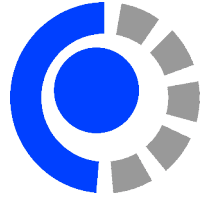




UNIVERSIDAD NACIONAL DEL COMAHUE
FACULTAD DE INFORMÁTICA



REPORTE TÉCNICO
(VERSIÓN PRELIMINAR)

Modelando Ontologías con Patrones en un Ambiente Gráfico Web de Ingeniería Ontológica


Tomás Luciano Quiñonez

NEUQUÉN

ARGENTINA

2022

Índice general

1. Patrones Diseño Ontológicos	1
1.1. Patrones de Diseño Ontológicos	1
1.2. ¿Clases o Tipos? de <i>OPs</i>	2
1.2.1. <i>OPs</i> Estructurales	2
1.2.2. <i>OPs</i> de Correspondencia	4
1.2.3. <i>OPs</i> de Contenido	4
1.2.4. <i>OPs</i> de Razonamiento	6
1.2.5. <i>OPs</i> de Presentación	7
1.2.6. <i>OPs</i> Léxico-Sintáctico	7
2. Metodología Propuesta	9
2.1. Metodología	9
2.2. Ejemplo de Aplicación de la Metodología	11
2.2.1. Ejemplo de consultas realizadas al razonador en <i>OWLlink</i>	14
2.3.  Selección de Patrones	16
2.4. Criterio a utilizar	16
2.5. Selección de Patrones	17
3. Diseño del Prototipo	23
3.1. Arquitectura del Sistema	23
3.2. Preprocesamiento de Patrones	23
3.3. Diseño del Procesador de Nombres	24
3.4. Diseño del Procesador de Axiomas	24
3.5. API del Prototipo	24
4. Implementación del Prototipo	27
4.1. Herramientas y Tecnologías a utilizar	27
4.2. Implementación del Procesador de Axiomas	27
4.3. Cómo se ejecuta el servidor	27
Referencias Bibliográficas	28

Índice de figuras

1.1. Patrón Lógico para relaciones n-arias [8]	3
1.2. Patrón Lógico para relaciones n-arias [8]	3
1.3. Metamodelo de mapeo en OWL [2]	5
1.4. Patrón de contenido <i>Participation</i> [14]	6
1.5. Patrón de contenido <i>Objectrole</i> [13]	6
2.1. Metodología de trabajo	10
2.2. Parte de una ontología de sistemas informáticos	12
2.3. Diagrama para el patrón Computer System	18
2.4. Diagrama para el patrón Reactor	18
2.5. Diagrama para el patrón AlgorithmImplementationExecution	19
2.6. Diagrama para el patrón HazardousSituation	20
2.7. Diagrama para el patrón Course	21
3.1. Diagrama de Componentes de la API	24
3.2. Diagrama de clases del Preprocesador	25
3.3. Diagrama de secuencia para el procesamiento de nombres	26

Índice de tablas

1.1. Tabla de símbolos utilizados por los patrones léxico-sintáctico	7
2.1. Información del patrón <i>Computer System</i>	13
2.2. Información del postprocesamiento de la ontología	13

Capítulo 1

Patrones Diseño Ontológicos

1.1. Patrones de Diseño Ontológicos

Para poder brindar una definición de los Patrones de Diseño Ontológicos (OPs), resulta útil brindar primero una definición genérica de patrones, tan genérica que esta es aplicable a cualquier tipo de patrones en cualquier campo, ciencia o disciplina. Luego, resulta conveniente proveer una definición de Patrones de Diseño en la Orientación a Objetos, puesto que estos poseen similitudes y diferencias con los Patrones de Diseño Ontológicos.

Los patrones en general pueden ser definidos como “Invariantes distintivas y repetitivas a través de datos observados, objetos, procesos, etc., que son creados o causados por el hombre o que ocurren de forma natural”[9]. Una versión más simplificada es la propuesta en [1], que establece que “Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, y luego describe la esencia de la solución a ese problema, de tal manera que la solución puede ser utilizada un millón de veces, sin hacerlo de la misma manera dos veces”. Esta misma definición puede ser utilizada para patrones en el diseño orientado a objetos, en el que las soluciones de esta definición son expresadas en términos de objetos e interfaces, tal como se explica en [7]. Otra definición más cercana a la utilizada para los patrones es la dada en [3], en el que un “Patrón de Diseño de Software describe un problema de diseño recurrente particular que tiene su origen en contextos de diseño específicos, y representa un esquema genérico bien probado para esa solución. El esquema de solución es especificado al describir sus componentes constituyentes, sus responsabilidades y relaciones, y la forma en que colaboran”.

Los Patrones de Diseño Ontológicos son soluciones de modelado para resolver un problema de diseño ontológico recurrente. Esto quiere decir que un OP describe un problema de modelado recurrente en particular, que se origina en distintos contextos en el desarrollo de ontologías y presenta una solución bien probada para dicho problema [5].

Los ODPs son similares a los utilizados en el desarrollo de software, ya que ambos tipos son soluciones para problemas típicos durante la etapa de diseño de software y diseño de ontología, respectivamente. Sin embargo, existen algunas diferencias entre ambos conceptos. Los ODPs, a diferencia de los patrones en el desarrollo de software, no están propiamente vinculados a la fase de desarrollo en el cual pueden ser aplicados. Además, el término “diseño” en “patrones de diseño ontológicos” no tiene el mismo significado que el de “diseño” en “patrones de diseño” para el desarrollo de software. El término “diseño” para ODPs es aplicado en un sentido más genérico, relacionado a la creación (construcción) de una ontología. En el desarrollo de software, por otro lado, el término “diseño” se refiere a la fase de desarrollo de software en el cual los desarrolladores pasan del espacio del problema al espacio de la solución [5].

Luego de haber dado una definición de ODPs, ahora se detallará el uso y la utilidad que estos tienen dentro de la Ingeniería Ontológica al momento de construir una ontología con patrones. Además, se dará una breve descripción de los tipos de patrones que existen y se hará hincapié en un tipo particular de patrones, los Patrones de Contenido o los CPs (Content Patterns).

El principal objetivo de las ontologías es poder expresar formalmente una descripción específica del

mundo. Permiten la descripción de entidades cuyos atributos y relaciones son de interés para el dominio en cuestión. Tienen un gran impacto en la interoperabilidad y la Web Semántica. Sin embargo, un gran problema surge en la Ingeniería Ontológica al desarrollar ontologías, y es el de la *reusabilidad*. Las ontologías ha ser reusadas suelen ser de un tamaño considerable, muy complejas y difíciles de comprender, debiendo el usuario adquirir conocimientos en estructuras lógicas complejas y poco amigables. Luego, el usuario debe realizar un proceso de adaptación costoso (a veces más costoso que crear una ontología desde cero) de esa ontología reusada para cumplir con los requerimientos propios de su dominio. Aunque existan varias herramientas de la Ingeniería Ontológica para el desarrollo de ontologías, estas herramientas presentan dos falencias claramente identificadas. Por un lado, si bien las visualizaciones proveen un adecuado nivel de abstracción para ontologías, ningún método de visualización ha sido ampliamente aceptado. Asimismo, recientes revisiones de la literatura sugieren que los ambientes actualmente disponibles presentan un débil soporte para la Ingeniería Ontológica, debido a la fragmentación de las metodologías en diversas herramientas.

Existen ontologías pequeñas, como *FOAF* [6], que han tenido éxito tanto desde el aspecto de portabilidad como el de la sustentabilidad. Esto permite establecer un importante resultado. En la ingeniería de software, existen los patrones de diseño orientado a objetos, que son definidos como elementos reutilizables en el diseño orientado a objetos. De la misma forma, en la ingeniería ontológica se pueden definir pequeños elementos (ontologías) capaces de ser reutilizados como bloques de construcción para el diseño de ontologías. Dichos bloques son llamados *Content Ontology Design Patterns* (CP), un tipo particular de ODP. Estos bloques pueden ser vistos como pequeñas ontologías bien definidas, usadas como modelos/templates, para ser comparadas con respecto a la ontología definida para el dominio en cuestión[9]. Esto implica una ayuda al usuario durante la etapa de diseño, al reutilizar patrones como guías para determinar si la ontología que está diseñando o reusando, está siendo correctamente desarrollada. Así, los ODPs son importantes ya que promueven el reuso de soluciones a problemas recurrentes de modelado.

