c27a-fcc7-48d9-9080-2d75926d8149> rdf:type bh:Message ;
        bh:from <urn:uuid:182d3393-17f1-46bc-9f10-62cbdd3f6c41> ;
        bh:to <urn:uuid:b02c2e12-96ed-4499-9769-6960bf1b92ed> ;
        bh:name "Message1" ;
        bh:inputModel <urn:uuid:d6887daf-5180-4e25-ad1f-c332b9fc37d0> ;
        bh:precondition <urn:uuid:f35267d2-96b7-4fd7-8beb-0c57eb6bd363>  .

<urn:uuid:605cb512-357c-42db-a036-2482a3313d72> rdf:type bh:Process ;
        bh:timeLine <urn:uuid:b02c2e12-96ed-4499-9769-6960bf1b92ed> ;
        bh:hasFlow <urn:uuid:573451bb-bca5-41ee-a1e3-ace6af0f1583> ;
        bh:after <urn:uuid:6499c27a-fcc7-48d9-9080-2d75926d8149>   .

<urn:uuid:10fe2a7d-3657-4343-803e-cb5ae8f37376> rdf:type bh:Message ;
        bh:from <urn:uuid:b02c2e12-96ed-4499-9769-6960bf1b92ed> ;
        bh:to <urn:uuid:182d3393-17f1-46bc-9f10-62cbdd3f6c41> ;
        bh:name "Message2" ;
        bh:inputModel <urn:uuid:b3b28bbd-a72f-4fe7-8ea0-578cd0873cf4> ;
        bh:precondition <urn:uuid:f1072829-7429-488a-8310-fd6cf3300faa> ;
        bh:after <urn:uuid:605cb512-357c-42db-a036-2482a3313d72>   .
```

Listado 9.11: RDF correspondiente al componente 2 de la figura 10.21.

El despliegue de los procesadores de interacción para los diagramas de interacción, basados en el paradigma ontológico, se ha llevado a cabo de la siguiente manera:

- Primero, se despliegan los *endpoints* (procesadores de interacción), ya implementados para la semántica del enfoque propuesto, que atenderán mensajes, ejecutarán flujos y enviarán mensajes a los asistentes de mensajes que correspondan.
- En segundo lugar, en la fase de modelado/desarrollo/implementación, las líneas de tiempo se cargan en los *endpoints* correspondientes. Una vez cargados, los mensajes están listos para ser procesados; Por lo tanto, la ejecutabilidad es directa.

El esfuerzo del modelador se reduce a definir los modelos y condiciones junto con la organización de la semántica temporal en base a la semántica del procesador de interacción (metamodelo de ejecución de flujo).

La ejecución directa es realizada por el procesador del grafo de interacción correspondiente a cada línea de tiempo. El algoritmo del procesador de interacción se ilustra en el Listado 9.12. Asimismo, se ha implementado el algoritmo dentro de la

plataforma web desarrollada utilizando el framework Spring Boot y STOMP *(Simple Text Oriented Protocol)* para la transferencia de mensajes.

```
function attendMessage(BHMessage msg) {
   # The elements following the currently received message until
   # the following message to be received are executed.
   executeNextElements(msg);
}


function executeNextElements(BHMessage msg){
   # The RDF of the message, defined in the interaction structure, is obtained.
   String msgRdf = msg.getMessageContent();
   # From the RDF of the message, it is obtained to which timeline it is addressed, which
is a graph.
   String tlineUri = getTimeLineUri(msgRdf);
   # The next element to the current one is obtained, which can be either a message or
a process.
   BHNext next  = getNextEltIdFromMessg(msg.getMessageId(),tlineUri);
   # While there exist items to deal with, following the current one, it stays in the loop.
   while(next!=null){
      # The type of the next element is obtained.
      String type = next.getType();
      # If it is a process
      if(type == "Process"){
         # The process identifier to be executed is obtained.
         String flowId = next.getEltId();
        # The graph of the flow representing the process is obtained.
         Model m = BhUtils.getGraphModel(flowId);
         # A flow is created.
         Flow flw = new Flow();
         # The object m is assigned to the model attribute of the flw object.
         flw.setModel(m);
         # The process flow graph implementing the concepts of the proposed approach
is created.
         BHGraph bhgrf = flw.createBhGraph();
         # The flow is executed.
         bhgrf.executeFlow();
         # The next element to be executed is found.
         next  = getNextEltIdFromMessg(next.getMsgId(),next.getTo());
      }
      // si es un mensaje
       # If it is a message.
      if(type=="Message"){
         # A message object is created whose identifier is that of the next object
         # and whose content mainly is the message destination.
```

```
        BHMessage msgn = new BHMessage();
        msgn.setMessageId(next.getMsgId());
        String msgContent = createRDFMsg(next.getMsgId(),next.getTo());
        msgn.setMessageContent(msgContent);