Existen otros tipos de *Patrones de Diseño Ontológicos* (OPs), útiles para distintos propósitos y tipos de usuarios. En [8] se identifican seis tipos: Estructural, de Razonamiento, de Correspondencia, de Presentación, Léxico-Sintáctico y los ya mencionados, de Contenido. En la siguiente sección se explicarán y ejemplificarán dichos tipos de ODPs.

1.2. ¿Clases o Tipos? de OPs

En la sección anterior se nombró un tipo específico de OP, llamado *Content Ontology Design Patterns* (CP). En esta sección se pretende definir en profundidad los CPs, junto con los OPs Estructurales, de Correspondencia, Razonamiento, Presentación y Léxico-Sintáctico [8].

1.2.1. OPs Estructurales

Estos tipos de patrones se dividen en dos categorías: los patrones lógicos y los patrones arquitectónicos.

- **OPs Lógicos:** estos patrones son composiciones de construcciones lógicas que resuelven problemas de expresividad. Son expresados solamente en términos de vocabulario lógico, ya que el conjunto de clases y propiedades es vacío. Son independientes de un dominio específico de interés, por otro lado, dependen de la expresividad del formalismo lógico que es usada para la representación. De esta forma, estos patrones ayudan a resolver problemas de diseño en el que las primitivas de la representación del lenguaje no soportan ciertas construcciones lógicas. Por ejemplo, el lenguaje OWL, para la especificación de ontologías para la Web Semántica, solamente soporta relaciones binarias entre elementos, debiendo utilizar un Patrón Lógico para poder representar relaciones n-arias entre elementos en OWL a partir de clases y relaciones binarias. En la Figura 1.1 se puede observar el metamodelo de este Patrón Lógico. La clase *NaryRelationClass* define la relación n-aria y es el dominio de las relaciones binarias que se definirán para cada argumento de esta

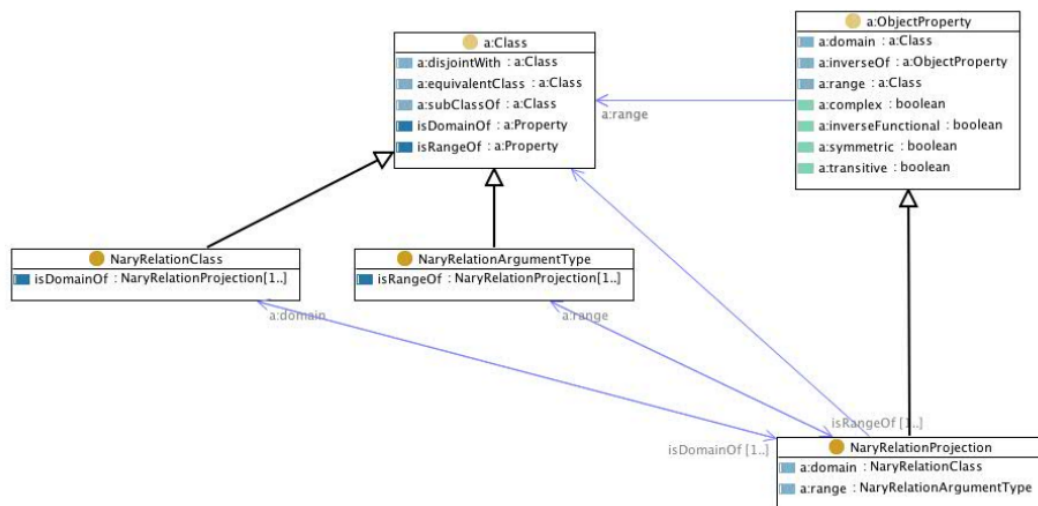


Figura 1.1: Patrón Lógico para relaciones n-arias [8]

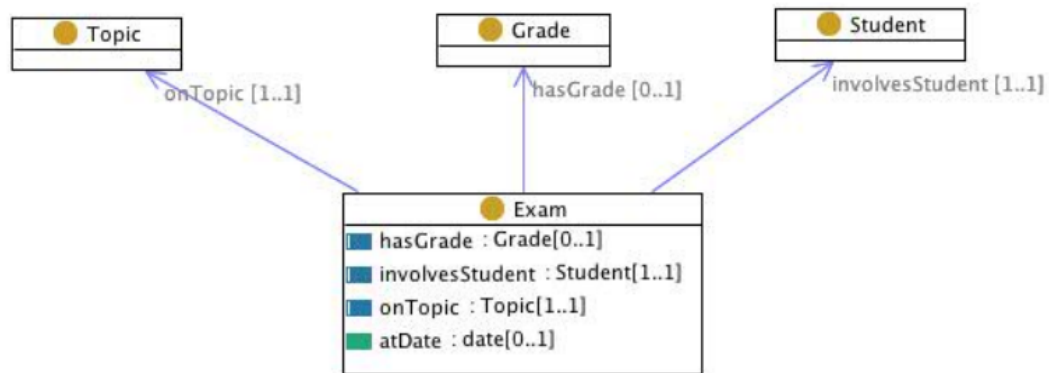


Figura 1.2: Patrón Lógico para relaciones n-arias [8]

relación. La clase `NaryRelationProjection` define las relaciones binarias cuyos rangos son instancias de la clase `NaryRelationArgumentType`, que representan las clases que forman parte del argumento de la relación.

Se puede apreciar un ejemplo de este patrón en la Figura 1.2, en el cual se representa una relación cuaternaria de un examen entre el estudiante, la nota obtenida, el tema del examen y la fecha del mismo. Para cada argumento, se define un relación binaria entre clases (`ObjectProperty`) o una relación entre una clase y un tipo de dato (`DatatypeProperty`), en el que el rango de cada uno de estas relaciones es una clase o un tipo de dato. Dichas relaciones son: `onTopic`, `hasGrade`, `involvesStudent`, `atDate`. Se define la clase `Exam`, que representa la relación cuaternaria entre `Topic`, `Grade`, `Student` y la fecha del examen. Luego, para cada instancia de `Exam` se definen todas estas relaciones, cuyo dominio es `Exam`, y el rango es el argumento de la relación[12].

- **OPs Arquitectónicos:** estos OPs afectan la forma general de una ontología. Están definidos en términos de OPs Lógicos, o composiciones de estos. Son usados en el diseño de una ontología como un todo, al proveer la composición de OPs Lógicos que tienen que ser exclusivamente empleados al diseñar una ontología. Su objetivo es restringir como la ontología debería verse.

1.2.2. OPs de Correspondencia

Esta categoría incluye lo OPs de Reingeniería y de Mapeo.

- **OPs de Reingeniería:** estos OPs proveen soluciones a problemas relacionados con la transformación de un modelo conceptual, que puede no ser un recurso ontológico, en una nueva ontología. Comprenden una serie de reglas de transformación de metamodelo para crear una nueva ontología (modelo objetivo) a partir de un modelo conceptual que puede ser otra ontología, u otro recurso que no sea una ontología, como por ejemplo un modelo UML o una estructura lingüística (modelo origen). Existen dos tipos de estos OPs:
 - **OPs de Reingeniería de Esquemas:** son reglas para transformar metamodelo no-OWL DL en un ontología OWL DL.
 - **Ops de Refactorización:** estos patrones proveen regla para transformar una ontología en OWL DL (modelo origen) en una nueva ontología OWL DL. Estas reglas cambian el tipo de los elementos de la ontología involucrados en la refactorización. Por ejemplo, suponer una ontología que define una relación *prepara_cafe* entre dos conceptos *Agente* y *Cafe*. Si ocurre un cambio de requerimientos y se necesita especificar que un agente prepara un café en una cierta hora utilizando algún utensilio, se necesitaría utilizar el Patrón Lógico **relación n-aria** (definido en la sección 1.2.1), ya que la nueva relación posee cuatro argumentos: agente, café, hora, utensilio. El patrón de refactorización resultante sería el patrón lógico **relación n-aria** junto con la descripción de como aplicarlo para reemplazar la relación definida originalmente.
- **OPs de Mapeo:** comprende las relaciones semánticas entre dos ontologías existentes, sin cambiar los tipos lógicos utilizados, como por ejemplo clases y relaciones. En la Figura 1.3 se puede ver el metamodelo para realizar mapeos entre dos ontologías en OWL. En la clase Mapping se definen dos asociaciones *sourceOntology* y *targetOntology*, que indican las ontologías de dominio y rango del mapeo. A su vez, contiene las asociaciones *elementMappings* con instancias de la clase *MappingAssertion*. Esta clase define las relaciones *sourceElement* y *targetElement* hacia la clase *MappableElement*, representando los elementos de la ontología de origen y objetivo sobre los cuales se quiere crear el mapeo (como por ejemplo dos clases, una definida en la ontología de origen y la otra en la ontología objetivo). El tipo de relación semántica será definida por *SemanticRelation* [2], la cual puede ser:
 - **Equivalence (\equiv):** indica que los elementos relacionados representan el mismo aspecto del mundo real de acuerdo con algún criterio de equivalencia.
 - **Containment (\sqsubseteq, \sqsupseteq):** indica que el elemento en una ontología representa un aspecto del mundo más específico que el elemento en la otra ontología.
 - **Overlap (\circ):** indica que los elementos relacionados representan diferentes aspectos del mundo real, pero pueden superponerse en otro aspecto. En otras palabras, establece que algunos objetos descritos en el elemento de una ontología puede también ser descrito por el elemento relacionado en la otra ontología.

1.2.3. OPs de Contenido

Los CPs son usados como componentes de modelado, permitiendo que una ontología sea el resultado de la composición de CPs, junto con la expansión necesaria para satisfacer requerimientos propios del problema a resolver. Además, al proveer soluciones a problemas orientados a dominios, son directamente reusables. Los OPs no dependen de ninguna representación del lenguaje específica (excepto por los patrones lógicos). Por otro lado, existen CPs específicos para crear ontologías dedicadas a la Web Semántica, llamados OWL CPs. Estos patrones son muy útiles para dicha web, ya que satisfacen muy

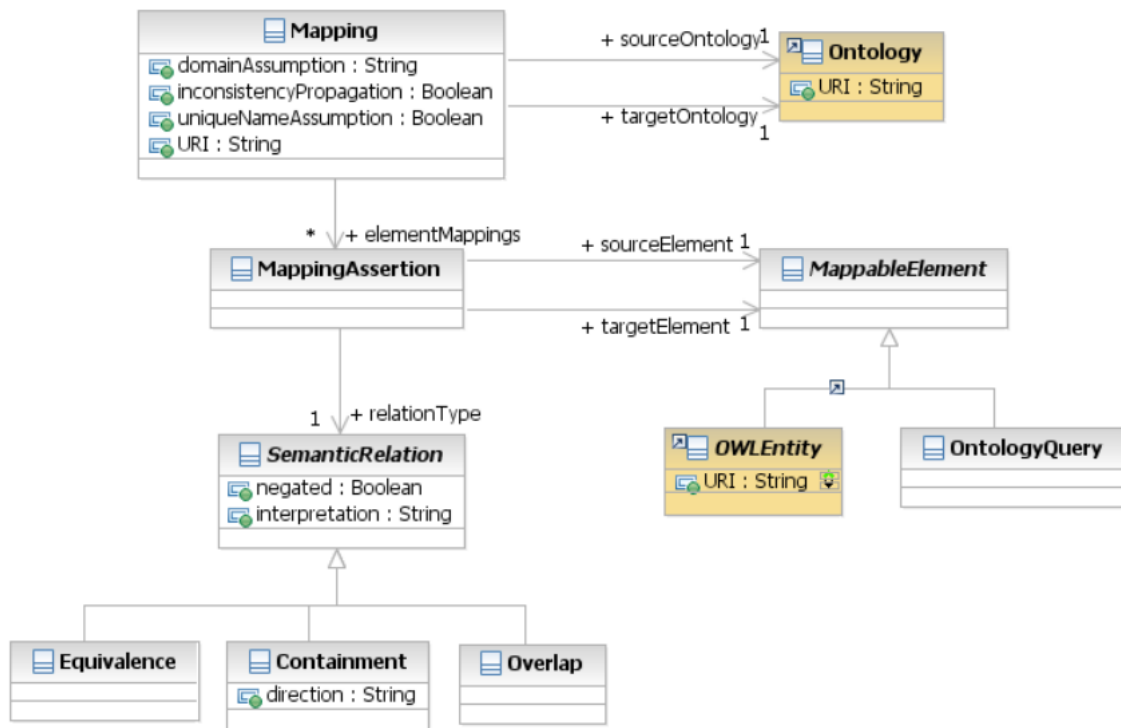


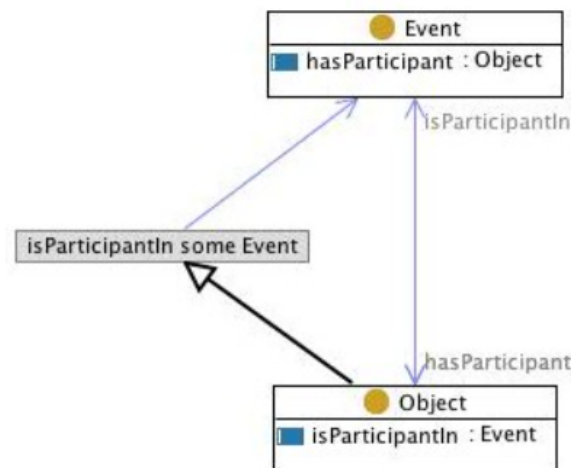
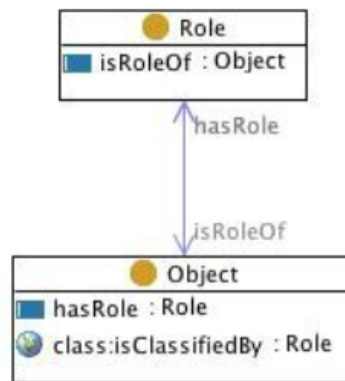
Figura 1.3: Metamodelo de mapeo en OWL [2]

bien los requerimientos de reuso e interoperabilidad de ontologías y datos.

Estos patrones se centran en resolver problemas de diseño con respecto a clases y propiedades de un dominio en particular, y por lo tanto resolver problemas de contenido. Estos se diferencian, por ejemplo, de los patrones lógicos, que resuelven problemas de diseño independientes de una conceptualización en particular. Por lo tanto, los patrones de contenido pueden ser vistos como instanciaciones de dichos patrones lógicos (o composiciones de estos), al proveer vocabulario no lógico para un dominio de interés en particular, independiente de cualquier lenguaje. Sin embargo, deben ser implementados en algún lenguaje para su reuso como *bloques de construcción*. Tal lenguaje, en el contexto de la Web Semántica, será OWL.

Los patrones de contenido son ontologías que proveen soluciones a ciertas preguntas de competencia. Estas representan el problema que las ontologías buscan resolver. Dichas preguntas suelen ser consultas que un experto puede realizar a una base de conocimiento en un dominio en particular. Los diseñadores de ontologías deberían especificar todas las conceptualizaciones necesarias para responder a las preguntas de competencia formuladas por expertos. Además, poseen ciertas características que los distinguen de las ontologías comunes; son considerados ontologías:

- **Computacionales:** son independientes del lenguaje utilizados para su implementación. Sin embargo, son representados en el lenguaje OWL para su reutilización en la Web Semántica.
- **Pequeñas y autónomas:** estas ontologías generalmente poseen entre dos y diez clases, junto con las relaciones entre estas. A su vez, facilitan la tarea a los diseñadores ya que componer estas ontologías les permite controlar la complejidad de la ontología que se desea construir
- **Jerárquicas:** un patrón de contenido puede ser parte de un orden parcial, donde se requiere al menos una clase o propiedad en el patrón sea especializada. Una jerarquía de CPs puede ser construida al especializar o generalizar algunos de sus elementos (clases o propiedades)
- **Cognitiva y Lingüísticamente relevantes:** la visualización de estas ontologías es intuitiva y com-

Figura 1.4: Patrón de contenido *Participation* [14]Figura 1.5: Patrón de contenido *Objectrole* [13]

pacta, y deben captar nociones relevantes del dominio. Además, muchas de estas ontologías se corresponden con patrones lingüísticos, llamados *frames*

- **Las mejores prácticas:** estas ontologías deberían ser usadas para describir las mejores prácticas de modelado. Estas mejores prácticas son desarrolladas por expertos, que surgen de aplicaciones reales. La calidad de estas ontologías se basan en la experiencia personal de los diseñadores de las mismas y de la fuente de conocimiento de donde se origina.

Un ejemplo de este tipo de patrones es el patrón *Participation*, visto de forma gráfica en la Figura 1.4. Este patrón busca resolver las preguntas de competencia: ¿Qué objetos participan en que eventos? y ¿En que evento participa un objeto? De forma coloquial, este patrón busca representar la participación del objeto *Object* en el evento *Event*, a través de la relación binaria *isParticipantIn* y su relación inversa *hasParticipant*.