        # The message is sent through the corresponding transport layer.
        sendMessage(msgn);
        # The next element to the current one is obtained.
        next  = getNextEltIdFromMessg(next.getMsgId(),next.getTo());
      }
    }
}
```

Listado 9.12: Algoritmo de atención de mensajes.

9.7. **Evaluación del enfoque**

" CGMES *(Common Grid Model Exchange Specification)* es una especificación técnica (TS) de IEC basada en la familia de estándares IEC CIM. Fue desarrollado para cumplir con los requisitos para los intercambios de datos entre TSOs en las áreas de desarrollo del sistema y operación del sistema" [43] [19]. ENTSO-E (La Red Europea de Operadores de Sistemas de Transmisión) proporciona la especificación del estándar CGMES, que consta de un conjunto de documentos. Los documentos clave de interés para este trabajo son el archivo XSD para definir el modelo de reglas; archivos XML que definen las reglas QoCDC *(Quality of CGMES Datasets and Calculations)* [78] para cada nivel de validación y especifican el formato los informes de validación; archivos RDFS y OCL para los distintos modelos correspondientes a los diferentes perfiles [Equipment (EQ), Diagram layout (DL), Dynamics (DY), Boundary equipment (BD-EQ), Boundary topology (BD-TP), Operation, cortocircuito, Geographical Location (GL), State variables (SV), Steady State Hypothesis (SSH)hipótesis de estado estacionario (SSH), y topología (TP)]

En este trabajo se considera que los elementos proporcionados por el estándar CIM (CGMES) pueden ser utilizados para evaluar el enfoque propuesto. El CIM y especialmente CGMES han creado una ontología en forma de grafos RDFS para la parte estructural del estándar junto con un conjunto de reglas clasificadas en 8 niveles y especificadas en lenguaje natural y OCL. Este estándar permite a los TSOs crear modelos que se ajusten a unos esquemas y unas reglas definidas. Además, el estándar CGMES especifica que el proceso de validación se realiza en secuencia desde el nivel 1 hasta el nivel 8. Si un modelo no cumple con las reglas de cierto nivel, se devuelve un informe de validación y no se realiza el siguiente nivel de validación. Por lo tanto, el proceso de validación especificado por el CGMES es un flujo de tareas de validación contra las reglas del estándar sobre modelos de datos, que son esencialmente ontologías. Además, este enfoque ha proporcionado una solución desarrollada utilizando los estándares W3C para la validación de modelos CIM (CGMES), tanto para los aspectos estructurales como de comportamiento.

Todos los estándares son un conjunto de conceptos, relaciones, reglas y flujos de transformaciones que se pueden representar con grafos u ontologías para realizar la tarea de razonamiento. Por lo tanto, se podría haber elegido cualquier estándar de otro dominio y aplicarle el mismo enfoque para la validación contra las reglas del estándar.

Cualquier proceso de negocio puede ser considerado como un flujo. Por lo tanto, teniendo un procesador de flujo para un lenguaje de modelado de flujo que admita el razonamiento se consigue la ejecutabilidad directa y, por consiguiente, se ahorra esfuerzo en las diferentes fases del proceso de desarrollo (especificación de requisitos, implementación, pruebas y tareas posteriores a la entrega). Como resultado se consiguen mejoras de transparencia (*Glass box*), escalabilidad, entre otras.

La Figura 9.24, extraída de la plataforma desarrollada, muestra el modelado de flujo del proceso de validación de los ocho niveles del estándar CGMES. La condición previa de cada nivel corresponde a las reglas especificadas por el estándar para ese nivel y se modela en SHACL. El modelo de entrada es el modelo necesario para un nivel dado de validación. El motivo o razón contiene el informe de validación de cada nivel.

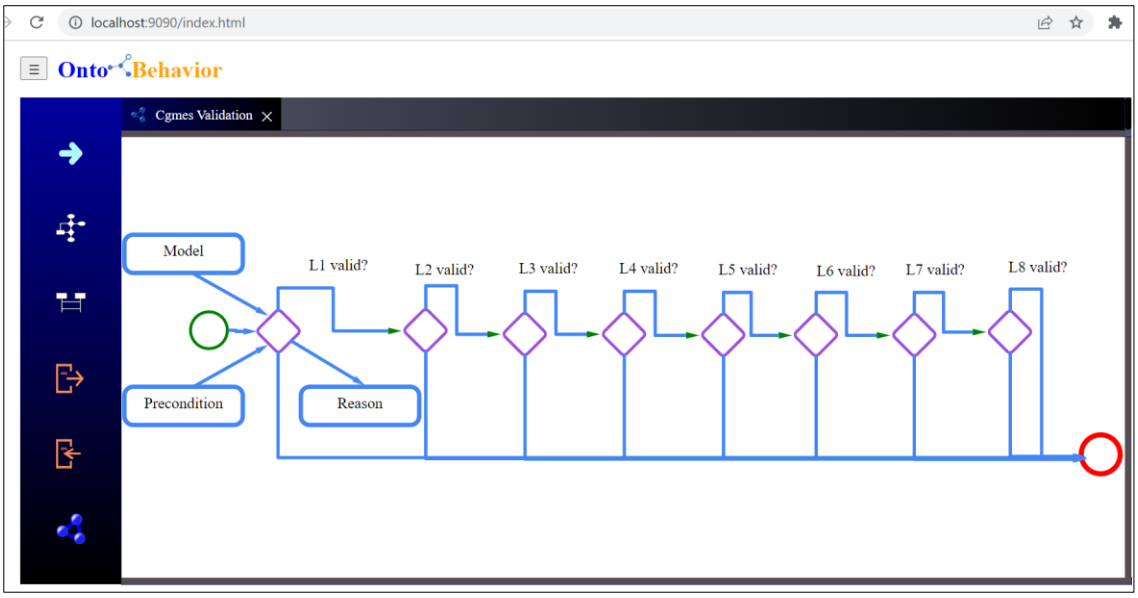


Figura 9.24: Flujo para el proceso de los ocho niveles de validación del estándar CGMES.

Se ha desarrollado una plataforma cuya arquitectura se muestra en la Figura 9.25. La plataforma ha sido desarrollada en Java, concretamente como un proyecto Spring Boot utilizando la librería JENA para la gestión de grafos RDF y el *endpoint Fuseki* para almacenar los grafos RDF que representan los modelos.

La aplicación consta principalmente de un procesador de flujo para la ejecucion de los procesos especificados, y un procesador de interacción para la comunicación e interacción entre diferentes componentes que pueden estar alojados en diferentes

máquinas. Para el uso de estos dos procesadores se han implementado un conjunto de servicios API REST para la carga y ejecución de grafos de flujo e interacción. Además, se han desarrollado librerias Javascript para crear, ejecutar y monitorizar la ejecución de estos diagramas desde un navegador web. La capa de integración de la plataforma se puede construir sobre los servicios API REST que exponen los procesadores de flujo e interacción.

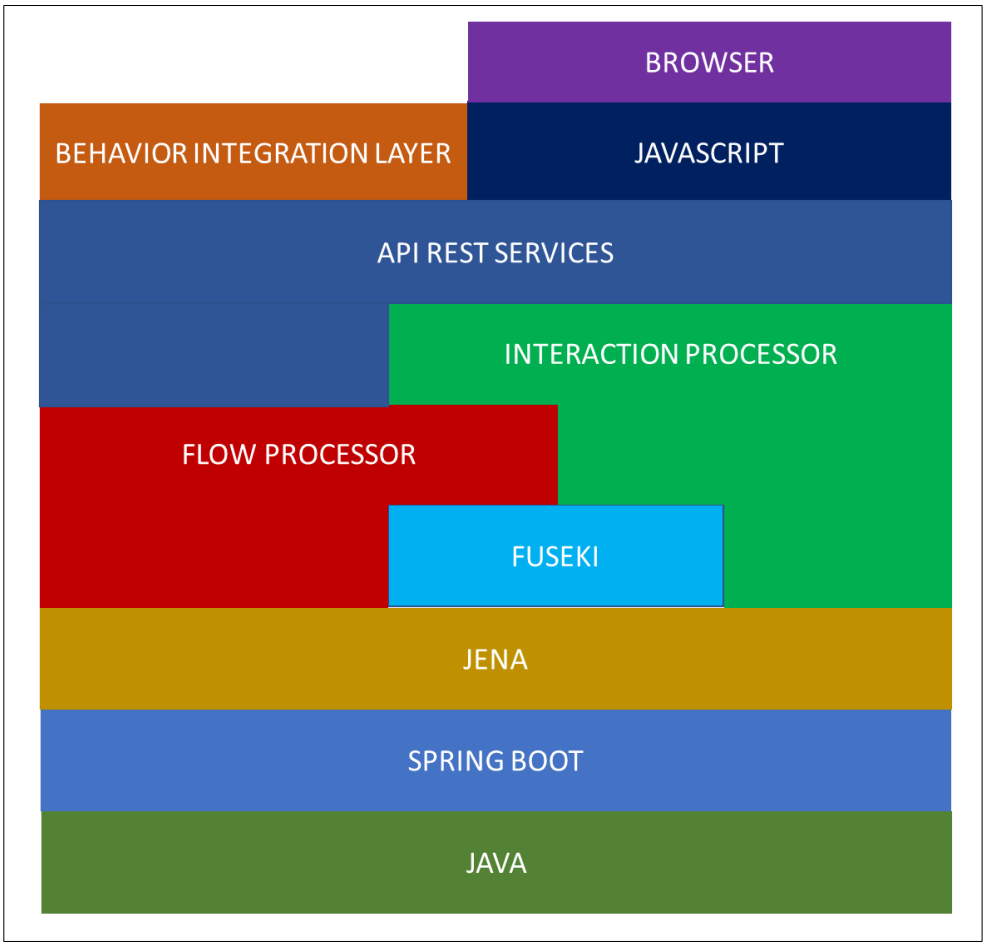


Figura 9.25: La arquitectura de la plataforma de procesadores de flujo e interacción.

La plataforma web para el modelado de flujo se muestra en la Figura 9.26. Se proporciona una barra de herramientas para crear constructos visuales relacionadas con un flujo: PerformedTask, StartNode, EndNode, GatewayNode y WaitingNode. El modelo RDF de cada elemento de flujo se puede editar en turtle [79] o RDF/XML, como se muestra en la Figura 9.26 para los editores de modelo de entrada y condiciones previas. En la Figura 9.27 se muestra la interfaz de la plataforma para la gestión de los diagramas de interacción con enfoque ontológico. En esta interfaz se presenta una barra de herramientas que permite la creación de componentes, como líneas de tiempo, y ofrece constructos visuales para la creación de mensajes, procesos y otros elementos relacionados.

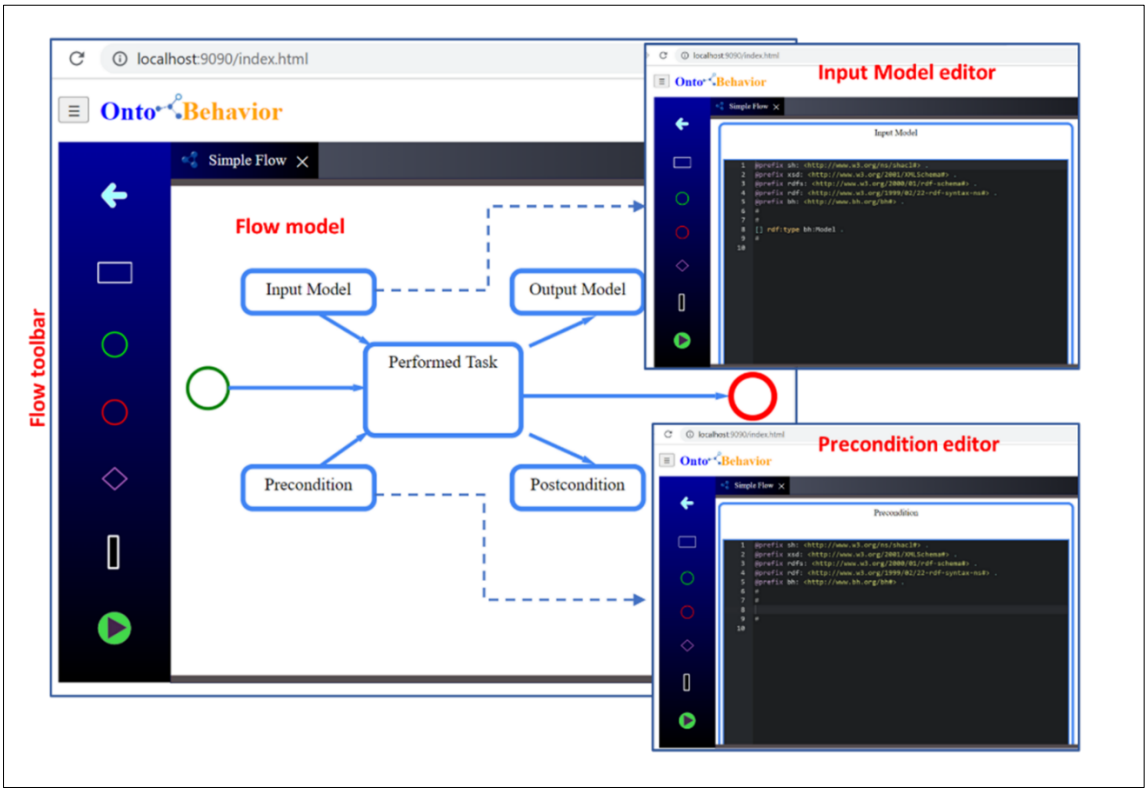


Figura 9.26: Interfaz de la plataforma de gestión del Comportamiento (Flujo).

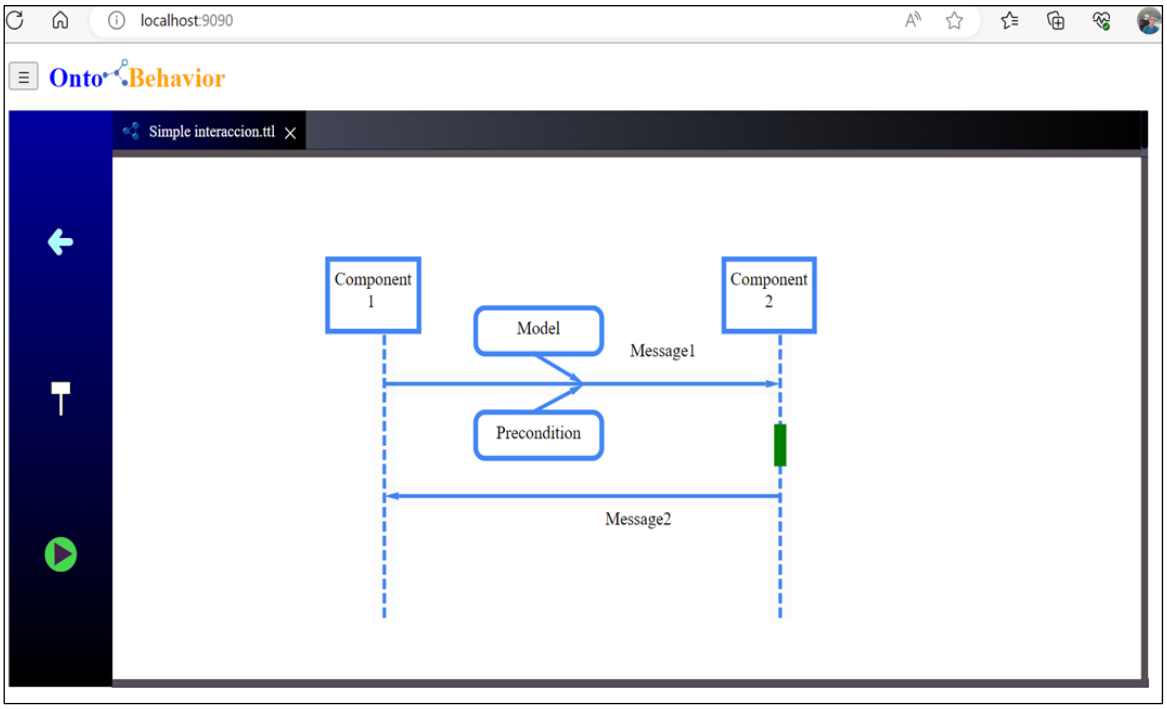


Figura 9.27: Interfaz de la plataforma de gestión del Comportamiento (Interacción).

9.8. Limitaciones

En este trabajo se han considerado las siguientes limitaciones: (i) La versión del estándar CGMES sobre la que se evaluó el enfoque es la 2.4.15. (ii) Solo se han utilizado tecnologías de la web semántica para el modelado ontológico. (iii) El lenguaje de modelado de las reglas para las condiciones previas y posteriores se ha definido solo en SHACL. (iv) Se pueden agregar otros constructos al enfoque, como el concepto de bucle en el diagrama de interacción. (v) Solo se ha limitado a mapear el paradigma orientado a objetos (UML) para mapear conceptos fundamentales de comportamiento al enfoque propuesto, como diagrama de actividad, máquina de estado y diagrama de interacción.

9.9. Conclusiones y trabajos futuros

Para el proceso de ingeniería de software con ontologías para la especificación de comportamiento, se ha proporcionado un enfoque con su correspondiente lenguaje visual para la especificación, diseño, implementación y prueba de requisitos. El enfoque propuesto considera un proceso de ingeniería de software como un proceso de modelado evolutivo. La fase de prueba se puede tratar como un proceso de validación, de un determinado modelo contra una población de instancias de prueba. El paradigma de modelado de comportamiento consiste en tareas realizadas cuando sus condiciones se cumplen frente a su modelo de entrada, considerando un modelo como un conjunto de conceptos y relaciones asociadas a la tarea. Los dos aspectos del comportamiento que se han abordado e implementado son el flujo y la interacción. Se ha realizado una transformación de los diagramas de comportamiento UML fundamentales en el enfoque propuesto, incluido el diagrama de actividad, el diagrama de interacción y el diagrama de máquina de estado. En este trabajo, solo se han abordado los constructos más comunes para el comportamiento; Para la lógica temporal del comportamiento no se han proporcionado constructos con su correspondiente sintaxis visual en RDF ni su semántica. Sin embargo, incluso en ausencia de estos constructos, el modelado de la lógica temporal del comportamiento se puede abordar con el enfoque proporcionado en el trabajo de esta sección, utilizando el modelo de entrada y la condición previa. En caso de que se necesite un nuevo constructo para cubrir un caso de uso específico en el futuro, solo es necesario dar un nombre (identificador o símbolo) al constructo y su semántica cuya implementación se agregará al procesador de grafos del comportamiento. Como líneas de investigación futuras, se proponen el desarrollo del diagrama temporal, la aplicación de la minería de procesos con semántica, la mejora de los procesos y la aplicación del enfoque propuesto en el ámbito de la seguridad. La evaluación de este enfoque ha sido realizada en el dominio de las compañías eléctricas y sistemas de potencia, con el propósito de modelar y lograr la ejecución directa del proceso de validación de CGMES.

10. **Un enfoque basado en el modelado para el ciclo de vida del desarrollo del software**

10.1. **Resumen**

Hoy en día, la especificación de requisitos se realiza de acuerdo con estándares que son principalmente modelos, y los pasos restantes del proceso de desarrollo se realizan a través de Ingeniería Dirigida por Modelos (MDE). Sin embargo, existe una brecha entre las dos etapas, lo que lleva a problemas de trazabilidad y consistencia, especialmente en la transición entre la especificación de requisitos y las fases posteriores. Dado que los modelos pueden ser directamente ejecutables, el enfoque propuesto en esta sección aborda todas las fases de ingeniería de software con el paradigma de modelado (MDE). Por lo tanto, el proceso de software comienza con la incepción de un modelo inicial que evoluciona a lo largo de las transformaciones. Por lo tanto, el modelado se vuelve omnipresente en todas las fases del ciclo de vida del proceso de desarrollo de software ya que el modelado tiene la capacidad de representar un análisis exhaustivo de las propiedades y el comportamiento inherentes al sistema, y puede representar una especificación detallada acerca de cómo se llevará a cabo tanto la construcción estructural como el comportamiento del sistema.

Se han utilizado las reglas basadas en texto del CGMES *(Common Grid Model Exchange Standard)*, como requisitos para evaluar enfoque propuesto. Para obtener modelos ejecutables se han utilizado RDF y SHACL. El trabajo de esta sección está dirigido principalmente a la comunidad de ingeniería de software, donde existe la necesidad de optimizar la productividad, la agilidad, la calidad, la escalabilidad, la fiabilidad y el esfuerzo del proceso de desarrollo. Además de abordar el proceso de software con un enfoque de modelado, se resuelve el problema de proceso de validación de los modelos en el ámbito de las compañías eléctricas y sistemas de potencia.

10.2. **Introducción**

Estándares como IEC 12207 [80] son marcos establecidos para el ciclo de vida del desarrollo de software, con el objetivo de crear un ciclo de vida personalizado que se adapte a proyectos específicos. Por otro lado, existe un estándar para la especificación de requisitos [81], que aborda las primeras etapas de la ingeniería de software. Según [48], la Ingeniería Dirigida por Modelos (MDE) es una metodología que utiliza modelos, notaciones y reglas de transformación para elevar el nivel de abstracción de un sistema, con el fin de mejorar la productividad. La principal limitación de esta definición radica en su objetivo, que es aumentar el nivel de abstracción. MDE se basa en el principio fundamental de que "todo es un modelo" [82]. Tal como se menciona en [50], MDE es un paradigma basado en modelos, cuyos elementos clave son modelos, metamodelos y transformaciones de modelos.

Independientemente del modelo de ciclo de vida adoptado (como el modelo en cascada, modelo en forma de V, modelo Iterativo, modelo en espiral, modelo *Big Bang*, modelo ágil, entre otros), un ciclo de vida básicamente consta de la especificación de requisitos, el diseño, la implementación, las pruebas y las actividades posteriores a la entrega. Tradicionalmente, las etapas iniciales se han tratado como una entidad independiente a través de la ingeniería de requisitos, mientras que MDE

ha abarcado las etapas restantes. Sin embargo, existe una brecha entre todas las etapas, y particularmente entre la especificación de requisitos y las etapas posteriores, lo cual es de suma importancia, ya que puede generar problemas de trazabilidad y consistencia entre las diferentes fases del desarrollo de software.