Otro ejemplo sencillo es el patrón *Objectrole*, que representa el rol que tiene un determinado objeto. Resuelve las preguntas de competencia: ¿Qué rol juega un objeto? y ¿Qué objetos juegan un rol?. En la Figura 1.5 se puede ver de forma gráfica la pequeña ontología.

1.2.4. OPs de Razonamiento

Estos OPs son aplicaciones de patrones lógicos orientados a obtener ciertos resultados de razonamiento, basados en el comportamiento implementado en un motor de razonamiento. Informan sobre el

Tabla 1.1: Tabla de símbolos utilizados por los patrones léxico-sintáctico

CD	Cardinal Number
CATV	Verbs of classification {classify in/into, comprise in, contain in, compose of, group in/into, divide in/into, fall in/into, belong to}
CN	Class Name: generic name for class type plus preposition {class of, group of, type of, member of, subclass of, category, etc.}
NEG	No, not
NP	Noun Phrase
PARA	Paralinguistic symbol, such as colon (:)
PREP	Preposition
RPRO	Relative Pronoun {that, which, whose...}
VB	Verb, base form

estado de una ontología, y dejan que el sistema decida que razonamiento tiene que ser realizado en la ontología para poder procesar consultas, evaluaciones, etc. Un ejemplo de estos patrones son las *normalizaciones*; consisten de un conjunto de reglas que al aplicarlas a la ontología, esta se considera normalizada. Una ontología normalizada es vista como un pre-procesamiento de la misma para poder aplicarle distintas métricas estructurales con un sentido semántico [15].

1.2.5. OPs de Presentación

Estos OPs están relacionados con la usabilidad y legibilidad de ontologías desde la perspectiva del usuario. Son consideradas buenas prácticas que soportan el reuso de ontologías al facilitar su evaluación y elección. Existen dos tipos de estos patrones:

- OPs de Nombre: comprenden convenciones sobre como crear espacio de nombres, elementos de una ontología, como nombres de clases y relaciones. Mejoran la legibilidad y el entendimiento por humanos.
- OPs de Anotación: proveen propiedades de anotaciones que pueden ser usados para mejorar el entendimiento de ontologías y sus elementos, como por ejemplo definir etiquetas significativas para clases y propiedades.

1.2.6. OPs Léxico-Sintáctico

Comprenden estructuras lingüísticas que consisten de ciertos tipos de palabras que siguen un orden específico, permitiendo generalizar y extraer conclusiones sobre el significado que expresan. Permiten asociar patrones lógicos y de contenido con sentencias en lenguaje natural. Estos patrones han sido formalizados utilizando notaciones empleados para describir la sintaxis de lenguajes, como la notación BNF. Por ejemplo, un patrón correspondiente a expresar la relación de herencia entre dos clases podría ser:

NP<subclass> be NP<superclass>

Donde NP es una Frase Nominal que aparece antes de un verbo en su forma básica (be) seguido de otra frase nominal. Si tenemos una clase *Animal* y una subclase de esta llamada *Perro*, se podría generar la expresión lingüística “Los perros son animales”. En la Tabla 1.1 se muestran las descripciones de los símbolos utilizados por los patrones léxicos-sintáctico.

Capítulo 2

Metodología Propuesta

2.1. Metodología

En la metodología propuesta, un Módulo Detector de Patrones toma como entrada la ontología del usuario. Se decidió utilizar dicho lenguaje de marcado debido a que como es necesario procesar la información de la ontología, este lenguaje facilita esta tarea debido a su poder de expresión y vocabulario ya definido [11], superior a otros lenguajes para representar ontologías, como lo son *XML*, *RDF* y *RDF Schema*.

Posteriormente, el Módulo Detector de Patrones realizará un análisis para determinar qué patrones se detectan en la misma. Para realizar esto, dicho módulo contendrá dos submódulos.

El primer submódulo, llamado **Submódulo de Preprocesamiento de Patrones (SPP)**, tiene como objetivo la extracción de información relevante de los patrones para la detección de los mismos en la ontología del usuario. Un Procesador de Nombres extraerá los nombres de todos los elementos presentes en todos los patrones, y un Procesador de Axiomas extraerá todos los axiomas de dichos patrones. Debido a que el vocabulario que utiliza un usuario para modelar un dominio puede diferir con respecto al vocabulario utilizado en los patrones de diseño, aún siendo el mismo dominio, este submódulo utilizará una estructura de datos que contiene una serie de sinónimos para los nombres de los elementos empleados para cada uno de los patrones previamente cargados en la herramienta. De esta forma, si un elemento de la ontología tiene un nombre distinto al usado en un patrón pero que coincide con algún sinónimo dentro de la estructura de datos, se considerará que dicho elemento del patrón ocurre dentro de la ontología. Además, durante esta etapa se irá contando y almacenando la cantidad de nombres de clases, de relaciones y axiomas presentes en los patrones para luego ser utilizado por el submódulo siguiente. Luego, con los nombres, axiomas y lista de sinónimos extraídos de los patrones, se procede a formar dos grupos de consultas. Un grupo contendrá todas las consultas de nombres y de sinónimos a realizar al razonador; y el segundo grupo contendrá todas las consultas de axiomas presentes en los patrones.

A partir de la información obtenida del submódulo anteriormente mencionado y de la ontología del usuario, se procede a formular consultas que se le harán a razonadores. El segundo submódulo, llamado **Submódulo Analizador**, realizará primero el Matching de Nombres, en el que se tomará primero como entrada las consultas de nombres, junto con la ontología y se le realizarán dichas consultas a un primer razonador. Esta tarea realizará una comparación de los nombres de los elementos involucrados en la ontología con cada uno de los nombres de los elementos de los patrones para encontrar coincidencias. Inicialmente, se consulta por los nombres presentes en los patrones. En caso de que no exista el nombre, se preguntará de forma iterativa por los sinónimos hasta encontrar alguna coincidencia. También, se consultará si existe algún nombre en la ontología que sea subcadena de algún nombre presente en el patrón.

A medida que se obtienen los resultados de las consultas, se irán agregando entradas a una estructura Diccionario. Las entradas son pares de la forma (nombre, nombre/sinónimo). Esta estructura será utilizada posteriormente para el Matching de Axiomas para verificar primero que los elementos de un axioma están presentes en la ontología, y para reformular las consultas en base a los sinónimos en caso de ser