En el contexto del proceso de especificación de requisitos, resulta evidente que existen dos actores de gran relevancia: el experto del dominio y el experto en modelado. Sin embargo, se observa una brecha significativa entre ellos en términos de la utilización de un lenguaje común para describir de manera directamente ejecutable las necesidades del experto del dominio. Además, es importante destacar que actualmente los expertos del dominio carecen de formación en los conceptos fundamentales del Ingeniería dirigida por modelos (MDE).

Esta situación plantea varios desafíos en el proceso de especificación de requisitos. En primer lugar, la falta de un lenguaje compartido dificulta la comunicación efectiva entre el experto del dominio y el experto en modelado. Esta falta de entendimiento mutuo puede conducir a malentendidos, errores de interpretación y, en última instancia, a la generación de modelos inadecuados o que no satisfacen las necesidades especificadas en los requisitos.

El enfoque propuesto surge de la observación de que es posible realizar las diferentes etapas del proceso a través de un enfoque holístico de modelado en relación con el ciclo de vida de la ingeniería de software, dado que el modelado es un factor transversal a todas las etapas de este. El modelado implica la identificación de los conceptos o elementos del dominio, las relaciones entre ellos, las reglas asociadas al dominio, las transformaciones en el modelo y la comunicación entre los elementos. El modelado está presente en todas las fases del ciclo de vida y contribuye a minimizar el esfuerzo de transición entre una etapa y otra, permitiendo una transición más fluida y transparente, como se muestra en la Figura 10.1.

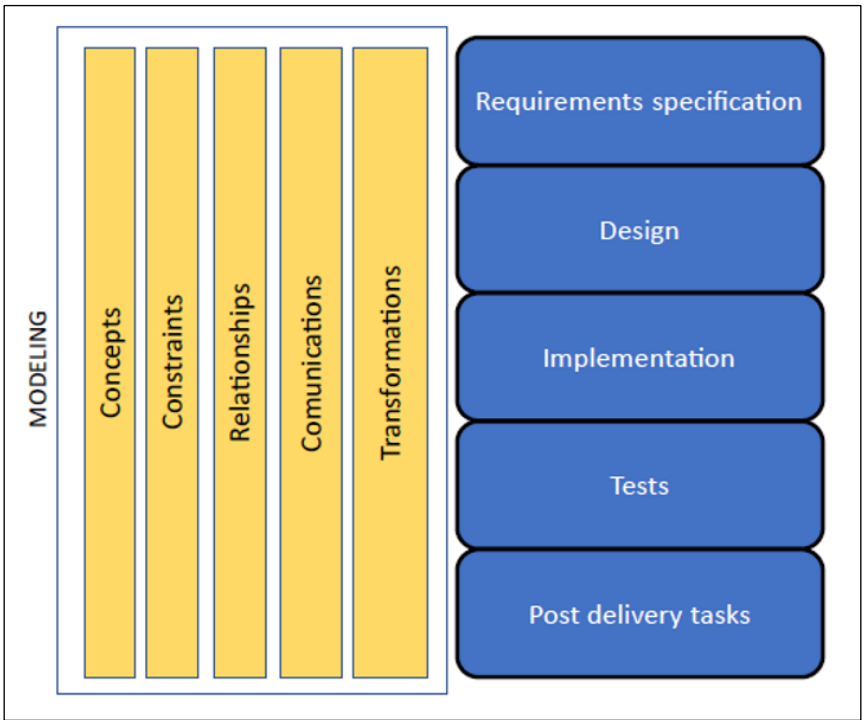


Figura 10.1: La transversalidad del modelado en el ciclo de vida del software.

Otra característica fundamental del enfoque propuesto es que el modelo es directamente ejecutable, a diferencia de otros enfoques donde los modelos requieren transformaciones para lograr la ejecutabilidad del artefacto inicial, como se ilustra en la Figura 10.2.

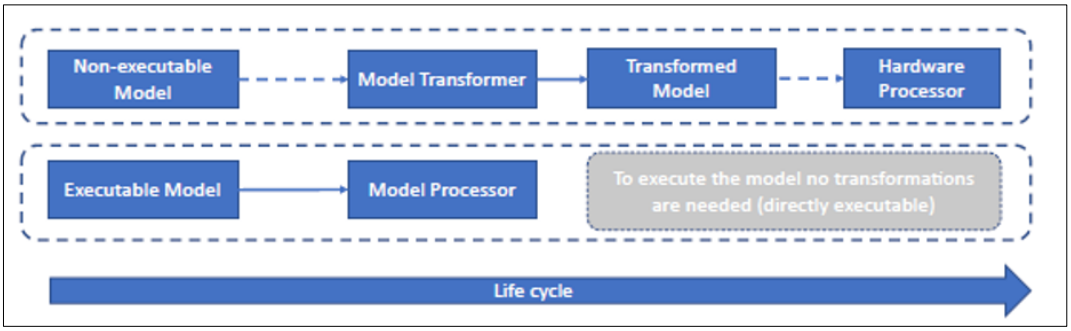


Figura 10.2: El enfoque de modelado directamente ejecutable VS. al enfoque de transformación de modelado.

Para el enfoque propuesto, se han empleado dos tecnologías: (i) RDF, una tecnología de la web semántica que utiliza identificadores únicos para representar todos los elementos como recursos. Los modelos se construyen desde una perspectiva de recursos y predicados, donde el conocimiento se representa de forma elemental mediante tripletas en la forma (sujeto, predicado, objeto); por lo tanto, los modelos se representan como grafos. (ii) SHACL, un lenguaje de validación estándar del W3C que permite validar restricciones en grafos de datos RDF; además, SHACL también puede utilizarse como un lenguaje de modelado.

La evaluación del enfoque propuesto se ha realizado en el dominio de las compañías eléctricas y los sistemas de potencia para la interoperabilidad con el estándar CIM en dos etapas: (i) Modelado de las reglas estándar de CGMES para la interoperabilidad entre las partes interesadas de la Red Europea de Operadores de Redes de Transporte de Electricidad (ENTSOE). La norma consiste en una serie de reglas definidas en un lenguaje natural considerado en este trabajo como una especificación de requisitos. Con el enfoque (modelado) que se propone en este trabajo, todas las reglas han sido modeladas/implementadas usando SHACL. (ii) Desarrollo de una aplicación basada en la arquitectura MVC *(Model View Controller)* con microservicios Spring para validar los ocho niveles de validación de la especificación CGMES. El modelado de reglas CGMES representa el componente modelo de la arquitectura MVC.

La audiencia objetivo de este trabajo es la comunidad de ingeniería de software y la comunidad de ingeniería eléctrica, específicamente en áreas donde se utiliza el formato IEC *(International Electrotechnical Commission)* CIM *(Common Information Model)*, y grupos de trabajo involucrados en la especificación de un estándar.

Este trabajo tiene tres contribuciones innovadoras. La primera contribución es la naturaleza transversal del modelado en el ciclo de vida del desarrollo de software. La segunda contribución es el modelado con un enfoque multiparadigma (repositorio de

paradigmas, es decir, Orientación a objetos y ontologías). La tercera contribución es el resultado de evaluar el enfoque en el dominio de las compañías eléctricas y sistemas de potencia mediante el modelado de todas las reglas del estándar CGMES. Por lo tanto, la comunidad de los interesados en el dominio de los sistemas de potencia ahorrará el esfuerzo de codificar las reglas, y pueden adoptar este enfoque para modelar sus propias reglas.

Este trabajo se divide en cinco partes. En la parte 10.3, se presenta una revisión bibliográfica del estado del arte. En la parte 10.4, se describe el enfoque propuesto. En la parte 10.5 describe la evaluación del nuevo enfoque. Finalmente, la parte 10.6 contiene las conclusiones y los trabajos futuros.

10.3. **Revisión del estado del arte**

En el ámbito de este trabajo se han considerado, en el contexto del desarrollo de software, dos paradigmas principales en MDE: el orientado a objetos utilizando el lenguaje de modelado unificado (UML) con el lenguaje de restricción de objetos (OCL), y el ontológico utilizando RDFS, OWL, SHACL, SPARQL. Sin embargo, la mayoría de los trabajos encontrados en la revisión bibliográfica en el estado del arte utiliza MDE con el paradigma orientado a objetos.

La revisión bibliográfica en el contexto de este trabajo se ha realizado en términos de cómo se teje o se hace trasversal MDE en el ciclo de vida de desarrollo de software con respecto a los procesos de modelado y validación. Esta revisión se ha organizado en tres grupos correspondientes a las tres áreas fundamentales de este trabajo: MDE con un enfoque OO (UML), MDE con un enfoque ontológico (RDF/ SHACL / SPARQL) y el ciclo de vida del desarrollo de software.

10.3.1 **Revisión bibliográfica en el contexto MDE con el enfoque Orientado a Objetos**

En la referencia [83], los autores han realizado una contribución en forma de desarrollo de un lenguaje específico de dominio (DSL) textual. Este lenguaje se utiliza para especificar las interconexiones de los objetos de Internet de las Cosas (IoT), proporcionando una representación cercana a los casos de uso definidos en la ingeniería de software. El lenguaje proporciona a los usuarios, sin conocimientos de programación, una forma de conectar sus objetos. En la referencia [84] [8], los autores han llevado a cabo el desarrollo de una taxonomía enfocada en la verificación de consistencia y la corrección de inconsistencias, junto con un marco dirigido a desarrolladores y partes interesadas. Se ha realizado una clasificación exhaustiva en su revisión bibliográfica del estado actual del arte en la gestión de la consistencia. En [85] [9], los autores han desarrollado un modelo para los aspectos estáticos y dinámicos del sistema aplicado al dominio militar. El comportamiento se ha modelado con el formalismo de la máquina de estados. Los aspectos estáticos se han realizado a través de la creación de perfiles. Los autores también utilizaron un conjunto de técnicas de transformación para reducir la heterogeneidad del modelo. El estudio realizado en el trabajo [86] demuestra que en la práctica del MDE, no se utiliza de manera continua

desde el diseño hasta la implementación. Según los resultados de esta encuesta, los autores concluyen que es preferible comenzar con el modelado del comportamiento, empleando una máquina de estados y el diagrama de secuencia, en lugar de enfocarse únicamente en el modelado estructural. En la referencia [29], los autores han aplicado la lógica de predicados para transformar UML/OCL con el objetivo de validar modelos UML. Mediante esta metodología, se logra reducir el nivel de abstracción hacia la lógica de primer orden (FOL) con el propósito de facilitar la validación. En [27], se presenta un grupo de estrategias de automatización para reglas OCL. La evolución del metamodelo tiene un impacto significativo en las reglas de OCL. Este trabajo presenta un enfoque semiautomático a través de una estrategia y una técnica para la evolución de las reglas OCL con respecto a los cambios en el diagrama de clases. En [28], se ha desarrollado una herramienta basada en la web utilizando un lenguaje fácil de usar en un nivel superior de abstracción. Este lenguaje permite definir reglas, por lo que no es necesario un conocimiento detallado de la sintaxis OCL, obteniendo un mayor nivel de abstracción. En [31], se muestra una conversión del modelo UML/OCL al paradigma de programación de lógica de restricciones (CLP) para permitir la validación, el razonamiento y la verificación. Sólo se han considerado los invariantes de OCL. Se ha desarrollado un nuevo marco para automatizar la transformación entre el diagrama de clases y la especificación lógica (formula). Con un razonador, la satisfacibilidad se determina considerando ciertos límites, es decir, si se puede crear un modelo de instancia.

10.3.2 **Revisión bibliográfica en el contexto de MDE con el enfoque ontológico**

El trabajo presentado en [41] representa uno de los primeros intentos de utilizar SHACL para modelar y validar los modelos CGMES. Sin embargo, dicho trabajo se enfoca exclusivamente en la validación del esquema RDF, sin abordar las reglas OCL ni las reglas del estándar CGMES basadas en texto. El dominio específico de este trabajo se centra en el intercambio de datos entre las partes involucradas en el ámbito de las compañías eléctricas. En la referencia [66], se lleva a cabo una revisión bibliográfica sistemática sobre los enfoques y herramientas de modelado utilizados en sistemas empotrados.

En [39], se crea una ontología colaborativa para flujos de información de diferentes fuentes públicas y privadas en sensores ambientales. Se realiza mapeo semántico y enriquecimiento de datos. Los datos se validan utilizando SHACL definiendo reglas de colaboración. Para ello, los autores han desarrollado un marco llamado LSane. En [38], los autores han transformado la Iniciativa de Modelado de Información Clínica (CIMI) en ontologías, utilizando SHACL para modelar y definir reglas con fines de validación. Este hecho dificulta la integración con otros enfoques como UML (es como modelar simplemente usando OCL). En [37], se utiliza una API de experiencia estándar (XAPI) de código abierto como punto de partida para el dominio de e-learning en la recopilación de datos de aprendizaje. Los autores han transformado el modelo estándar XAPI en una ontología, y utilizan SHACL para validar las restricciones definidas en este estándar XAPI. Se ha utilizado el formato JSON para serializar datos. En [34], se utiliza SWRL *(Semantic Web Rule Language)* para el

intercambio de datos aplicado al dominio del agua utilizando WDTF *(Water Data Transfer Format)* para la definición de restricciones de integridad y utilizando OWL para modelar este dominio. A pesar de que OWL y SWRL consideran OWA *(Open-World Assumption)*, los autores han utilizado OWL/SWRL para la validación del modelo. La validación se realiza transformando axiomas en consultas en el dominio lógico y transformándolos en SPARQL, que son legibles por la máquina. Sin embargo, los autores reconocen que SHACL es adecuado en los contextos de validación. En [33], en lugar de validar los datos RDF contra el esquema, un modelo se valida contra los datos RDF. El propósito es validar la credibilidad del modelo o proporcionar la capacidad de aprendizaje de las ontologías. La validación se realiza utilizando heurísticas de puntuación de axiomas basadas en la teoría de posibilidades.

10.3.3 **Revisión del estado del arte en el contexto del ciclo de vida**

Este libro [87] representa una visión histórica de la evolución de los modelos de ciclo de vida del software durante los últimos sesenta años. Este artículo [88] es una revisión histórica de las últimas seis décadas de la evolución del ciclo de vida del software. Este trabajo [89] propone una metodología basada en cuestionarios para obtener y validar los requisitos del modelo creados por el experto en modelado. La validación se realiza de forma no automatizada con la colaboración de expertos en el dominio. Los autores consideran que existe una inconsciencia tanto académica como industrial sobre el establecimiento de un puente entre las diferentes etapas del ciclo de vida del software con un esfuerzo razonable. La razón por la cual los autores han clasificado la revisión bibliográfica de esta manera consiste en que MDE se ha aplicado erróneamente a las primeras etapas del proceso de ingeniería de software. Algunos trabajos en el estado del arte abordan esta problemática.