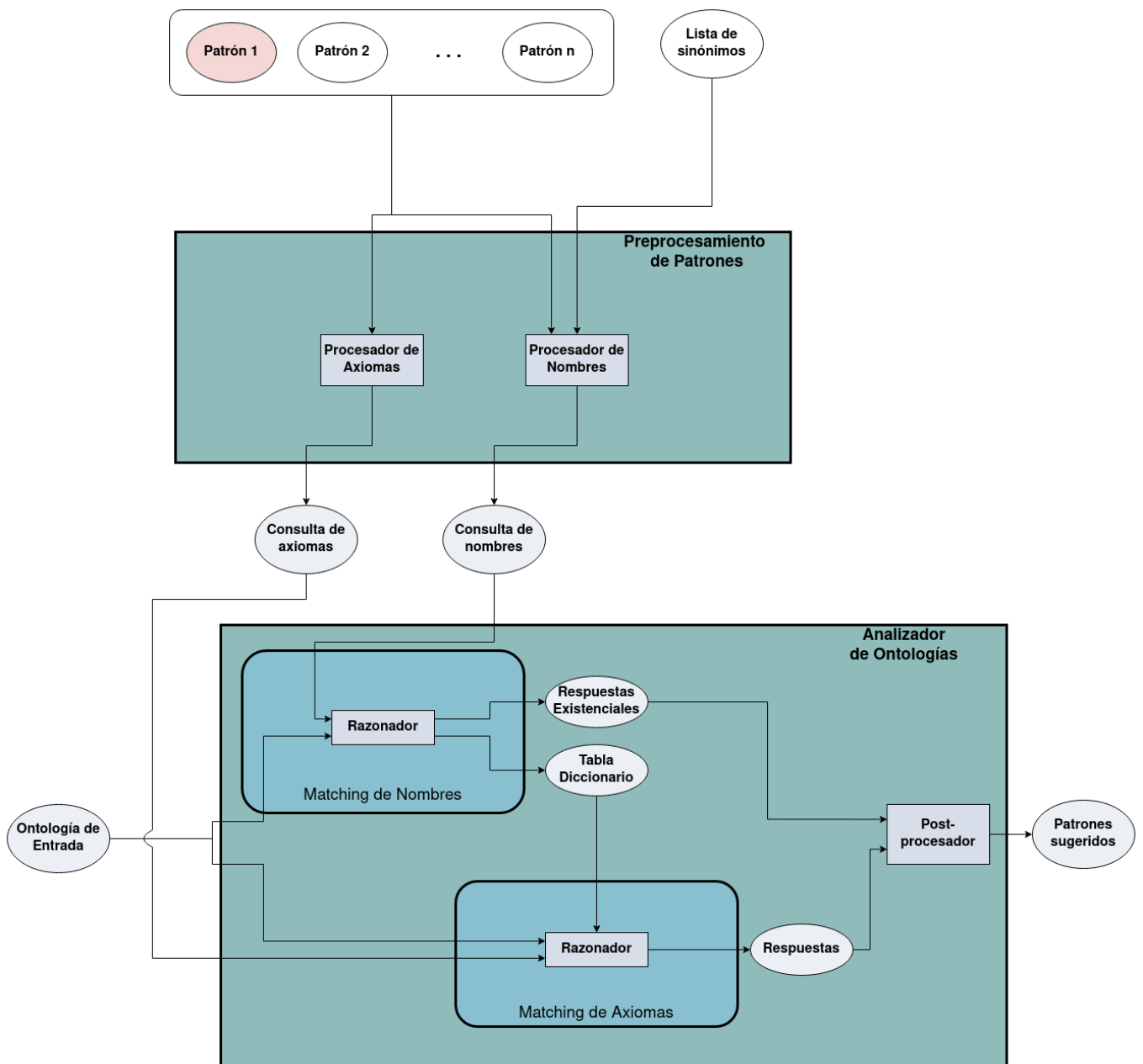


Figura 2.1: Metodología de trabajo


necesario.

La segunda tarea, llamada Matching de Axiomas, involucra la comparación de la estructura de los axiomas que existen en la ontología con los axiomas de los patrones, esto incluye: cardinalidades de las relaciones entre clases, tipos de las relaciones de herencia (disjunta, covering), tipos de datos, etc., realizando los cambios necesarios según el Diccionario poblado durante la tarea de Matching de Nombres.

La segunda consulta es sobre la comparación de la estructura de los axiomas que existen en la ontología con los axiomas de los patrones, esto incluye: cardinalidades de las relaciones entre clases, tipos de las relaciones de herencia (disjunta, covering), tipos de datos, etc.

Estas dos tareas son similares a dos de los tres tipos de análisis (Léxico y Estructural) realizados en [16] para detectar patrones en una ontología. Es importante destacar que los patrones a detectar serán cargados previamente en la herramienta, en algún lenguaje de marcado. Dichos patrones son los propuestos en [4], conteniendo todos los axiomas de dichos patrones.

Finalmente, luego de este análisis, se hará un postprocesamiento de los resultados obtenidos para cada patrón. Se realizarán dos tipos de evaluaciones que la ontología debe superar para ser sugerida al usuario. Primero, el porcentaje de clases del patrón presente en la ontología debe de ser mayor o igual a un umbral en particular (inicialmente un 50 %). Si es menor que este umbral, el patrón no es sugerido y se procede a analizar otro patrón. En caso contrario, se procede a la segunda evaluación. Para cada patrón se calculan 3 puntajes individuales para la cantidad de nombres de clases, nombres de relaciones y de axiomas que están presentes en la ontología. Cada puntaje se calcula transformando el porcentaje de aciertos de cada grupo en un número real entre 0 y 10. Luego, el puntaje total se obtiene haciendo un promedio de estos puntajes individuales. Para que el patrón sea sugerido, el puntaje total mínimo deberá de ser de 2,3. Se eligió este valor debido a que como la presencia de clases de un patrón en la ontología tiene mayor relevancia que la presencia de nombres de relaciones y de axiomas a la hora de ser sugerido al usuario, el patrón será sugerido si existe al menos un 70 % $((7 + 0 + 0)/3)$ de clases en la ontología (puntaje individual de 7), aún cuando no existan relaciones ni axiomas del patrón en dicha ontología. Finalmente se retornará una lista con los patrones sugeridos que ocurren en la ontología junto con sus puntajes asociados.

C:  Expandir y citar. Cambiar crowd → crowd – 06/12/2021

Seguir la metodología en papel para lograr el objetivo.

1. Selección de un patrón simple de contenido
2. Ver qué preguntas se le haría a una ontología para detectar un **patrón parcial**: qué partes del patrón le permitimos faltar en una ontología. Y a partir de esto ver qué consultas le haríamos al razonador.

2.2. Ejemplo de Aplicación de la Metodología

En la Figura 2.2 se puede ver parte de una ontología cuyo dominio es sobre un sistema informático. A continuación se mostrará como la herramienta detectaría el patrón *Computer System*, cuyo UML se puede apreciar en la Figura 2.3.

Primero, el Módulo Detector de Patrones toma como entrada la ontología del usuario en algún formato específico (inicialmente en formato *OWL*). Luego, dentro de dicho módulo, el Submódulo de Preprocesamiento de Patrones (SPP) realiza la tarea de extraer información útil de cada uno de los patrones cargados previamente en una estructura de datos. Para simplificar la explicación en este ejemplo, el submódulo tendrá cargado solamente el patrón *Computer System*. Para realizar la extracción de información, dos procesadores realizarán la extracción de nombres y de axiomas de los patrones, correspondientemente. El Procesador de Nombres recorrerá el patrón *Computer System* y extraerá los siguientes nombres: *Operating System*, *Software*, *Driver*, *Hardware*, *requiresSoftware*, etc. Además, la Lista de Sinónimos contendrá los sinónimos para los siguiente nombres:

- Software: SW

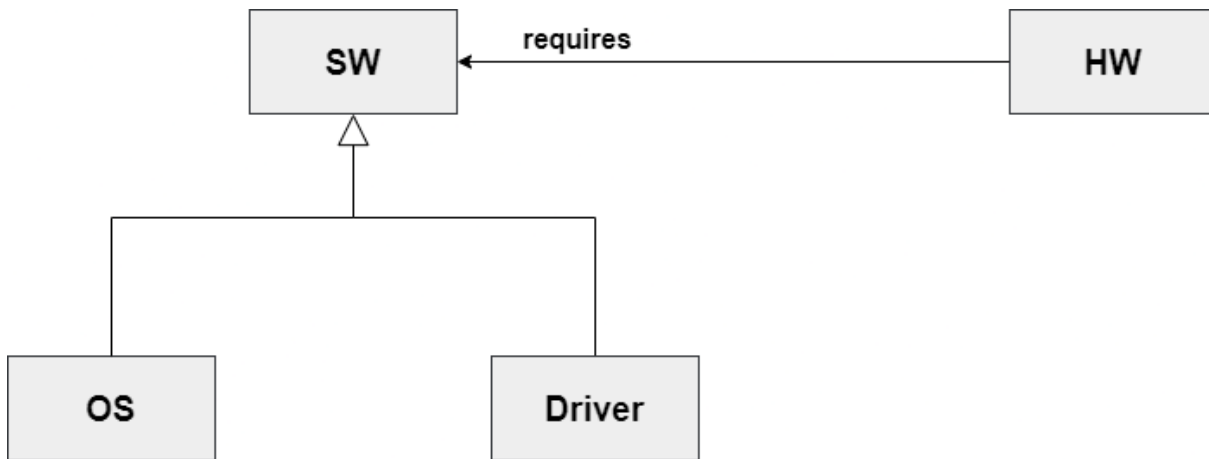


Figura 2.2: Parte de una ontología de sistemas informáticos

- Hardware: HW
- Driver: Controlador
- Operating System: OS

El Procesador de Axiomas extraerá todos los axiomas presentes en el patrón. Para esto, se explorará el árbol *XML* del patrón y se buscarán etiquetas que definan axiomas para subclases, relaciones, restricciones de dominio y rango, uso de operadores existenciales, etc. Luego, estas sentencias serán adecuadas para poder realizar consultas al razonador. Para nuestro ejemplo, se extraen, entre otros, los siguientes axiomas:

- `Operating System` es subclase de `Software`.
- `Driver` es subclase de `Software`.
- `Hardware` se relaciona con la clase `Software` a través de la relación `requiresSoftware`.