En el contexto de esta tesis se ha contribuido en esta problemática, reduciendo la brecha entre las diferentes etapas del ciclo de vida o abordando la falta de alineamiento e integración entre el proceso de ingeniería de software y el proceso de desarrollo. En la revisión de la literatura que se ha realizado, se encontraron pocos trabajos que aborden la brecha entre las diferentes etapas del ciclo de vida o que aborden la falta de alineamiento e integración entre el proceso de ingeniería de software y el proceso de desarrollo de software. Estos hechos han dado lugar a la dificultad de adaptar las normas que se ocupan de los ciclos de vida de un proyecto determinado, por ejemplo, IEC 12207. Además, la metodología Agile para el proceso de desarrollo no suele contemplar el concepto del modelo en ninguna de sus fases. El enfoque propuesto en contexto de esta tesis es adoptar el modelado (MDE) en todas las etapas de desarrollo de software y el proceso de ingeniería de software. MDE generalmente aborda el ciclo de vida de desarrollo de software desde el diseño hasta las fases de entrega con un paradigma orientado a objetos (UML). Sin embargo, el paradigma ontológico se utiliza sólo para el modelado estructural y la serialización. Tal es el caso de CIM en las compañías eléctricas. Las especificaciones de requisitos no se han abordado con MDE. Consideramos ambos paradigmas, el orientado a objetos y ontológico como formalismos válidos para el modelado en el proceso de la ingeniería del software. MDE puede beneficiarse de la integración de ontologías en términos de

obtener modelos directamente ejecutables con capacidad de razonamiento. Ambos paradigmas (ontológico y orientado a objetos) son complementarios. El hecho de que el modelo sea ejecutable proporciona transparencia y minimiza la codificación *hardcoding*, e intrínsecamente sirve como una metodología ágil. Proporciona el rigor del que carece la metodología ágil. Este enfoque ha sido evaluado en el dominio de las compañías eléctricas. Las reglas (OCL y basadas en texto) del formato CIM (CGMES) se han modelado en SHACL.

10.4. **Enfoque propuesto**

Como el enfoque se basa en el proceso de modelado y sus paradigmas; una definición de modelo, paradigma de modelado y repositorio de paradigmas es la siguiente:

Modelo = {conceptos, relaciones, reglas, transformaciones, comunicaciones}.

Esta definición puede considerarse la más detallada en términos de sus conceptos básicos.

Un paradigma de modelado es un patrón que consiste en la composición de un conjunto de conceptos para interpretar la realidad.

Un repositorio de paradigmas es un conjunto de paradigmas a partir de los cuales se selecciona un paradigma de modelado para adaptarse mejor al dominio.

En la Figura 10.3 se ilustra el enfoque de modelado multiparadigma, considerando el repositorio de paradigmas.

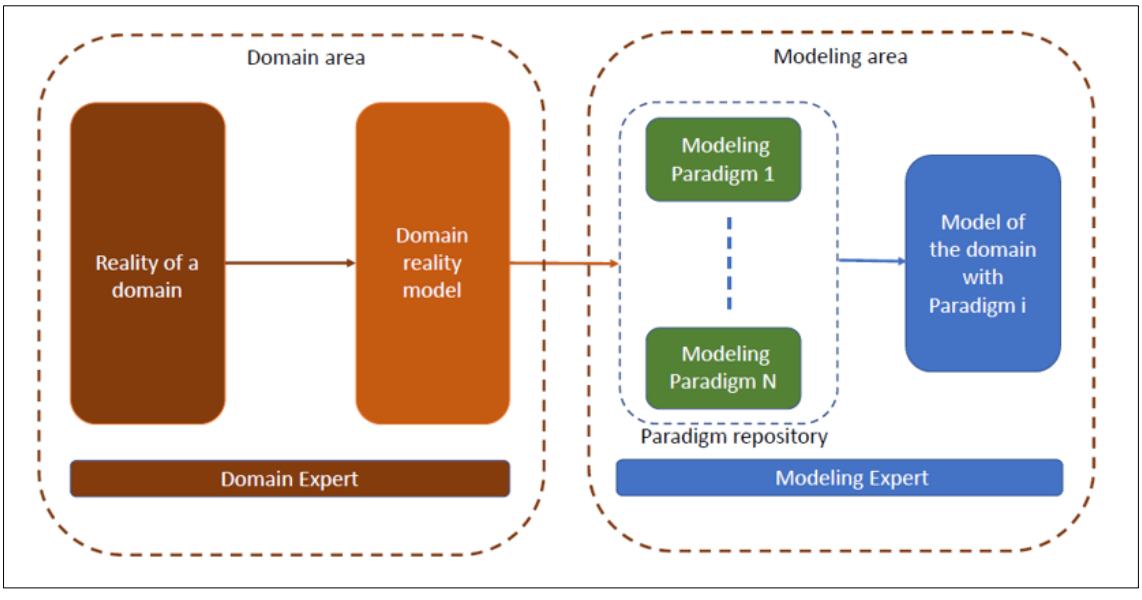


Figura 10.3: Repositorio de paradigmas y el modelado de la realidad de un dominio.

Es fundamental que tanto los expertos en modelado como los expertos en el dominio consensuen abordar el proceso de ingeniería para la producción de un sistema en términos de su modelo. En este contexto, se entiende que un sistema puede ser

definido como una composición de subsistemas interrelacionados que operan dentro de un contexto específico.
El acuerdo entre estos dos actores es crucial, ya que el modelo del sistema se convierte en una representación conceptual y abstracta de la realidad que se pretende construir. Este modelo actúa como una guía durante todo el proceso de ingeniería, facilitando la comprensión compartida de los requisitos, las funcionalidades y las restricciones del sistema en cuestión.
Los expertos en modelado aportan su experiencia en la creación de representaciones estructuradas y formales del sistema, utilizando lenguajes y técnicas especializadas. Estas representaciones modeladas permiten capturar aspectos clave del sistema, como su estructura, comportamiento y relaciones entre los diferentes subsistemas.
Por otro lado, los expertos en el dominio poseen un conocimiento profundo de la problemática y los requisitos específicos del contexto en el que se desarrollará el sistema. Su participación activa y colaborativa es esencial para asegurar que el modelo resultante sea una adecuada y fiel representación de las necesidades del dominio.
Al establecer un acuerdo conjunto en cuanto al enfoque del proceso de ingeniería y al reconocer la importancia del modelo como herramienta central, se facilita la comunicación y el alineamiento entre los expertos en modelado y en el dominio. Esto, a su vez, promueve un desarrollo más eficiente y efectivo del sistema, ya que se establece una base sólida y común para la toma de decisiones y la solución de problemas a lo largo de todo el ciclo de vida del proyecto.
El contexto puede ser conceptualizado como un sistema compuesto por un conjunto de subsistemas interrelacionados, los cuales poseen una configuración específica y cumplen una función que determina su utilidad dentro del sistema global. En este sentido, los sistemas pueden ser representados y analizados mediante técnicas de modelado, tal como se ilustra en la Figura 10.4 que proporciona una representación visual de cómo se pueden capturar y representar los sistemas a través del modelado. Mediante la utilización de herramientas y técnicas adecuadas, es posible identificar y describir los componentes clave de un sistema, sus relaciones y su interacción con el entorno circundante.

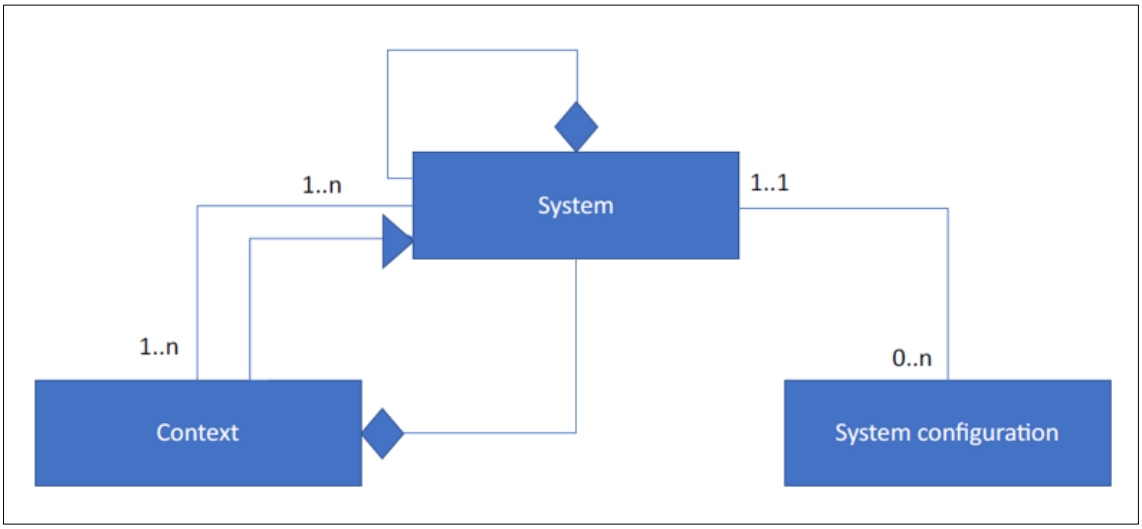


Figura 10.4: El modelo de un sistema.

Toda actividad encaminada a obtener un artefacto útil requiere de dos tipos de tareas: una tarea de gestión ya sea consciente o inconsciente, y una tarea técnica. En el contexto de la ingeniería de software, los procesos fundamentales involucrados son el proceso de gestión de ingeniería de software y el proceso de desarrollo de software. La Figura 10.5 ilustra el metamodelo utilizado para representar estos procesos.

El proceso de gestión de ingeniería de software abarca las actividades relacionadas con la planificación, organización y supervisión del proyecto de software en su totalidad. Incluye la definición de los objetivos, la asignación de recursos, la coordinación de equipos y la evaluación del progreso y los resultados obtenidos. Esta tarea de gestión tiene como objetivo garantizar que el proyecto se desarrolle de manera eficiente, dentro del marco establecido y cumpliendo con los requisitos establecidos.
Por otro lado, el proceso de desarrollo de software se centra en las actividades técnicas necesarias para la construcción del software en sí. Esto implica la aplicación de metodologías, técnicas y herramientas específicas para diseñar, implementar, probar y mantener el sistema de software. El proceso de desarrollo de software se basa en el uso de buenas prácticas de ingeniería, con el fin de asegurar la calidad, la eficiencia y la funcionalidad del producto final.

El metamodelo representado en la Figura 10.5 proporciona una estructura visual para comprender y comunicar los diferentes elementos y relaciones involucrados en los procesos de gestión de ingeniería de software y desarrollo de software. Sirve como un marco conceptual que ayuda a los profesionales de la ingeniería de software a comprender la secuencia lógica de actividades y los roles asociados en cada etapa del proceso.

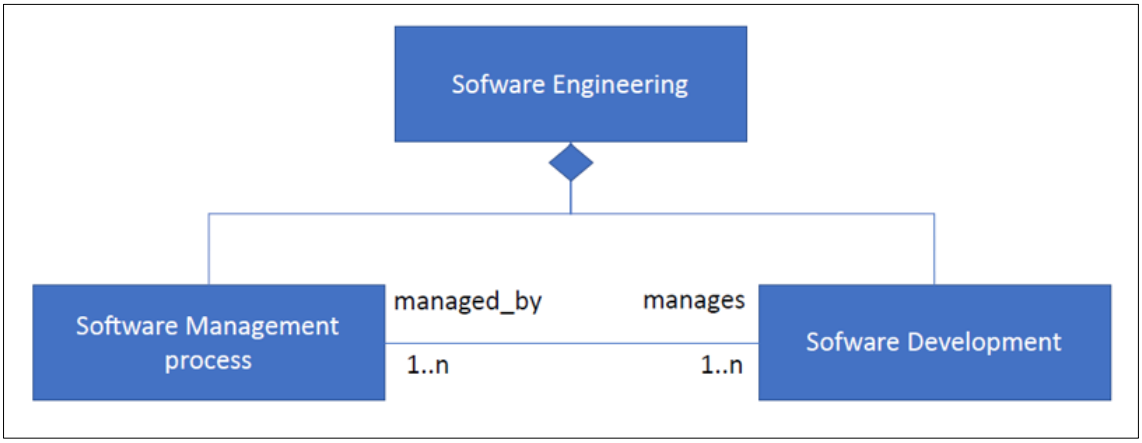


Figura 10.5: Metamodelo del proceso de la ingeniería del software.

Dentro del enfoque de Ingeniería Dirigida por Modelos (MDE), se reconoce que todo elemento del sistema puede ser concebido como un modelo. Sin embargo, hasta ahora, la aplicación de MDE no ha sido considerada de manera adecuada en la fase de especificación de requisitos. No obstante, es posible abordar la especificación de requisitos como un proceso de modelado, en el cual los conceptos que definen los requisitos son comunicados desde una fuente de conceptos al experto del modelado. Figura 10.6 muestra un modelo representativo de este proceso.

En este contexto, la especificación de requisitos se trata como una actividad de modelado, donde se busca establecer una comunicación efectiva entre la fuente de conceptos, que posee el conocimiento sobre los requisitos del sistema, y el experto en modelado, encargado de transformar estos conceptos en modelos formales.

El proceso de especificación de requisitos se inicia con la identificación y captura de los conceptos relevantes, los cuales son proporcionados por la fuente de conceptos. Estos conceptos pueden incluir información sobre los objetivos, las funcionalidades, las restricciones y otros aspectos importantes del sistema. Después, el experto en modelado utiliza estos conceptos como base para construir modelos formales que representen los requisitos del sistema de manera precisa y comprensible.

El modelo presentado en la Figura 10.6 es una representación visual de este proceso de comunicación y modelado. Proporciona una estructura conceptual para comprender y comunicar los flujos de información y los roles involucrados en la especificación de requisitos basada en MDE.

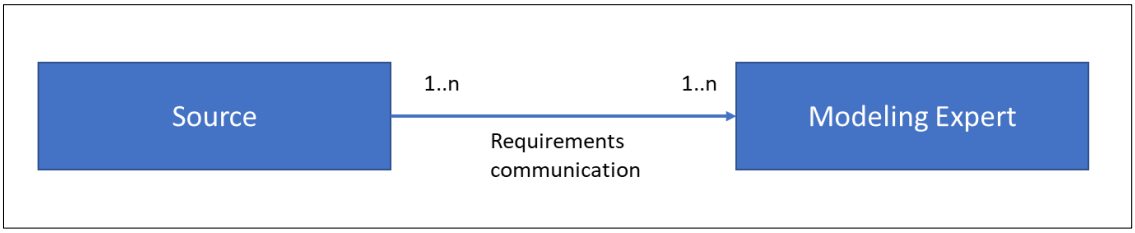


Figura 10.6: Modelo de la comunicación de requisitos.

El modelo en sí consiste en conceptos, relaciones, comunicaciones, transformaciones y reglas. El esfuerzo de la actividad de especificación de requisitos se basa en el concepto de comunicación entre el experto en modelado y la fuente, que es parte de lo que son los constructos de modelado. Por lo tanto, la metodología propuesta en este trabajo es ágil.

La Figura 10.7 muestra un metamodelo de la fuente de los requisitos. Estos conceptos se muestran en este metamodelo: CIM, PIM y PSM. Una definición detallada de estos tres conceptos se puede encontrar en [49] [25]. La entropía en el formalismo y la comunicación se refiere al grado de desorden en la comunicación entre la fuente de los requisitos y el experto del modelo.

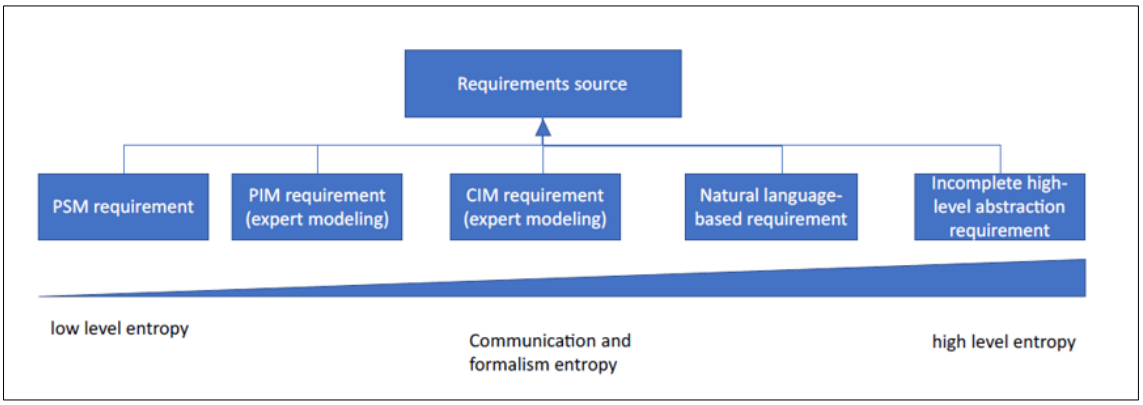


Figura 10.7: Clasificación de las fuentes de requisitos.

La especificación de requisitos es modelada considerando los requisitos como una composición de objetivos, cada uno puede ser mapeado a uno o más modelos. La Figura 10.8 muestra el modelo de los requisitos.

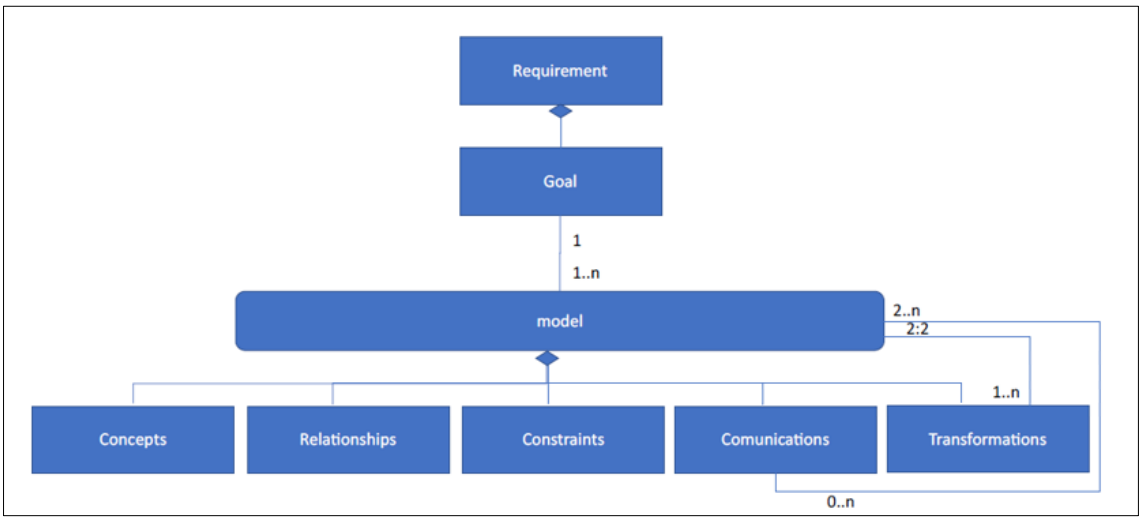


Figura 10.8: Modelo de requisitos.

Se requiere capacitación experta en el dominio. Se debe proporcionar la capacitación para adquirir las habilidades para colaborar en el contexto del proceso de modelado. Empatía cognitiva: el experto en el dominio debe ser consciente del enfoque de modelado para especificar el modelo a transferir de su dominio al experto en modelado.

Inicialmente, un primer modelo que representa la especificación de requisitos considera las áreas de experto en dominio y experto en modelado. La Figura 10.9 muestra un metamodelo independiente del paradigma para el proceso de modelado asociado con el requisito detrás de un objetivo y el algoritmo para este proceso de modelado se describe en el Listado 10.1. Las principales tareas por abordar para desarrollar un modelo son las siguientes:

- Obtención de conceptos y relaciones entre ellos para la construcción del modelo.

- Validación del modelo, realizada por el experto en el dominio.
- Modelado de reglas para cumplir con los requisitos.
- Actualización del modelo

Como se puede concluir de la construcción del modelo, la gestión del modelo puede concebirse como un proceso colaborativo e iterativo que implica la validación para obtener un modelo acordado entre el experto de dominio y el experto en modelado. Si el modelo no es válido, el experto del dominio transmite el motivo/ubicación de las inconsistencias, refinándolas iterativamente hasta que el modelo sea válido y útil. Las instancias se generan para comprobar que las reglas se modelan correctamente con respecto a la especificación de un experto en el dominio y, a continuación, el modelo se valida contra las instancias del dominio (que supuestamente son válidas). Si se produce una infracción en este proceso de validación, es necesario actualizar el modelo en consecuencia.

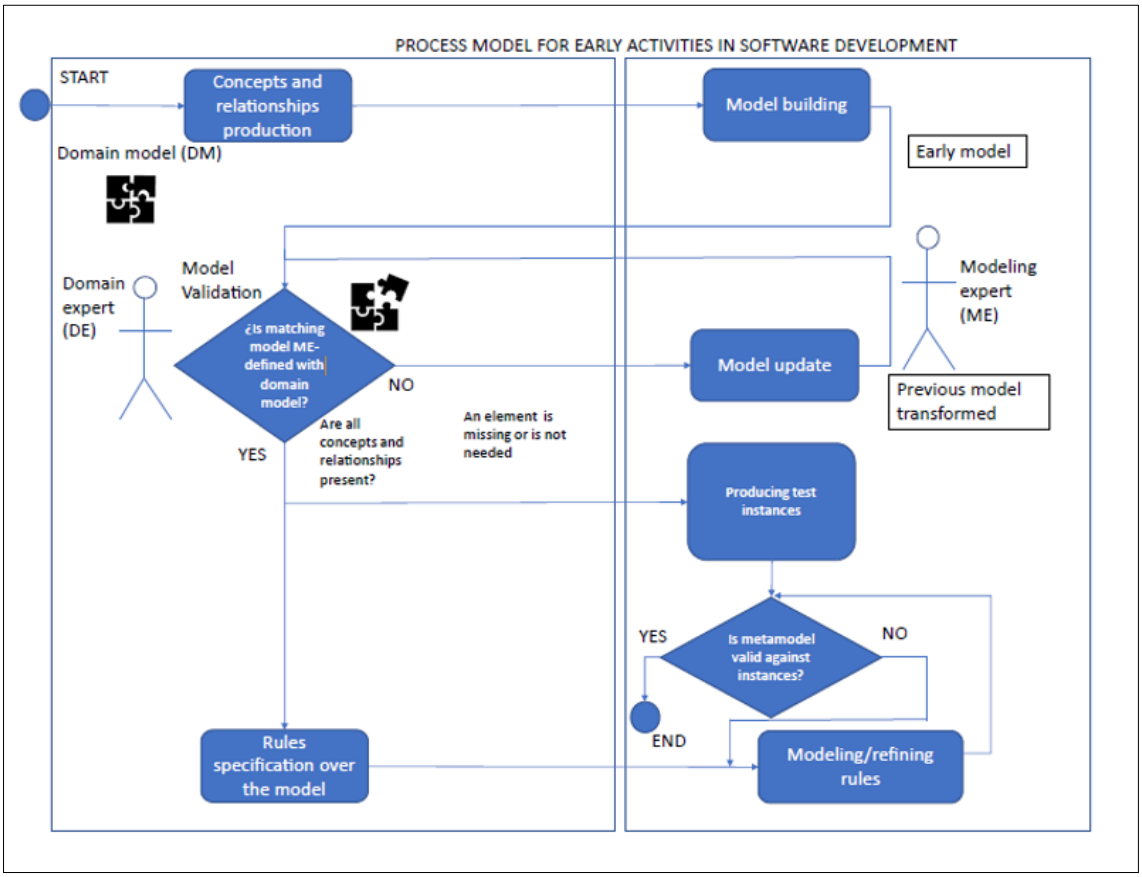


Figura 10.9: Modelo de las tareas para las etapas tempranas del desarrollo del software.