A partir de esto, se formulan las consultas que se realizarán a los razonadores sobre la ontología del usuario para detectar la presencia de los elementos de los patrones en dicha ontología. Las consultas de nombres son las siguientes:

1. a) ¿Existe una clase llamada `Operating System`?
- b) ¿Existe una clase llamada `OS`?
2. a) ¿Existe una clase llamada `Software`?
- b) ¿Existe una clase llamada `SW`?
3. ¿Existe una clase llamada `Driver`?
4. a) ¿Existe una clase llamada `Hardware`?
- b) ¿Existe una clase llamada `HW`?
5. a) ¿Existe una relación llamada `requiresSoftware`?
- b) ¿Existe una relación que contenga la palabra `requires`?

De la misma forma, las consultas de axiomas iniciales serán las siguientes:

1. La clase `Operating System`, ¿es subclase de `Software`?

Tabla 2.1: Información del patrón *Computer System*

Patrón	Clases	Relaciones	Axiomas
<i>Computer System</i>	5	7	15

Tabla 2.2: Información del postprocesamiento de la ontología

Matcheo de patrones	Clases	Relaciones	Axiomas	Sugerencia
<i>Computer System</i>	4/5 80 % Puntaje: 8	1/7 15 % Puntaje: 1, 5	3/15 20 % Puntaje: 2	Sí Puntaje Total: 3, 8

2. La clase *Driver*, ¿es subclase de *Software*?

3. La clase *Hardware*, ¿se relaciona con la clase *Software* a través de la relación *requiresSoftware*?

Después de la etapa de preprocesamiento, se procede a realizar el análisis con la información obtenida de los patrones y la ontología del usuario. Primero, se realizará el Matching de Nombres, buscando ocurrencias de nombres (o sinónimos) en la ontología. Se realizan las consultas formuladas anteriormente al razonador y se obtienen las siguientes respuestas, llenando con entradas el Diccionario en el proceso:

- Respuesta a la pregunta 1a: **No**.
- Respuesta a la pregunta 1b: **Sí**, se guarda el par (*Operating System*, *OS*).
- Respuesta a la pregunta 2a: **No**.
- Respuesta a la pregunta 2b: **Sí**, se guarda el par (*Software*, *SW*).
- Respuesta a la pregunta 3a: **Sí**, se guarda el par (*Driver*, *Driver*).
- Respuesta a la pregunta 4a: **No**.
- Respuesta a la pregunta 4b: **Sí**, se guarda el par (*Hardware*, *HW*).
- Respuesta a la pregunta 5a: **No**.
- Respuesta a la pregunta 5b: **Sí**, se guarda el par (*requiresSoftware*, *requires*).

Es importante señalar que si el nombre original de un elemento ocurre en la ontología, las preguntas sobre los sinónimos de dicho elemento no se realizan, como ocurre con la consulta 3b.

Una vez obtenidas las respuestas a las consultas de nombres y llenado el Diccionario, se procede a realizar el matching de axiomas, modificando en cada caso, las consultas a realizar al razonador:

- Para la consulta 1, se detectaron en el diccionario las entradas para *Operating System* y *Software*, por lo tanto, la consulta a realizar se reformula como: La clase *OS*, ¿es subclase de *SW*?, cuya respuesta es **Sí**.
- Para la consulta 2, se detectaron en el diccionario las entradas para *Driver* y *Software*, por lo tanto, la consulta a realizar se reformula como: La clase *Driver*, ¿es subclase de *SW*?, cuya respuesta es **Sí**.
- Para la consulta 3, se detectaron en el diccionario las entradas para *Hardware*, *Software* y *requiresSoftware*, por lo tanto, la consulta a realizar se reformula como: La clase *HW*, ¿se relaciona con la clase *SW* a través de la relación *requires*?, cuya respuesta es **Sí**.

Finalmente, al obtener las respuestas a las consultas de nombres y de axiomas, se realiza el post-procesamiento para determinar si se debe sugerir o no los patrones para la ontología del usuario. En la Tabla 2.1 se muestra la información de preprocesamiento obtenida para el patrón *Computer System* que será utilizada por el postprocesador. A partir de las respuestas obtenidas de las consultas de nombres y de axiomas realizadas al razonador sobre la ontología del usuario, se determina que 4 de las 5 clases (50 %) existen en la ontología, 1 de las 7 relaciones (15 %) y 3 de los 15 axiomas (20 %) existen en la ontología; como se puede apreciar en la Tabla 2.2. Los puntajes individuales para nombres de clases, nombres de relaciones y de axiomas es 8, 1, 5, y 2, respectivamente. Debido a que el umbral de respuestas positivas para nombres de clases debe ser de al menos 50 %, se considera que en este punto, el patrón ocurre parcialmente en la ontología, y se procede a calcular el puntaje del patrón, cuyo valor mínimo debe ser de 2,3. Como este valor es de 3,8, el Postprocesador determina que el patrón *Computer System* debe ser sugerido y lo agrega en la lista de Patrones Sugeridos junto con su puntaje para luego ser retornado al usuario.

2.2.1. Ejemplo de consultas realizadas al razonador en *OWLink*

En esta subsección se mostrarán las consultas elaboradas en la sección anterior en formato *OWLink* [10] que se pasarán al razonador *Konclude*¹. Además, se mostrarán y explicarán las respuestas retornadas por dicho razonador.

Para las consultas de nombres, al consultar por la satisfacibilidad de una clase, la interfaz *OWLink* retorna un mensaje booleano *true* si la clase es satisfacible; y en caso contrario retorna un mensaje de error. Debido a que resulta complejo manejar el error en caso de que no exista una clase con un nombre en particular, es preferible consultar al razonador por todos los nombres de las clases de la ontología del usuario y delegar la tarea de verificar si la clase existe o no al **Postprocesador**. Este mismo procedimiento se realizará para las relaciones entre las clases. Por lo tanto, las consultas que se realizarán al razonador serán: *GetAllClasses*, que retorna un conjunto de todas las clases de la ontología; y *GetAllObjectProperties*, que retorna un conjunto con todas las relaciones de la ontología. A continuación se muestra un fragmento de *XML* de las posibles consultas:

```
1 <GetAllClasses kb="http://www.owllink.org/ont/sistema" />
2 <GetAllObjectProperties kb="http://www.owllink.org/ont/sistema" />
```

Las respuestas retornadas serán dos conjuntos: un conjunto de clases (*SetofClasses*), conteniendo los nombres de todas las clases de la ontología, y otro conjunto (*SetofObjectProperties*) con los nombres de todas las relaciones de la ontología; como se aprecia en el siguiente fragmento de código *XML*:

```
1 <SetOfClasses>
2   <owl:Class IRI="#Thing"/>
3   <owl:Class IRI="#Nothing"/>
4   <owl:Class IRI="#Driver"/>
5   <owl:Class IRI="#HW"/>
6   <owl:Class IRI="#OS"/>
7   <owl:Class IRI="#SW"/>
8 </SetOfClasses>
9 <SetOfObjectProperties>
10  <owl:ObjectProperty IRI="#topObjectProperty"/>
11  <owl:ObjectProperty IRI="#bottomObjectProperty"/>
12  <owl:ObjectProperty IRI="#requires"/>
13 </SetOfObjectProperties>
```

¹Ver <https://www.derivo.de/en/produkte/konclude.html> para consultar la pagina oficial del razonador.

Para las consultas de axiomas se utiliza la consulta `IsEntailed`. Con esta consulta, se consulta al razonador si un axioma está implicado en la ontología.