```
Start
# Select a modeling paradigm from modeling paradigm repository
# Paradigm repository PR = {Paradigm1, ..., ParadigmN}
# selectParadigm is a task performed by modeling expert.
P = selectParadigm(PR)
```

```
# D is domain of interest
# CR is the set of concepts and relationships in the domain of interest
CR = produceConceptsAndRelationships(D)
# M is the model in the modeling expert domain
# modelbuilding is task performed by modeling expert to build a model from the
concepts and
relationships
# with paradigm P.
M = modelbuilding(CR,P)
# VR is the validation report. It contains a field indicating the conformity and an
inconsistencies sequence.
# modelValidation is a task performed by the domain expert to check the model
validity limited
to concepts and relationships
# under the modeling paradigm P.
VR = modelValidation(M,P)
# The model is updated until is validated by the expert domain
While Not (conforms(VR)) {
M = modelUpdate(M,VR,P)
VR = modelValidation(M,P)
}
# rulesSpecification is task performed by the expert domain to obtain the set of rules
R (in
domain expert language).
R=rulesSpecification(D)
# RM is the set of rules defined by the modeling expert
# rulesModeling is task performed by the modeling expert to obtain the set of rules
RM (in
modeling language)
RM = rulesModeling(R, P)
# Both expert domain and modeling expert collaborate to create strategically a set of
valid and
invalid model instances I.
I=produceInstances(M, RM,P)
# A machine process validates (M, RM) against I and it returns a validation report VR.
VR=Validate (I, M, RM, P)
# while VR does not conform, (M, RM) is refined and validated again.
while Not (conforms (VR)) {
RM=refineMetamodel(M, RM,P)
VR=Validate (I, M, RM,P)
}
```

Listado 10.1: Algoritmo del proceso de modelado en la fase de especificación de requisitos.

Debido a que el proceso de construcción del modelo es iterativo, puede ser representado por dos estados elementales (máquina de estados): el estado intermedio del modelo y un estado compatible y útil (consistente y acordado). El modelo estará en

un estado intermedio mientras se consideran o actualizan nuevos conceptos. El modelo estará en un estado compatible y acordado cuando el experto en modelado considere que el modelo es válido y completo. La Figura 10.10 muestra tal máquina de estado.

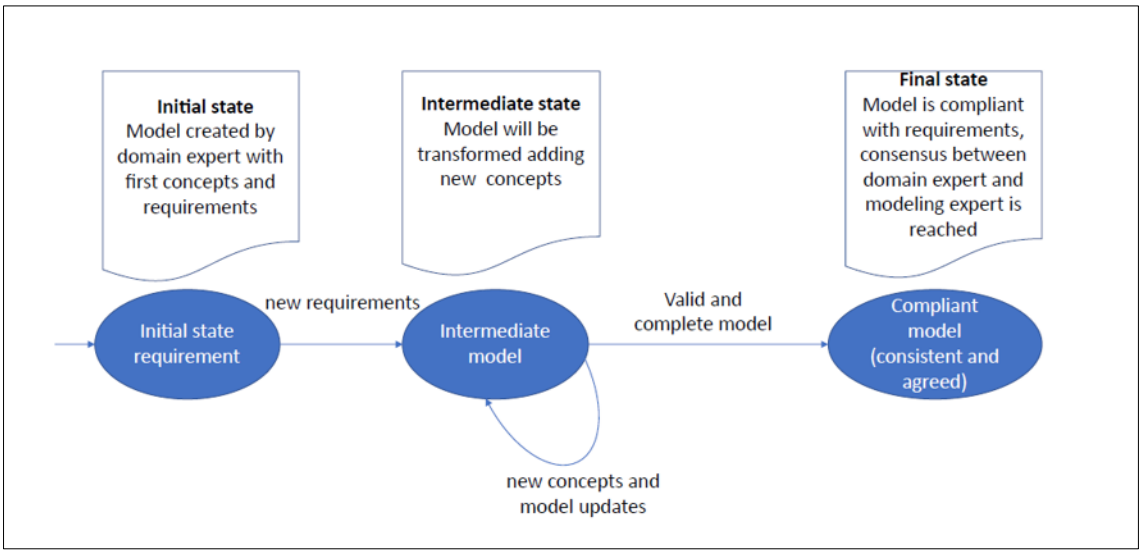


Figura 10.10: La máquina de estados del proceso de modelado.

El enfoque no está limitado por UML o RDF/OWL/SHACL/SPARQL ya que tiene una naturaleza multiparadigma; se espera que haya aportaciones en este sentido en términos de nuevos paradigmas y lenguajes para fines de modelado que tengan como objetivo producir modelos ejecutables. En este contexto, el enfoque ontológico se utiliza para mejorar la ingeniería de software, y nunca para reemplazar UML. La Figura 10.11 representa un enfoque tradicional frente al enfoque MDE para el proceso de desarrollo de software. La parte izquierda representa el enfoque tradicional, que consiste en cargar datos, un algoritmo imperativo, operaciones CRUD sobre datos, resultados y presentación de datos. Con el nuevo enfoque, el desarrollo de software debe abordarse como una tarea de ingeniería de modelado en términos de modelo / metamodelo / reglas, ya que de esta manera permite a los desarrolladores tener una comprensión sostenible del sistema y su contexto. Con este enfoque, el modelado es omnipresente en todas las fases del ciclo de vida del software. Las actividades para desarrollar un algoritmo se traducen en operaciones CRUD sobre modelos (en este caso, ontologías). Los resultados y los informes se modelan a través de consultas SPARQL y formas SHACL de una manera legible por humanos y máquinas. La Figura 10.11 muestra las tareas fundamentales en el desarrollo de software y su correspondiente enfoque de desarrollo ontológico.

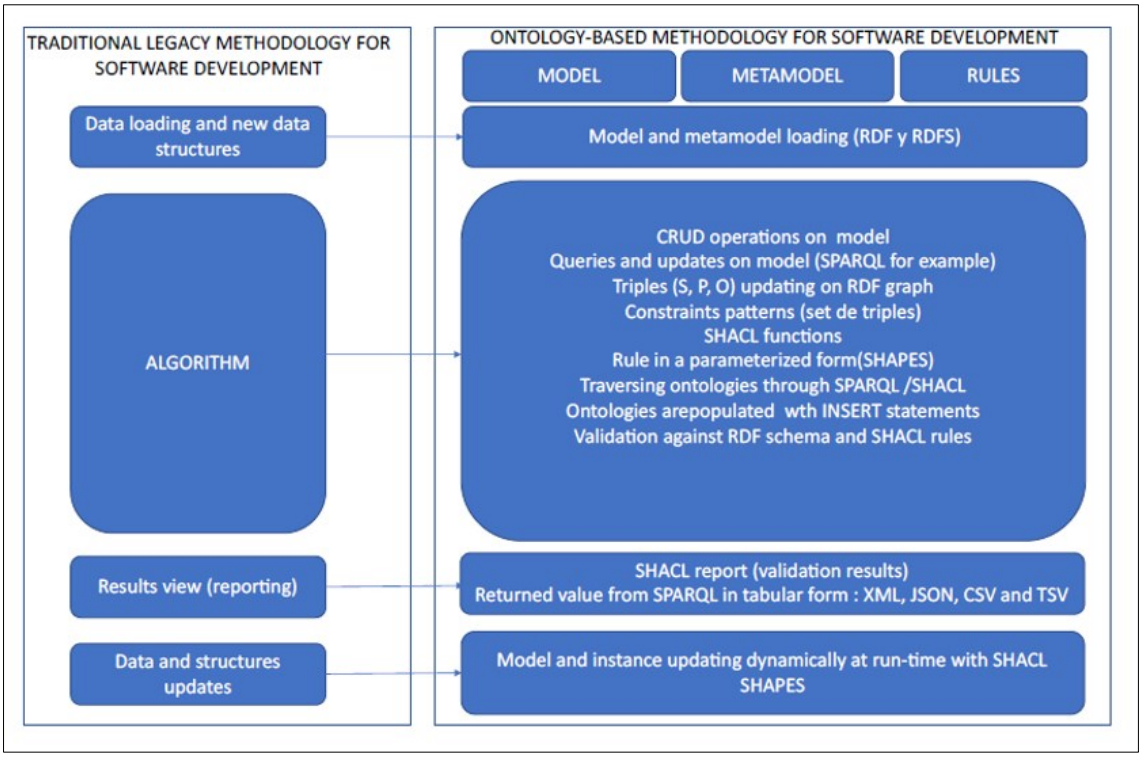


Figura 10.11: Modelo tradicional VS. modelo ontológico para el desarrollo del software.

10.5. **Evaluación del enfoque propuesto**

El propósito de esta sección es describir la evaluación del nuevo enfoque de modelado. La hoja de ruta de las subsecciones es la siguiente. En la sección 9.5.1 se presenta el impacto del enfoque sobre la arquitectura modelo-vista-controlador (MVC). En la arquitectura MVC, todas las reglas de negocio se han movido a la capa del modelo. Dado que las reglas son directamente ejecutables en el enfoque propuesto, esto es factible. La Sección 11.5.2 muestra los fundamentos y conceptos básicos para modelar una regla en SHACL. Un ejemplo de regla en SHACL se describe con su grafo RDF correspondiente en términos de sus constructos para mapear la definición de la regla en lenguaje natural con los constructos del lenguaje SHACL. La Sección 11.5.3 presenta la especificación de requisitos con el enfoque MDE aplicado a las reglas CGMES basadas en texto.

10.5.1 **El impacto del enfoque propuesto en la arquitectura MVC**

La arquitectura MVC se utiliza para evaluar este enfoque. MVC consta de las siguientes tres capas: modelo, vista y controlador. La capa del modelo es donde se asignan los datos, su estructura y reglas. La capa de vista desempeña el papel de interfaz entre el sistema y su entorno, y finalmente, la capa de controlador es donde las reglas de negocio se implementan principalmente con *hardcoding*.

El esfuerzo que solía dedicarse a la codificación en la capa del controlador se mueve a la capa de modelado ya que el modelo que será directamente ejecutable se desarrolla a partir de la fase de requisitos. La Figura 10.12 y la Figura 10.13 muestran cómo la capa del modelo reemplaza a la capa del controlador.

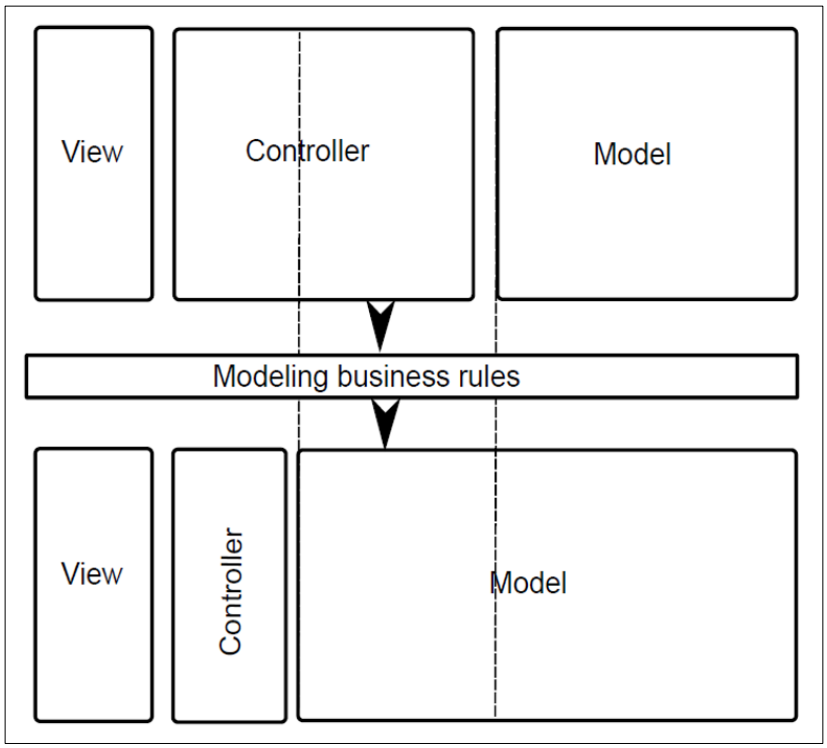


Figura 10.12: El impacto del nuevo enfoque en la arquitectura MVC.

El modelado de reglas reemplaza su *hardcoding*. En la Figura 10.13, el ancho de cada capa representa el esfuerzo en términos de código asociado con el desarrollo de software y el papel de cada capa. Una capa más estrecha significa que su función o funcionalidad ha sido reducida o asumida por la capa controladora.

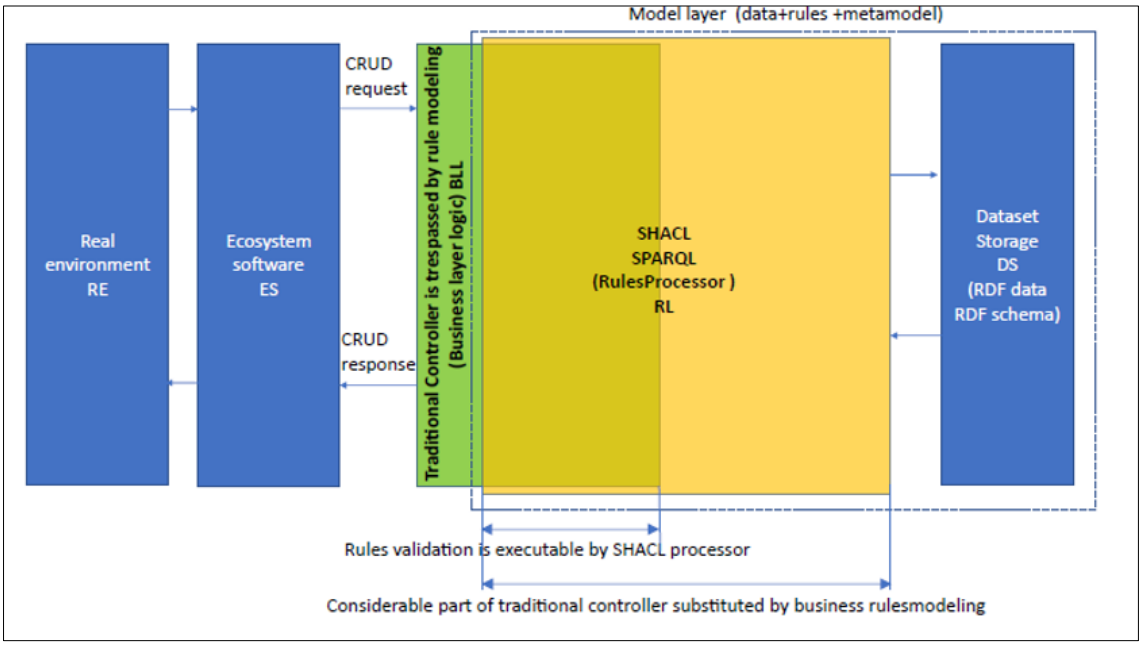


Figura 10.13: El impacto del modelado de las reglas en la capa controlador.

Como resultado, solo queda una parte del área del controlador. En este contexto, el lenguaje para el modelado es SHACL, y los datos y metadatos se almacenan en *datasets*. La interacción entre el controlador y el ecosistema se realiza a través de operaciones CRUD. El software del ecosistema interactúa con el contexto del entorno real, que puede ser hardware o sistemas de software.

10.5.2 **Constructos básicos y el modelado de una regla**

El modelado básico de una regla SHACL se describe a continuación. Su objetivo es proporcionar un trasfondo esencial sobre SHACL, RDF y su uso en las siguientes secciones. Los constructos básicos de SHACL [46] [26] son los siguientes:

- sh:NodeShape: Es una clase SHACL que representa la definición de restricciones.
- sh:PropertyShape: Los objetos de esta clase representan la definición de restricciones.
- sh:targetClass: Se utiliza para especificar nodos de foco desde los que el procesador SHACL comprueba las condiciones de la forma.
- sh:property: Es donde se describen las condiciones de restricción.
- sh:path:Pattern que representa la regla puede ser una arista que representa un predicado entre un sujeto (nodo) y un objeto (nodo). Con este patrón, el procesador SHACL atraviesa el grafo para la comprobación de condición. Comienza a atravesar el grafo, y para cada nodo de enfoque determina el borde (predicado) representado por sh:path. Dado que SPARQL es un lenguaje de consulta para un grafo RDF, SPARQL se puede incrustar en SHACL para expresarlos si se necesitan reglas sofisticadas.
- sh:mensaje. Es opcional y contiene un texto que describe el mensaje que se muestra cuando se infringe la regla.

Tanto sh:NodeShape como sh:PropertyShape se utilizan para definir restricciones y SHAPE es una restricción en el vocabulario de SHACL.

La Figura 10.14 muestra un modelo RDF, representado a través de un grafo y sus tripletas.

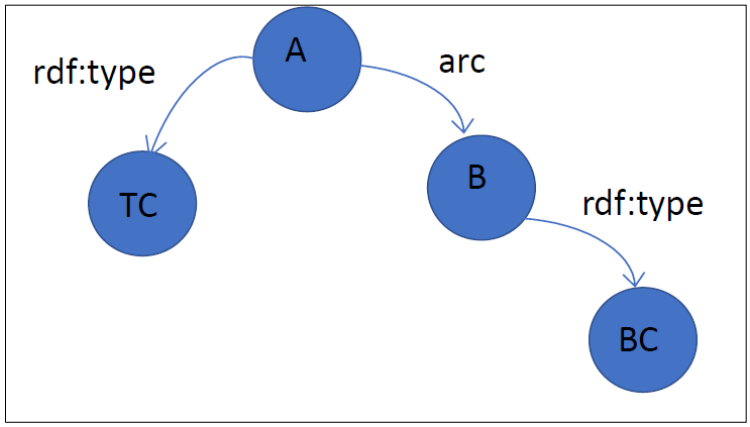


Figura 10.14: Ejemplo de grafo RDF.

Las tripletas RDF de este modelo de muestra se presentan en el Listado 10.2 Listado 2.