Para la primer consulta: La clase `Driver`, ¿es subclase de `SW`?, se consulta por la existencia del axioma `SubClassOf`, como se puede observar a continuación:

```
1 <IsEntailed kb="http://www.owllink.org/ont/sistema">
2   <owl:SubClassOf>
3     <owl:Class IRI="#OS" />
4     <owl:Class IRI="#SW" />
5   </owl:SubClassOf>
6 </IsEntailed>
```

La respuesta retornada será un mensaje booleano con el valor `true`, indicando que dicho axioma existe en la ontología:

```
1 <BooleanResponse result="true"/>
```

De la misma forma, para la consulta: La clase `OS`, ¿es subclase de `SW`?:

```
1 <IsEntailed kb="http://www.owllink.org/ont/sistema">
2   <owl:SubClassOf>
3     <owl:Class IRI="#Driver" />
4     <owl:Class IRI="#SW" />
5   </owl:SubClassOf>
6 </IsEntailed>
```

Al retornar un valor booleano verdadero, se indica que el axioma existe en la ontología:

```
1 <BooleanResponse result="true"/>
```

Para el caso de la consulta: La clase `HW`, ¿se relaciona con la clase `SW` a través de la relación `requires`?, se formulará dicha consulta primero de la siguiente forma (en *Description Logic*):

$$HW \sqsubseteq \exists \text{requires}.SW$$

En *OWLlink*:

```
1 <IsEntailed kb="http://www.owllink.org/ont/sistema">
2   <owl:SubClassOf>
3     <owl:Class IRI="#HW" />
4     <owl:ObjectSomeValuesFrom>
5       <owl:ObjectProperty IRI="#requires" />
6       <owl:Class IRI="#SW" />
7     </owl:ObjectSomeValuesFrom>
8   </owl:SubClassOf>
9 </IsEntailed>
```

Otra posible elaboración para la consulta anterior es restringiendo el dominio y el rango de la relación, por lo que se preguntará si existe la definición de restricción de dominio (`ObjectPropertyDomain`) y de rango (`ObjectPropertyRange`) para la relación `requires`, como se muestra a continuación:

```

1 <IsEntailed kb="http://www.owllink.org/ont/sistema">
2   <owl:ObjectPropertyRange>
3     <owl:ObjectProperty IRI="#requires"/>
4     <owl:Class IRI="#SW"/>
5   </owl:ObjectPropertyRange>
6 </IsEntailed>
7
8 <IsEntailed kb="http://www.owllink.org/ont/sistema">
9   <owl:ObjectPropertyDomain>
10    <owl:ObjectProperty IRI="#requires"/>
11    <owl:Class IRI="#HW"/>
12  </owl:ObjectPropertyDomain>
13 </IsEntailed>

```

Por como está definida la ontología, la respuesta para la consulta con la primera formulación es verdadera:

```

1 <BooleanResponse result="true"/>

```

2.3. Selección de Patrones

La selección de patrones a ser sugeridos juega un papel crucial en la etapa de implementación de la recomendación de patrones a un usuario que está desarrollando una ontología. Esto se debe a que existen ciertos patrones de contenido que no resultan estar bien definidos. Ejemplos de una mala definición de un patrón son: no poseer un diagrama que describa la estructura del mismo, no tener preguntas de competencia claras y concisas, etc. Por otra parte, existen patrones que son muy genéricos e independientes de algún dominio en particular, provocando que sea dificultoso la comparación de estos patrones con una ontología para un dominio muy específico. Además, se deben tener en cuenta otros factores para la elección de patrones, como lo es la probabilidad de que usuarios desarrollen ontologías para dominios similares a los de los patrones. Patrones con dominios muy particulares y que contemplan casos muy particulares de su dominio tienen muy poca probabilidad de ser recomendados a un usuario. Debido todas estas y otras cuestiones, resulta imprescindible la elaboración de un criterio para la elección de estos patrones para ser integrados a la herramienta. En este capítulo se presenta un criterio de elección de patrones, en el que todos los patrones que se pretendan integrar a la herramienta deben superar exitosamente este criterio. Además, se presenta un listado con todos los patrones que han sido vistos y analizados pero no han cumplido con alguna parte del criterio anteriormente mencionado. Para cada patrón seleccionado, se mostrará su correspondiente diagrama (*UML*, *MER*, etc.), propósito de su elaboración y preguntas que el patrón pretende resolver.

2.4. Criterio a utilizar

En esta sección se presenta un criterio para selección de patrones que serán implementados en la herramienta de detección de patrones en la etapa de implementación de dicha herramienta. Los patrones serán seleccionados y evaluados según el criterio a partir de una lista de patrones de contenido publicados².

- Probabilidad de desarrollar una ontología para el dominio. El dominio para el cual se pretende utilizar el patrón debería ser frecuentemente usado dentro de la informática o dentro del ámbito académico.

²Ver <http://ontologydesignpatterns.org/wiki/Category:ProposedContentOP> para el listado de patrones de contenido.

- El patrón no debería tener una fuerte dependencia con su dominio. Si los elementos (clases, relaciones, etc.) del patrón son demasiado dependientes del dominio en cuestión, resultaría complejo poder detectar los mismos en una ontología con dominios similares.
- Para reducir la complejidad temporal en la detección de patrones y aumentar la probabilidad de detectarlos en la ontología del usuario, los patrones a seleccionar deben tener como máximo diez clases y diez relaciones. Sin embargo, tampoco deben ser demasiados simples, es decir, deberían contener al menos cuatro clases y cuatro relaciones.
- Para facilitar la detecciones de patrones en una ontología, estos no deben ser difíciles de expandir. Debe ser posible agregar nuevas clases/relaciones/axiomas para adecuar el patrón al dominio de interés sin tener que realizar muchos cambios dentro del patrón.
- Los patrones deben estar bien definidos. Un patrón bien definido implica que haya un diagrama de dicho patrón (de clases, Entidad Relación, etc.) claro, tenga preguntas de competencia claras y concisas y posea explicación de los elementos del patrón.
- El patrón no debe tener más de quince preguntas de competencia definidas y sus respuestas deben ser concretas y no deben ser ambiguas.

2.5. Selección de Patrones

A continuación se muestra una lista de los nombres de todos los patrones evaluados que no han cumplido con uno o más puntos del criterio presentado en la subsección 2.4.

Lista de patrones evaluados y descartados: *Time indexed person role*³, *PrivacyPolicyPersonalData*⁴, *Pollution*⁵, *Policy*⁶, *AgentRole*⁷, *ActivitySpecification*⁸, *DataTransformationPattern*⁹.

Los patrones que han sido evaluados y han cumplido con todos los puntos del criterio son presentados y explicados a continuación.

ComputerSystem

El propósito de este patrón es intentar modelar sistemas informáticos basándose en una aproximación hardware/software. En la Figura 2.3 se presenta el diagrama de clases del patrón. Las preguntas de competencia asociadas al patrón son las siguientes: ¿Qué está instalado actualmente en una computadora? Una computadora usa piezas específicas de hardware y de software. ¿Qué software y hardware se requiere para que otro software o hardware opere apropiadamente? Esta relación es representada con las propiedades `requiresSoftware` y `requiresHardware`. ¿Qué software o hardware es compatible con otro software o hardware? Esto es expresado a través de la relación `isCompatibleWith`.

Reactor

En la Figura 2.4 se presenta el diagrama de clases del patrón.

³Ver http://ontologydesignpatterns.org/wiki/Submissions:Time_indexed_person_role para más información sobre el patrón.

⁴Ver <http://ontologydesignpatterns.org/wiki/Submissions:PrivacyPolicyPersonalData> para más información sobre el patrón.

⁵Ver <http://ontologydesignpatterns.org/wiki/Submissions:Pollution> para más información sobre el patrón.

⁶Ver <http://ontologydesignpatterns.org/wiki/Submissions:Policy> para más información sobre el patrón.

⁷Ver <http://ontologydesignpatterns.org/wiki/Submissions:AgentRole> para más información sobre el patrón.

⁸Ver <http://ontologydesignpatterns.org/wiki/Submissions:ActivitySpecification> para más información sobre el patrón.

⁹Ver <http://ontologydesignpatterns.org/wiki/Submissions:DataTransformationPattern> pa-