```
A rdf:type TC.
A arc B.
B rdf:type BC.
```

Listado 10.2: Tripletas RDF del grafo de la figura 11.14.

Tanto el grafo como las tripletas RDF representan el mismo modelo. Supongamos que la siguiente restricción, definida como lenguaje natural, se aplica al grafo RDF anterior. Cada instancia de TC debe tener al menos una asociación con una instancia de la clase BC.

Esto significa que al menos una arista debe comenzar desde el nodo A y llegar a otro nodo B cuyo tipo debe ser de clase BC.
De acuerdo con los constructos definidos anteriormente, el código SHACL correspondiente a esta restricción se muestra en el Listado 10.3.

```
X a sh:NodeShape;
sh:message "arc must be set";
sh:targetClass TC;
sh:property [
sh:path arc;
sh:minCount 1;
sh:class BC
].
```

Listado 10.3: Regla SHACL.

10.5.3 **Especificación de requisitos con el enfoque MDE aplicado a las reglas CGMES basadas en texto.**

10.5.3.1. Algoritmo con enfoque de modelado.

El algoritmo para la transformación manual de una regla basa en texto a SHACL se muestra en el Listado 10.4.

```
textBasedToShaclManualReviewed(text_rule,schemaDomain)
#TBR = {is a set of the concepts in the Text-Based Rule};
# extractTextConcepts is a task performed by both experts.
TBR = extractTextConcepts(text_rule)
# ICC is a set of pairs (text-based-concept,{set of RDFS concepts})
# where {set of RDFS concepts} is the RDFS concepts that correspond
# to text-based-based-concept.
ICC = {(textbasedconcept,{set of RDFS concepts})}
RC = {set of pair of related concepts, initially empty};
Foreach c in TBR
# {cd} is a set of domain concepts defined in RDF schema.
# rdfsConcepts is a task performed by the modeling expert
# and domain expert to find the RDFS concepts that
```

```
# corresponds to textual textual concept c. rdfsConcepts
# returns the pair CD = (c,{cd})
CD=(c,{cd})= rdfsConcepts(c);
ICC->add(CD);
End Foreach
# relatedConcepts(ICC, text_rule) is a task performed by the modeling expert (new)
# and domain expert to obtain relationships between the RDFS concepts that
corresponds
to textual concepts. See Fig. 15 (new)
RC = relatedConcepts(ICC, text_rule);
# constructPattern is a modeler task which consists of
# searching for a subgraph that interconnects the concepts in RC pairs.
# see Figure 11.16 and Figure 11.17
pattern = constructPattern(RC, schemaDomain);
# getOperationsInTextBasedRules is a modeler task which
# consists of extracting from text_based the operations and their
# associated operands involved in the rule from the pattern.
OO = {(Operation,{Operands})} = getOperationsInTextBasedRules(text_rule, pattern);
# createSHACLShape is a modeler manual task which consists of
# building SHACL shape from the pattern and the operations stated
# in text-based rule
SHACLShape = createSHACLShape (pattern,OO);
Return SHACLShape;
End textBasedToShaclManualReviewed
```

Listado 10.4: Regala basada en texto a SHACL con el enfoque de modelado.

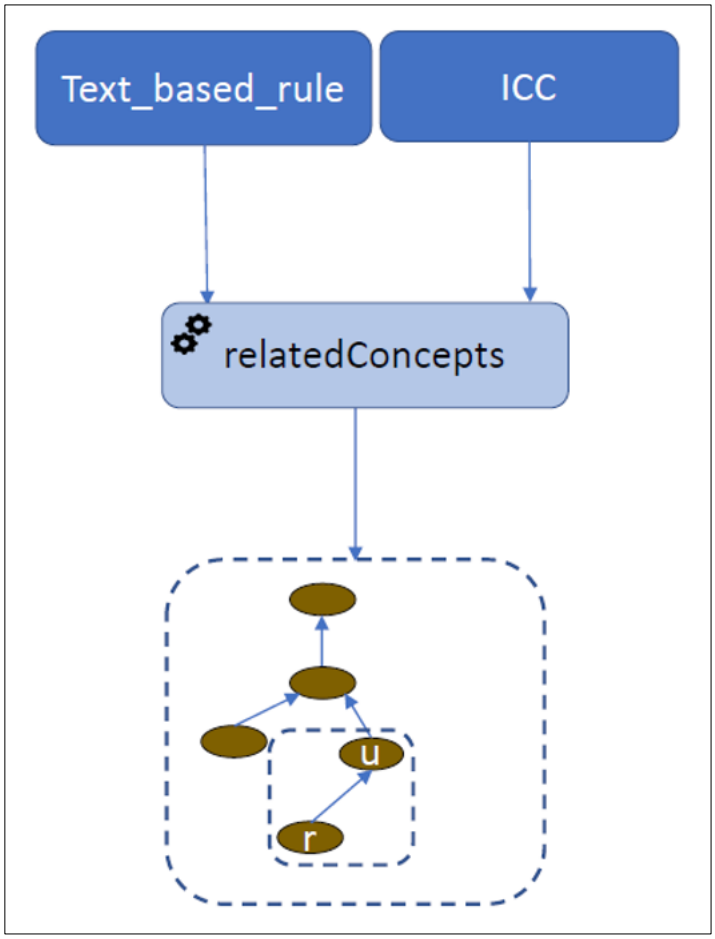


Figura 10.15: Conceptos relacionados en RC (Subconjunto de RDFS del dominio).

El diagrama de flujo descrito en la Figura 10.18 ilustra el algoritmo representado en el Listado 10.5.

10.5.3.2. Ejemplo de modelado de una regla CGMES

Para evaluar el algoritmo anterior, la regla 5_5 del nivel 5 correspondiente a QoCDC 2.2 del estándar CGMES 2.4.15 se ha tomado como especificación de requisitos para modelarla en SHACL.

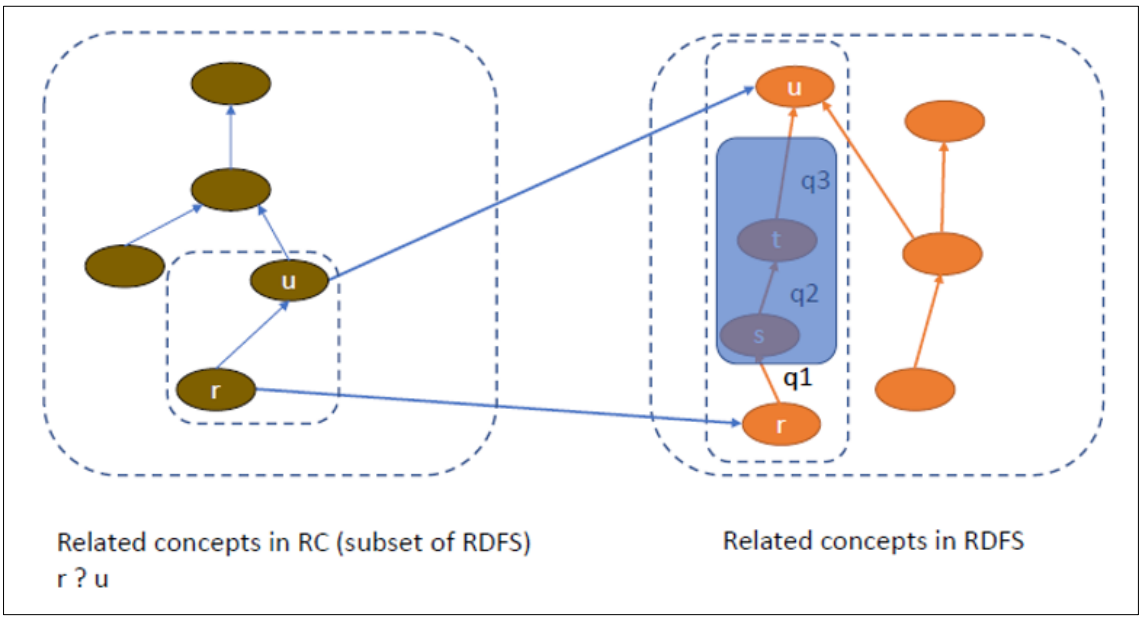


Figura 10.16: Búsqueda del camino para los conceptos relacionados en RC.

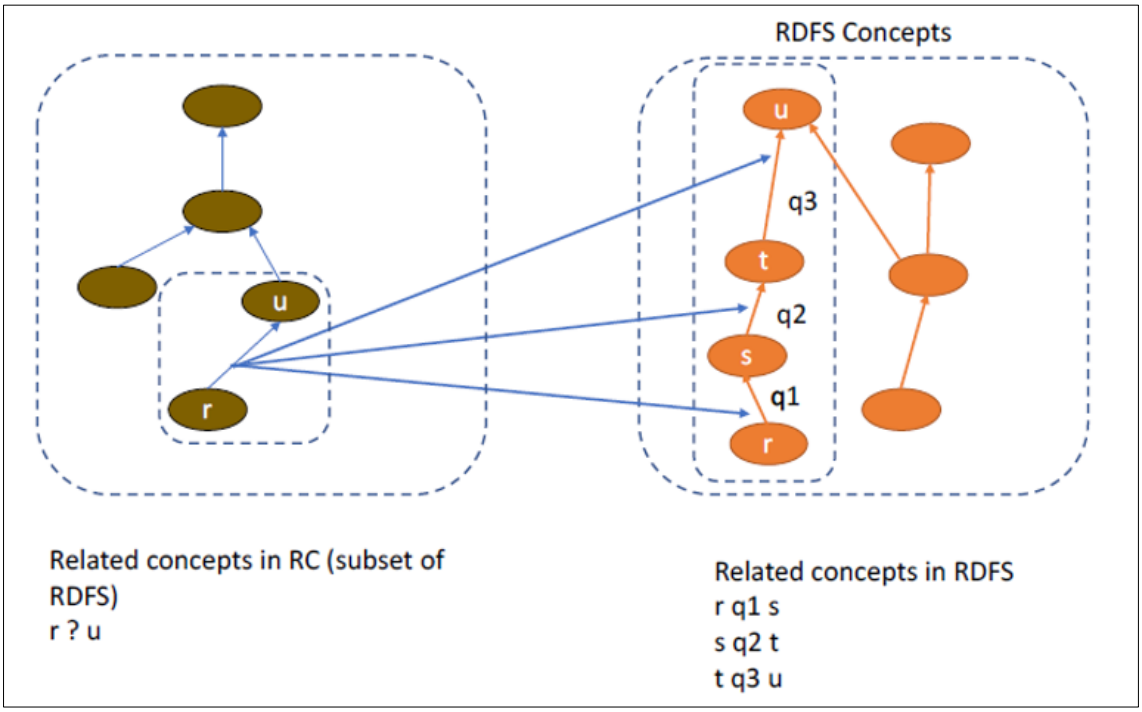


Figura 10.17: Mapeo de relación.

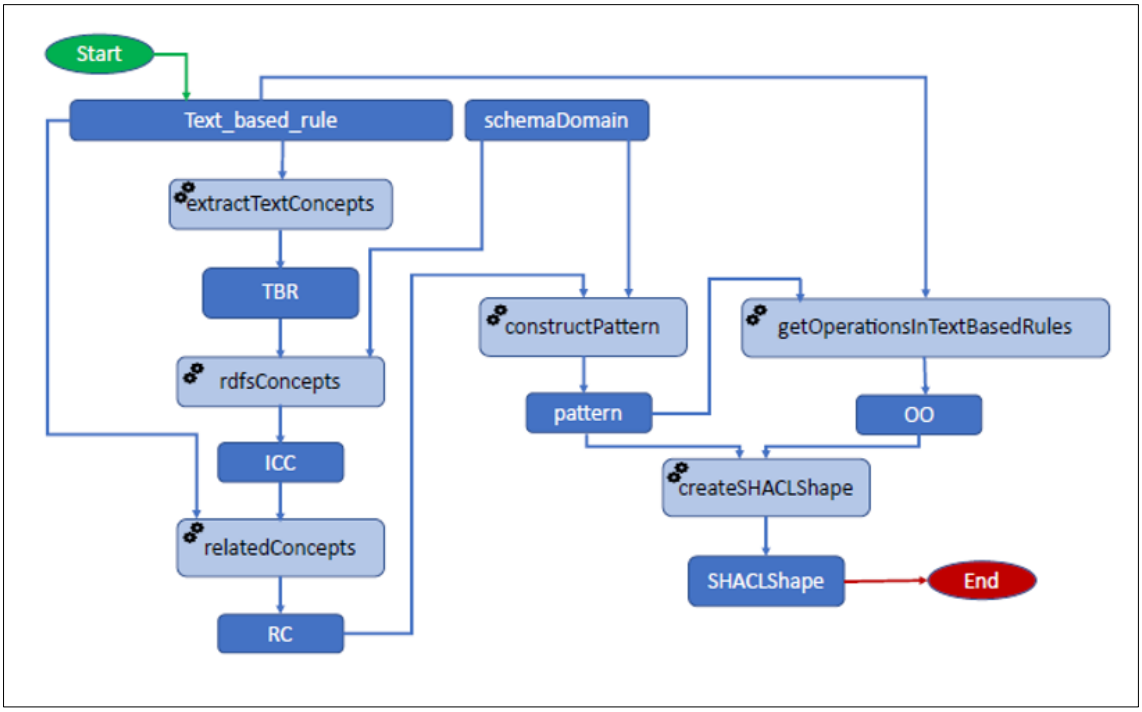


Figura 10.18: Diagrama de flujo del algoritmo textBasedToShaclManualReviewed.

En el Listado 10.5 se muestra Una aplicación del algoritmo textBasedToShaclManualReviewed para una regla del CGMES.

text_rule = "5_5. All equipment associated with a specific Line container must have the same
BaseVoltage.nominalVoltage value.
MSG 23. WARNING: BaseVoltage.nominalVoltage** <Voltage1, Voltage2> **detected for the line
<IdentifiedObject.name>"
schemaDomain=" cim:Equipment rdf:type rdfs:Class;
rdfs:subClassOf cim:PowerSystemResource .
cim:BaseVoltage rdf:type rdfs:Class;
cims:belongsToCategory cim:Package_Core;
rdfs:comment "Defines a system base voltage which is referenced. ";
rdfs:label "BaseVoltage"@en;
cim:Line rdf:type rdfs:Class;
rdfs:subClassOf cim:EquipmentContainer .
Cim:BaseVoltage.nominalVoltage rdf:type rdf:Property ;
rdfs:domain cim:BaseVoltage .
cim:Equipment.EquipmentContainer rdf:type rdf:Property ;
rdfs:domain cim:Equipment ;
rdfs:range cim:EquipmentContainer .
"
textBasedToShaclManualImproved(text_rule, schemaDomain)
 # TBR is a set of the concepts in the Text-Based Rule
TBR = { equipment, line container, BaseVoltage.nominalVoltage};

```
# ICC = {(text-based-concept,{set of RDFS concepts})}
#Foreach c in TBR
c= "equipment"
CD={ ("equipment",cim:Equipment)} = rdfsConcept(c);
ICC->add(CD);
# ICC = {("equipment",{cim:Equipment})}
c= "line container"
CD = ("line container",{ cim:Line, cim:EquipmentContainer, Cim:Equipment.
EquipmentContainer})=rdfsConcept(c);
ICC->add(CD);
# ICC = {
# ("equipment",{cim:Equipment}),
# ("line container",{cim:Equipment, cim:Line, cim:EquipmentContainer,
Cim:Equipment.
EquipmentContainer})
#}
c = "BaseVoltage.nominalVoltage"
CD = ("BaseVoltage.nominalVoltage",{ BaseVoltage.nominalVoltage,
cim:BaseVoltage, cim:NominalVoltage})
= rdfsConcept(c)
ICC->add(CD)
# ICC = {("equipment",
{
cim:Equipment
}
),
("line container",
{
cim:Line, cim:EquipmentContainer,
Cim:Equipment.EquipmentContainer
}
),
("BaseVoltage.nominalVoltage",
{
BaseVoltage.nominalVoltage ,
cim:BaseVoltage, cim:NominalVoltage
}
)
}
#End Foreach
# RC is a set of pair of related concepts, initially empty
#
RC = {( cim:Equipment , cim: EquipmentContainer) ( cim:Equipment , cim:line), (
cim:Equipment , cim: BaseVoltage) ,( cim:Equipment, cim:NominalVoltage)};
# constructPattern is a modeler manual task which consists of
# searching in RDFS for a subgraph that interconnects the concepts in each pair of RC.
# SHACL processor needs that a focus node to be specified.
```

```
# from the text-based rule, objects of class cim:Equipment are considered focus node
which are represented by
# $this.
pattern = constructPattern(RC, schemaDomain);
#pattern = "
$this cim:Equipment.EquipmentContainer ?ec .
?ec rdf:type cim:Line .
$this cim:ConductingEquipment.BaseVoltage ?bv .
?bv cim:BaseVoltage.nominalVoltage ?nv.
?ce2 cim:Equipment.EquipmentContainer ?ec .
?ce2 cim:ConductingEquipment.BaseVoltage ?bv2 .
?bv2 cim:BaseVoltage.nominalVoltage ?nv2."
# getOperationsInTextBasedRules is a modeler manual task which
# consists of extracting from text_based the operations and their
# associated operands involved in the rule.
OO = {(Operation,{Operands})} = getOperationsInTextBasedRules(text_rule);
OO={("=",{?nv, ?nv2})}
# createSHACLShape is a modeler manual task which consists of
# building SHACL shape from the pattern and the operations stated
# in text-based rule
SHACLShape = createSHACLShape (pattern,OO);
Return SHACLShape;
End textBasedToShaclManualImproved
```

Listado 10.5: Aplicación del algoritmo textBasedToShaclManualImproved a la regla 5_5 de CGMES.

Como resultado de la aplicación del algoritmo anterior, se obtiene la regla SHACL del Listado 10.6.

```
sh:targetClass cim:Equipment;
sh:message "MSG 23. WARNING: BaseVoltage.nominalVoltage <Voltage1, Voltage2>
detected for the line <IdentifiedObject.name>.";
sh:sparql [
############################
# PREFIXES DECLARATION
#...
#END PREFIXES DECLARATION
############################
a sh:SPARQLConstraint;
sh:select """
select $this
where {
$this cim:ConductingEquipment.BaseVoltage ?bv .
$this cim:Equipment.EquipmentContainer ?ec .
?ec rdf:type cim:Line .
?bv cim:BaseVoltage.nominalVoltage ?nv.
?ce2 cim:ConductingEquipment.BaseVoltage ?bv2 .
?ce2 cim:Equipment.EquipmentContainer ?ec .
?bv2 cim:BaseVoltage.nominalVoltage ?nv2.
```

```
bind (xsd:double(?nv) as ?nvv)
bind (xsd:double(?nv2) as ?nv2v)
filter (($this != ?ce2) && (?nvv != ?nv2v))
}
""";
]
```

Listado 10.6: Regla SHACL obtenida con el algoritmo del listado 11-5.

10.6. Desarrollo de una herramienta para la validación de reglas CGMES con el nuevo enfoque

Con este enfoque, se han modelado todas las reglas basadas en texto de CGMES. Se ha puesto a disposición de los usuarios una aplicación para validar los modelos CGMES contra las reglas del nivel 3 una vez que las hayan modelado con el enfoque propuesto. La aplicación, junto con el esquema RDF de todos los perfiles y una introducción, está disponible en [90].

La aplicación requiere el archivo que corresponde al modelo CGMES y al archivo de reglas SHACL para realizar la validación y devuelve el informe de validación. La arquitectura de la aplicación se muestra en la Figura 10.19. Se ha proporcionado un ejemplo de una regla modelada y validada en el enlace de GitHub.

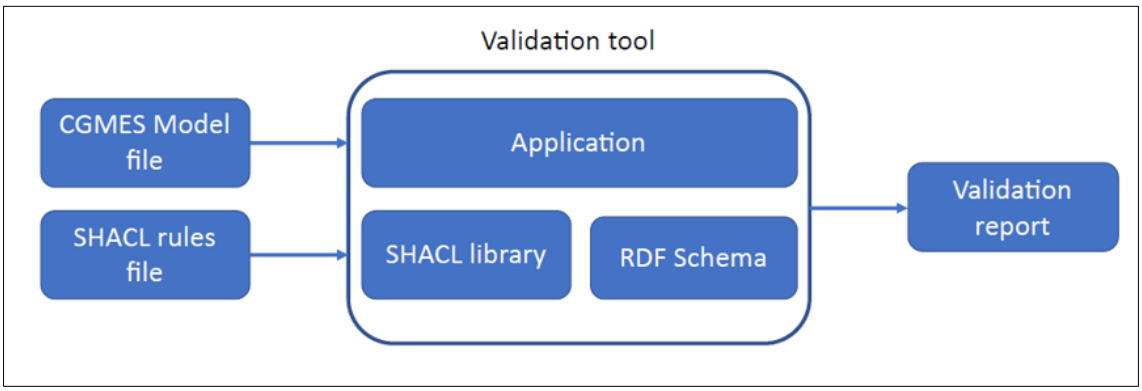


Figura 10.19: Arquitectura de la aplicación de validación.

La lógica de negocio representada por las reglas CGMES se ha modelado en la arquitectura MVC en lugar de estar codificada en un lenguaje de propósito general, como se muestra en la Figura 10.20. Por lo tanto, la ejecutabilidad de las reglas se ha trasladado de la capa de controlador a la capa de modelo.

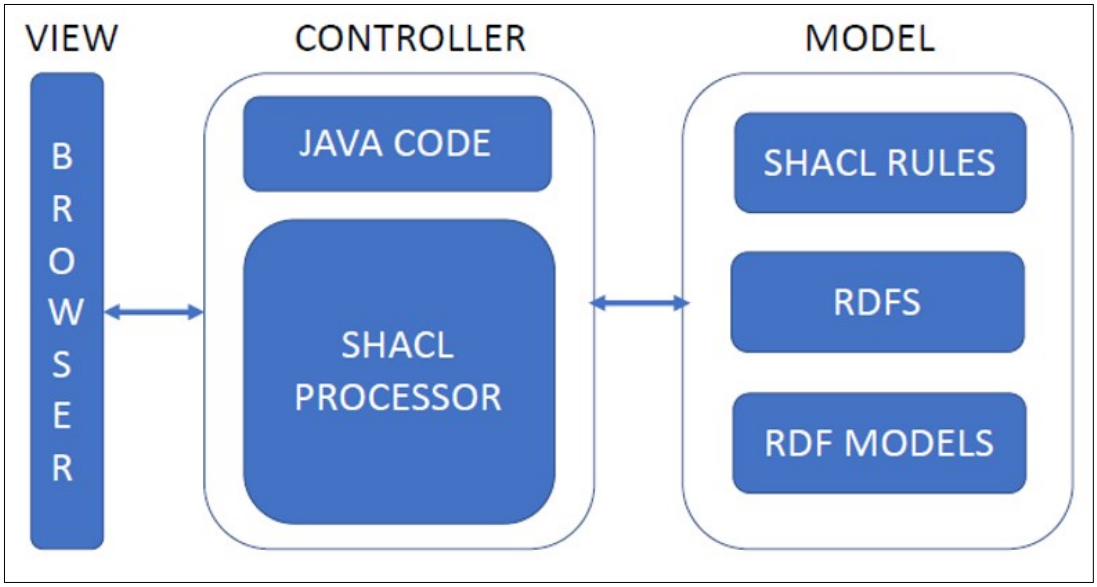


Figura 10.20: Arquitectura MVC de la aplicación desarrollada.

En la aplicación web anterior, todas las reglas del estándar CGMES han sido modeladas con el enfoque propuesto. El esfuerzo de desarrollo se ha desplazado de la capa del controlador a la capa del modelo porque el modelo que representa las reglas es directamente ejecutable.

10.7. Conclusiones y trabajos futuros

Este trabajo ha propuesto un enfoque de modelado en todas las fases del ciclo de vida del desarrollo de software, abordando principalmente las primeras etapas: la especificación de requisitos y el diseño para proporcionar una solución al hueco entre la especificación de requisitos y las siguientes fases. Se hace hincapié en el concepto de repositorio de paradigmas, ya que MDE se aborda comúnmente a través del paradigma orientado a objetos, pero el paradigma ontológico apenas se considera a pesar de que las ontologías constituyen un paradigma esencial en MDE. En este trabajo, el paradigma ontológico se ha utilizado dentro del enfoque MDE.

Se han mostrado diferentes modelos para cada una de las fases del ciclo de vida. Con este enfoque, se ha logrado una interacción del estilo de *Glass Box* entre el experto en el dominio y el experto en modelado.

Para evaluar este nuevo enfoque de modelado, todas las reglas del estándar CGMES en el dominio de los sistemas de potencia se han modelado en SHACL. Se ha desarrollado un repositorio de reglas para validar un modelo con una aplicación que se ha implementado a lo largo de este trabajo. Esta aplicación ha sido desarrollada utilizando microservicios que definen cada uno de los niveles de las reglas de validación. Las reglas del estándar CGMES se han considerado como una especificación de requisitos para evaluar este enfoque. Se ha definido un algoritmo con un enfoque de modelado para obtener sistemáticamente las reglas en SHACL a partir de las reglas basadas en texto. Se ha presentado un ejemplo de regla del estándar CGMES para mostrar la validez del enfoque propuesto.

La comunidad de los sistemas de potencia puede adoptar este enfoque para desarrollar sus aplicaciones en el estándar CIM, ya que las reglas que representan las restricciones del estándar han sido modeladas (implementadas) y son directamente ejecutables. Se pondrá el servicio de validación y las reglas a disposición de cualquier usuario interesado en el dominio de los sistemas de potencia y la ingeniería del software.

Este nuevo enfoque aporta precisión y transparencia al proceso del ciclo de vida del software, ya que todas las fases se modelan y se acuerdan entre el experto en el dominio y el experto en modelado. Este enfoque permite al experto en el dominio comunicar los requisitos, participar en la validación del modelo, permite al experto en modelado ejecutar el modelo y modelar las reglas. El modelado de la regla en SHACL se define sistemáticamente determinando el grafo que conecta los nodos RDF obtenidos de los conceptos identificados en la regla textual.

El desarrollo de un marco que integre todas las fases del ciclo de vida con el enfoque MDE y multiparadigma se propone como trabajo futuro.

11. Conclusiones finales

Dentro de la perspectiva de la ingeniería dirigida por modelos (MDE), el objetivo de este trabajo es alinear los dos principales paradigmas de modelado: el paradigma orientado a objetos y el paradigma ontológico. En este proceso de alineamiento, se busca lograr una convivencia de ambos paradigmas en lugar de reemplazar uno por otro. Sin embargo, es importante tener en cuenta que estos dos enfoques de modelado tienen un conjunto de conceptos que pueden ser mapeados entre sí, pero también existen conceptos que no son mapeables directamente, o al menos no de manera directa. La presente tesis ha constado de tres trabajos.

El primer trabajo se ha centrado en abordar el alineamiento estructural entre los paradigmas de modelado orientado a objetos y ontológico, y ha presentado una metodología para transformar los constructos OCL a los constructos SHACL. Esta metodología se basa en el hecho de que, en ambos paradigmas, tanto el orientado a objetos como el ontológico, un metamodelo está asociado a la definición o representación estructural junto con un conjunto de reglas. Las tecnologías de la web semántica permiten el razonamiento sobre los modelos, ya que se basan en la lógica de descripciones y los modelos obtenidos son grafos con semántica correspondiente al dominio que se está modelando. La evaluación de la metodología de transformación de OCL a SHACL se ha llevado a cabo en el dominio de los sistemas de potencia, en el estándar CGMES, que está construido como un perfil UML a partir del estándar CIM. Este perfil consta de un conjunto de diagramas de clase y un conjunto de reglas definidas en OCL. Con este trabajo se ha proporcionado una solución al problema de validación que el estándar CIM y, en particular, CGMES llevan sin resolver desde sus inicios. De hecho, actualmente ENTSOE está trabajando en la definición de las reglas SHACL para la nueva versión de este estándar. La audiencia de este trabajo abarca un

amplio espectro, desde instituciones encargadas de establecer y mantener estándares, hasta aquellas interesadas en modelar sus estructuras y procesos. Además, se dirige específicamente a la comunidad de ingeniería del software, proporcionándoles una alternativa para llevar a cabo sus tareas de modelado con capacidad de razonamiento y, especialmente, haciendo uso de las tecnologías de la web semántica, las cuales son inherentemente relevantes en el contexto de la ingeniería del software.

En el segundo trabajo se ha abordado la carencia en el estado del arte de un lenguaje visual para la especificación de requisitos cuando se aborda el desarrollo con ontologías. Se han creado unos constructos visuales cuya semántica queda formalmente definida en grafos RDF. El constructo básico es una tarea que va a llevar a cabo una modificación sobre un modelo determinado, cuando se dan unas condiciones definidas en el elemento *precondition* que contendrá las reglas que se deben cumplir sobre un modelo de entrada *inputModel.* Para cubrir las necesidades de la realidad para la definición de comportamiento ha sido necesario modelar los conceptos de tarea, de decisión, de sincronización, y de concurrencia entre otros. En última instancia, el comportamiento se puede conceptualizar como un grafo que engloba los elementos requeridos y adecuados para representar un flujo de ejecución. La ejecución del flujo recae en el procesador de flujo, que se encarga de implementar la semántica de ejecución al recorrer los nodos del grafo correspondiente. Este recorrido ocurre cuando las reglas de *precondition* son validadas en relación con el *inputModel*.

En el tercer trabajo se ha aportado un enfoque con una metodología para las fases tempranas del ciclo de vida del desarrollo del software, considerando primero que en el fondo todas las fases son tareas de modelado, y se han resaltado dos roles que son el experto de modelado y el experto de dominio, este último expone la realidad de su dominio con su correspondiente lenguaje formal que en definitiva es un conjunto de conceptos relacionado sobre los cuales se aplican unas reglas, esto último es en esencia una ontología que el experto de modelado define directamente. Con esto se consigue una mejora en la productividad por el hecho de no hay transformación intermedia entre la especificación de requisitos y la implementación ya que los modelos junto con sus reglas son directamente ejecutables. Los modelos obtenidos son después validados por el experto de dominio contra un conjunto de ejemplares de la realidad de este. Las reglas son distintas en términos de su definición en el contexto del experto del dominio y el experto de modelado ya que el modelo puede contener conceptos de solo utilidad dentro del modelo y al definir las reglas se deben tener en cuenta para una correcta navegabilidad entre los conceptos.

Para trabajos futuros se pueden considerar las siguientes áreas: (i) plantear la exploración de la transformación bidireccional entre SHACL y OCL, así como entre SHACL y otros lenguajes de reglas utilizados en los diversos paradigmas de modelado; esto permitiría ampliar aún más las capacidades de integración y colaboración entre los diferentes enfoques, promoviendo un intercambio efectivo de conocimientos y facilitando la interoperabilidad entre sistemas basados en diferentes paradigmas de modelado; (ii) dado que OMG no ha abordado el mapeo de un numero de conceptos y

constructos de UML a OWL/RDF como por ejemplo el control de acceso en términos de declarar un concepto de solo lectura, privado o público. Se puede llevar estos conceptos al dominio de las ontologías creando los conceptos y relaciones para dar cabida al control de acceso en el dominio de las ontologías; esto conlleva a que los motores por ejemplo de SPARQL tengan conciencia de ello; (iii) crear los conceptos necesarios para abordar en OWL los constructos de composición y agregación ofrecidos por UML; (iv) incorporar las ontologías, especialmente el hecho de que están basadas en grafos con soporte de razonamiento, en el área de los lenguajes de programación de propósito general e intérpretes tanto en la fase de desarrollo como en la fase de ejecución así como la calidad de los distintos artefactos generados; (v) desarrollar un procesador de flujo especifico customizado para un dominio específico para aprovechar las características del dominio y ganar en eficiencia.

Estos trabajos futuros son de gran interés para las instituciones tanto privadas como públicas dado que en la actualidad están muy interesadas en la generación de grafos de datos, que generalmente, se obtienen del legado de la orientación a objetos integrado con datos modelados en entidad-relación. Por lo tanto, se necesita un mapeo bidireccional para integrar el paradigma ontológico en el proceso de ingeniería de software.

12. Acrónimos

BPMN	Business Process Model and Notation
CGM	Common Grid Model
CGMES	Common Grid Model Exchange Standard
CIM	Common information model
CIM	Computation Independent Model
CLP	Constraint logic programming
CTL	Computation tree logic
DSL	Domain Specic Language
ENTSOE	European Network of Transmission System Operators for Electricity
EPC	Event-driven process chain
EQ	Equipment
FOL	First-order logic
HTTP	Hypertext Transfer Protocol
IEC	International Electrotechnical Commission
IGM	Individual Grid Model
IOT	Internet Of Things
JSON	JavaScript Object Notation
M2M	Model to model
M2T	Model to text
MDE	Model driven engineering
MOF	Meta-Object Facility
MVC	Model View Controller
OCL	Object Contrainst Language
ODM	Ontology Definition Metamodel
OMG	Object Management Group
OWL	Web Ontology Language
PIM	Platform Independent Model
PSM	Platform Specific Model
QOCDC	Quality of CGMES Datasets and Calculations
RDF	Resource description framework
RDFS	Resource description framework Schema
RTL	Register-transfer level
SHACL	Shapes Constraint Language
SPARQL	SPARQL Protocol and RDF Query Language
SSH	Steady State Hypothesis
STOMP	Simple Text Oriented Protocol
SV	State Variables
SVA	System Verilog assertion
SWRL	Semantic Web Rule Language
SysML	SysML
TP	Topology
TSO	Domain Specic Language

UML	Unified Modeling Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSD	XML Schema Definition

13. Bibliografía

[1] OMG, «MDA® - The architecture of choice for a changing world,» 2020. [En línea].

[2] T. W. S. W. Group, «SPARQL 1.1 Overview,» 2013.

[3] S. Harris y A. Seaborne, «SPARQL 1.1 Query Language,» 2013.

[4] P. Gearon, A. Passant y A. Polleres, «SPARQL 1.1 Update,» 2013.

[5] G. Williams, «SPARQL 1.1 Service Description,» 2013.

[6] E. Prud'hommeaux y C. B. Aranda, «SPARQL 1.1 Federated Query,» 2013.

[7] A. Seaborne, «SPARQL 1.1 Query Results JSON Format,» 2013.

[8] A. Seaborne, «SPARQL 1.1 Query Results CSV and TSV Formats,» 2013.

[9] J. Broekstra y D. Beckett, «SPARQL Query Results XML Format (Second Edition),» 2013.

[10] B. Glimm y C. Ogbuji, «SPARQL 1.1 Entailment Regimes,» 2013.

[11] K. Clark, G. Williams, L. Feigenbaum y E. Torres, «SPARQL 1.1 Protocol,» 2013.

[12] C. Ogbuji, «SPARQL 1.1 Graph Store HTTP Protocol,» 2013.

[13] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi y P. F. Patel-Schneider, Edits., The Description Logic Handbook: Theory, Implementation, and Applications, New York, NY, USA: Cambridge University Press, 2003.

[14] F. van Harmelen y D. McGuinness, «OWL Web Ontology Language Overview,» 2004.

[15] D. McGuinness y C. Welty, «OWL Web Ontology Language Guide,» 2004.

[16] G. Schreiber y M. Dean, «OWL Web Ontology Language Reference,» 2004.

[17] I. Horrocks, P. Patel-Schneider y P. Hayes, «OWL Web Ontology Language Semantics and Abstract Syntax,» 2004.

[18] J. D. Roo y J. Carroll, «OWL Web Ontology Language Test Cases,» 2004.

[19] J. Heflin, «OWL Web Ontology Language Use Cases and Requirements,» 2004.

[20] I. TopQuadrant, «TopBraid SHACL API,» 2017. [En línea]. Available: https://github.com/TopQuadrant/shacl.

[21] OMG, «About the MOF Query/View/Transformation Specification Version 1.3,» 2016. [En línea].

[22] OMG, «About the Unified Modeling Language Specification Version 2.5.1,» 2017. [En línea].

[23] OMG, «About the Object Constraint Language Specification Version 2.4,» 2017. [En línea].

[24] M. Larhrib, M. Escribano, C. Cerrada y J. J. Escribano, «Converting OCL and CGMES Rules to SHACL in Smart Grids,» *IEEE Access,* vol. 8, p. 177255–177266, 2020.

[25] SWSA, «Semantic Web Science Association,» 2020. [En línea]. Available: https://swsa.semanticweb.org/.

[26] M. Uschold, Y. Ding y P. Groth, Demystifying OWL for the Enterprise, Morgan & Claypool, 2018.

[27] D. E. Khelladi, R. Bendraou, R. Hebig y M.-P. Gervais, «A semi-automatic maintenance and co-evolution of OCL constraints with (meta)model evolution,» *ELSEVIER Journal of Systems and Software,* vol. 134, pp. 242-260, 2017.

[28] M. Hammad, T. Yue, S. Wang, S. Ali y J. F. Nygård, «IOCL: An interactive tool for specifying, validating and evaluating OCL constraints,» *ELSEVIER Science of Computer Programming,* vol. 149, pp. 3-8, 2017.

[29] M. Gogolla, F. Hilken y K.-H. Doan, «Achieving model quality through model validation, verification and exploration,» *ELSEVIER Computer Languages, Systems & Structures,* vol. 54, pp. 474-511, 2018.

[30] N. Przigoda, P. Niemann, J. G. Filho, R. Wille y R. Drechsler, «Frame conditions in the automatic validation and verification of UML/OCL models: A symbolic formulation of modifies only statements,» *ELSEVIER Computer Languages, Systems & Structures,* vol. 54, pp. 512-527, 2018.

[31] B. Pérez y I. Porres, «Reasoning about UML/OCL class diagrams using constraint logic programming and formula,» *ELSEVIER Information Systems,* vol. 81, pp. 152-177, 2019.

[32] M. S. Ángel, J. de Lara, P. Neubauer y M. Wimmer, «Automated modelling assistance by integrating heterogeneous information sources,» *ELSEVIER Computer Languages, Systems & Structures,* vol. 53, pp. 90-120, 2018.

[33] A. G. B. Tettamanzi, C. Faron-Zucker y F. Gandon, «Possibilistic testing of OWL axioms against RDF data,» *ELSEVIER International Journal of Approximate Reasoning,* vol. 91, pp. 114-130, 2017.

[34] Y. Shu, «A practical approach to modelling and validating integrity constraints in the

Semantic Web,» *ELSEVIER Knowledge-Based Systems,* vol. 153, pp. 29-39, 2018.

[35] J. E. L. Gayo, E. Prud'hommeaux, I. Boneva y D. Kontokostas, Validating RDF Data, Morgan & Claypool Publishers, 2017.

[36] J. E. L. Gayo, «RDFShape: RDF playground,» 2017. [En línea].

[37] J. C. Vidal, T. Rabelo, M. Lama y R. Amorim, «Ontology-based approach for the validation and conformance testing of xAPI events,» *ELSEVIER Knowledge-Based Systems,* vol. 155, pp. 22-34, 2018.

[38] C. Martínez-Costa y S. Schulz, «Validating EHR clinical models using ontology patterns,» *ELSEVIER Journal of Biomedical Informatics,* vol. 76, pp. 124-137, 2017.

[39] M. T. Frank, S. Bader, V. Simko y S. Zander, «LSane: Collaborative Validation and Enrichment of Heterogeneous Observation Streams,» *ELSEVIER Procedia Computer Science,* vol. 137, pp. 235-241, 2018.

[40] M. R. A. Rashid, G. Rizzo, M. Torchiano, N. Mihindukulasooriya, O. Corcho y R. García-Castro, «Completeness and consistency analysis for evolving knowledge bases,» *ELSEVIER Journal of Web Semantics,* vol. 54, pp. 48-71, 2019.

[41] K. Nenadic, M. Gavric y V. Djurdjevic, «Validation of CIM datasets using SHACL,» 2017.

[42] S. Biffl y M. Sabou, Semantic Web Technologies for Intelligent Engineering Applications, 1st ed., Springer Publishing Company, Incorporated, 2016.

[43] ENTSO-E, «Common Information Model,» 2019. [En línea].

[44] IEC, «IEC 61970-552:2016,» 2016. [En línea].

[45] IEC, «IEC 61970-501:2006,» 2006. [En línea].

[46] W3C, «Shapes Constraint Language (SHACL),» 2017. [En línea].

[47] IEC, «IEC 61970-301:2016,» 2016. [En línea].

[48] F. S. Parreiras, Semantic web and model-driven engineering, Wiley, 2012.

[49] M. Brambilla, J. Cabot y M. Wimmer, Model-driven software engineering in practice, Morgan & Claypool Publishers, 2017.

[50] G. Dragan, D. Dragan y D. Vladan, Model driven engineering and ontology development, Springer, 2009.

[51] M. Fahad, N. Moalla y Y. Ourzout, «Dynamic Execution of a Business Process via Web Service Selection and Orchestration,» *Procedia Computer Science,* vol. 51, p. 1655–1664, 2015.

[52] M. W. Iqbal, N. A. Ch, S. K. Shahzad, M. R. Naqvi, B. A. Khan y Z. Ali, «User Context Ontology for Adaptive Mobile-Phone Interfaces,» *IEEE Access,* vol. 9, pp. 96751-96762,

2021.

[53] M. Elsayed, N. Elkashef y Y. F.Hassan, «Mapping UML Sequence Diagram into the Web Ontology Language OWL,» *International Journal of Advanced Computer Science and Applications,* vol. 11, 2020.

[54] S. Poslad, S. E. Middleton, F. Chaves, R. Tao, O. Necmioglu y U. Bugel, «A Semantic IoT Early Warning System for Natural Environment Crisis Management,» *IEEE Transactions on Emerging Topics in Computing,* vol. 3, p. 246–257, June 2015.

[55] A. T. Elve y H. A. Preisig, «From ontology to executable program code,» *Computers & Chemical Engineering,* vol. 122, p. 383–394, March 2019.

[56] J. Fabra, M. J. Ibanez, P. alvarez y J. Ezpeleta, «Behavioral Analysis of Scientific Workflows With Semantic Information,» *IEEE Access,* vol. 6, p. 66030–66046, 2018.

[57] B. Teixeira, G. Santos, T. Pinto, Z. Vale y J. M. Corchado, «Application Ontology for Multi-Agent and Web-Services' Co-Simulation in Power and Energy Systems,» *IEEE Access,* vol. 8, p. 81129–81141, 2020.

[58] M. H. Mughal, Z. A. Shaikh, A. I. Wagan, Z. H. Khand y S. Hassan, «ORFFM: An Ontology-Based Semantic Model of River Flow and Flood Mitigation,» *IEEE Access,* vol. 9, p. 44003–44031, 2021.

[59] M. Driss, A. Aljehani, W. Boulila, H. Ghandorh y M. Al-Sarem, «Servicing Your Requirements: An FCA and RCA-Driven Approach for Semantic Web Services Composition,» *IEEE Access,* vol. 8, p. 59326–59339, 2020.

[60] Y. Pradeep, S. A. Khaparde y R. K. Joshi, «High Level Event Ontology for Multiarea Power System,» *IEEE Transactions on Smart Grid,* vol. 3, p. 193–202, March 2012.

[61] C.-W. Yang, V. Dubinin y V. Vyatkin, «Ontology Driven Approach to Generate Distributed Automation Control From Substation Automation Design,» *IEEE Transactions on Industrial Informatics,* vol. 13, p. 668–679, April 2017.

[62] R. Stojanov, S. Gramatikov, I. Mishkovski y D. Trajanov, «Linked Data Authorization Platform,» *IEEE Access,* vol. 6, p. 1189–1213, 2018.

[63] N. F. S. de Sousa, D. A. L. Perez, R. V. Rosa, M. A. S. Santos y C. E. Rothenberg, «Network Service Orchestration: A survey,» *Computer Communications,* Vols. %1 de %2142-143, p. 69–94, June 2019.

[64] S. Isotani, I. Ibert Bittencourt, E. Francine Barbosa, D. Dermeval y R. Oscar Araujo Paiva, «Ontology Driven Software Engineering: A Review of Challenges and Opportunities,» *IEEE Latin America Transactions,* vol. 13, pp. 863-869, 2015.

[65] A. Meidan, J. A. García-García, M. J. Escalona y I. Ramos, «A survey on business processes management suites,» *Computer Standards & Interfaces,* vol. 51, p. 71–86, March 2017.

[66] M. Rashid, M. W. Anwar y A. M. Khan, «Toward the tools selection in model based system engineering for embedded systems—A systematic literature review,» *Journal of Systems*

and Software, vol. 106, p. 150–163, August 2015.

[67] M. M. Mkhinini, O. Labbani-Narsis y C. Nicolle, «Combining UML and ontology: An exploratory survey,» *Computer Science Review,* vol. 35, p. 100223, February 2020.

[68] I. Zafar, F. Azam, M. W. Anwar, B. Maqbool, W. H. Butt y A. Nazir, «A Novel Framework to Automatically Generate Executable Web Services From BPMN Models,» *IEEE Access,* vol. 7, p. 93653–93677, 2019.

[69] P. Valderas, V. Torres y V. Pelechano, «A microservice composition approach based on the choreography of BPMN fragments,» *Information and Software Technology,* vol. 127, p. 106370, November 2020.

[70] S. Appel, P. Kleber, S. Frischbier, T. Freudenreich y A. Buchmann, «Modeling and execution of event stream processing in business processes,» *Information Systems,* vol. 46, p. 140–156, December 2014.

[71] F. Martins y D. Domingos, «Modelling IoT behaviour within BPMN Business Processes,» *Procedia Computer Science,* vol. 121, p. 1014–1022, 2017.

[72] E. Schäffer, V. Stiehl, P. K. Schwab, A. Mayr, J. Lierhammer y J. Franke, «Process-Driven Approach within the Engineering Domain by Combining Business Process Model and Notation (BPMN) with Process Engines,» *Procedia CIRP,* vol. 96, p. 207–212, 2021.

[73] M. W. Anwar, M. Rashid, F. Azam, M. Kashif y W. H. Butt, «A model-driven framework for design and verification of embedded systems through SystemVerilog,» *Design Automation for Embedded Systems,* vol. 23, p. 179–223, November 2019.

[74] M. W. Anwar, M. Rashid, F. Azam, A. Naeem, M. Kashif y W. H. Butt, «A Unified Model-Based Framework for the Simplified Execution of Static and Dynamic Assertion-Based Verification,» *IEEE Access,* vol. 8, p. 104407–104431, 2020.

[75] N. Silega, M. Noguera y D. Macias, «Ontology-based Transformation from CIM to PIM,» *IEEE Latin America Transactions,* vol. 14, p. 4156–4165, September 2016.

[76] I. Lazăr, S. Motogna y B. Pârv, «Behaviour-Driven Development of Foundational UML Components,» *Electronic Notes in Theoretical Computer Science,* vol. 264, p. 91–105, August 2010.

[77] J. A. García-García, J. G. Enríquez y F. J. Domínguez-Mayo, «Characterizing and evaluating the quality of software process modeling language: Comparison of ten representative model-based languages,» *Computer Standards & Interfaces,* vol. 63, p. 52–66, March 2019.

[78] ENTSOE, «Quality of CGMES datasets and calculations for system operations. Second edition,» 2016. [En línea].

[79] W3C, «RDF 1.1 Turtle,» 2014. [En línea].

[80] IEC, «12207-2017 - ISO/IEC/IEEE International Standard - Systems and software

engineering – Software life cycle processes,» 2017. [En línea].

[81] IEEE, «830-1998 - IEEE Recommended Practice for Software Requirements Specifications - IEEE Standard,» 1998. [En línea].

[82] J. Bézivin, «On the unification power of models,» *Software & Systems Modeling,* vol. 4, p. 171–188, 2005.

[83] C. G. Garcia, L. Zhao y V. Garcia-Diaz, «A User-Oriented Language for Specifying Interconnections Between Heterogeneous Objects in the Internet of Things,» *IEEE Internet of Things Journal,* vol. 6, p. 3806–3819, 2019.

[84] N. Macedo, T. Jorge y A. Cunha, «A Feature-Based Classification of Model Repair Approaches,» *IEEE Transactions on Software Engineering,* vol. 43, p. 615–640, January 2017.

[85] Z. a. Y. L. a. H. S. a. X. L. a. Y. Z. Zhi, «UML-based combat effectiveness simulation system modeling within MDE,» *Journal of Systems Engineering and Electronics,* vol. 29, p. 1180, 2018.

[86] D. Akdur, V. Garousi y O. Demirörs, «A survey on modeling and model-driven engineering practices in the embedded software industry,» *Journal of Systems Architecture,* vol. 91, p. 62–82, 2018.

[87] R. A. L. F. KNEUPER, SOFTWARE PROCESSES AND LIFE CYCLE MODELS: an introduction to modelling, using and managing... agile, plan-driven and hybrid processes, SPRINGER, 2019.

[88] R. Kneuper, «Sixty Years of Software Development Life Cycle Models,» *IEEE Annals of the History of Computing,* vol. 39, p. 41–54, 2017.

[89] M. Autili, A. Bertolino, G. D. Angelis, D. D. Ruscio y A. D. Sandro, «A Tool-Supported Methodology for Validation and Refinement of Early-Stage Domain Models,» *IEEE Transactions on Software Engineering,* vol. 42, p. 2–25, January 2016.

[90] M. Larhrib, M. Escribano, C. Cerrada y J. J. Escribano, «Modeling CGMES rules using RDF/OWL/SHACL,» 2020. [En línea]. Available: https://github.com/cimcgmes/cgmes-modeling-shacl.

14. Lista de figuras

15. Lista de tablas

16. Lista de extractos de código