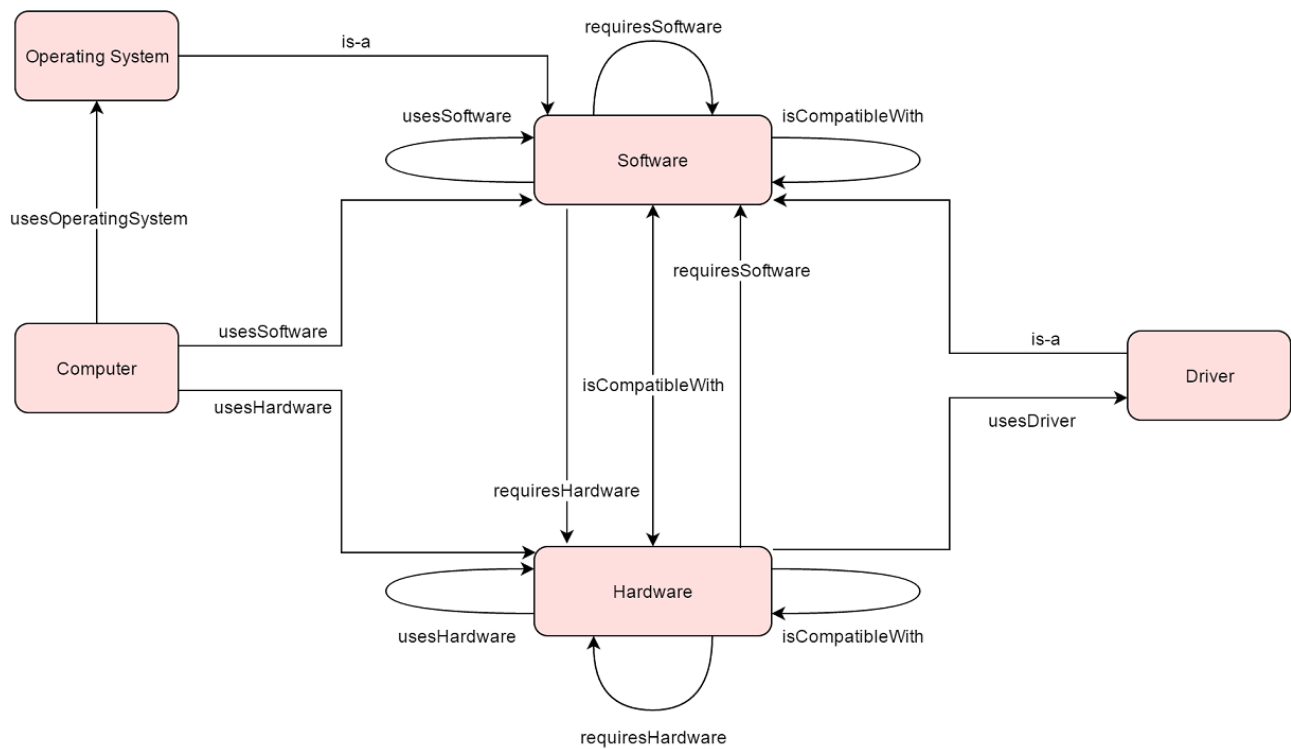


Figura 2.3: Diagrama para el patrón Computer System

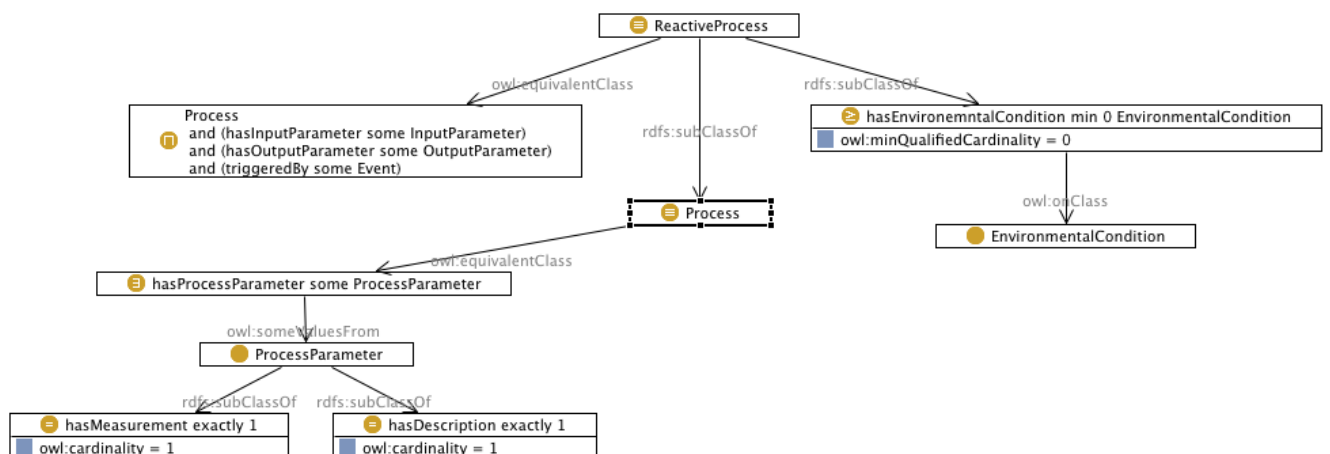


Figura 2.4: Diagrama para el patrón Reactor

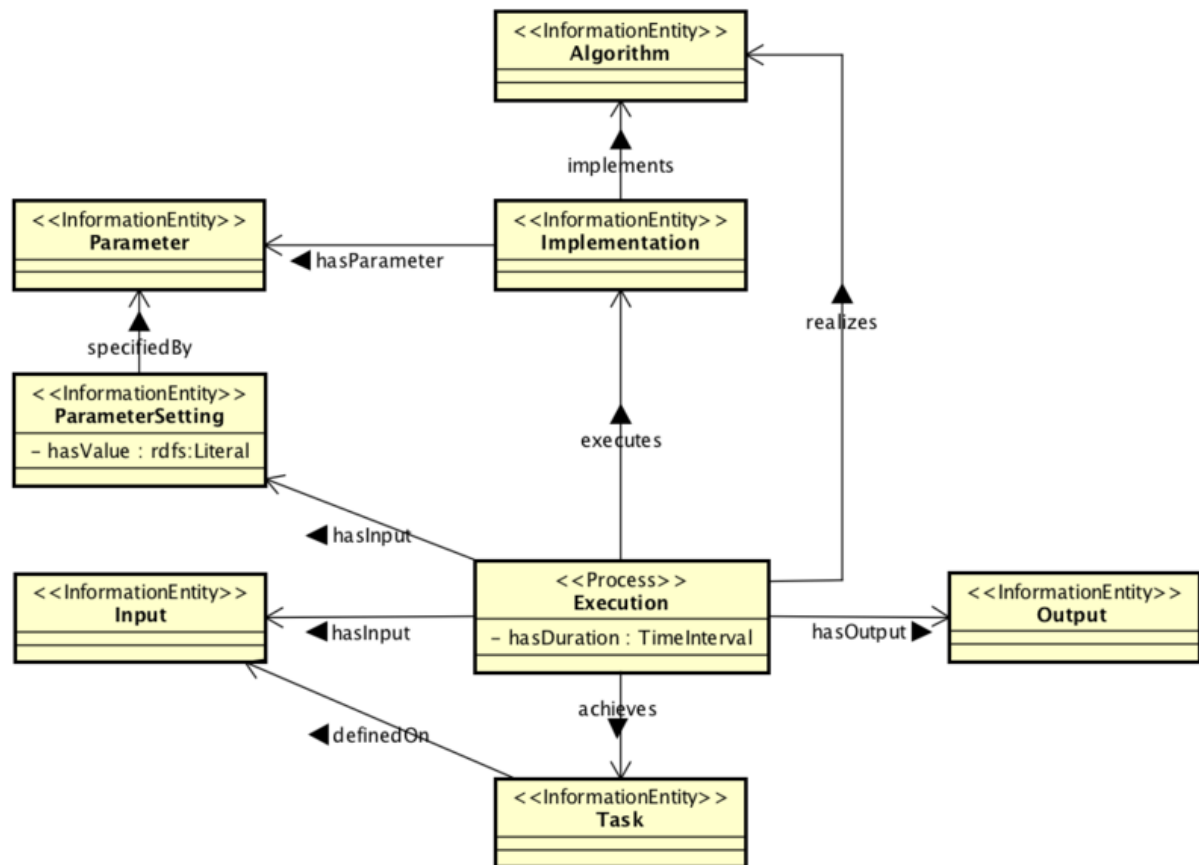


Figura 2.5: Diagrama para el patrón AlgorithmImplementationExecution

El propósito de este patrón es permitir el modelado de procesos reactivos que consumen entradas y producen salidas bajo condiciones ambientales específicas y al ser activados por ciertos eventos. Las preguntas de competencia asociadas al patrón son las siguientes: ¿Cuales son las entrada consumidas por un proceso en particular? ¿Qué condiciones ambientales se necesitan satisfacer para que el proceso sea activado? ¿Cuales son las salidas producidas por el proceso? ¿Qué evento activa un proceso específico? ¿Cual es el criterio de medición para un parámetro específico?

AlgorithmImplementationExecution

En la Figura 2.7 se presenta el diagrama de clases del patrón. El propósito de este patrón es modelar especificaciones de algoritmos, sus implementaciones y ejecuciones, junto con parámetros de implementación, configuraciones de dichos parámetros, y las entradas que consume la ejecución y las salidas que produce. Las preguntas de competencia asociadas al patrón son las siguientes: ¿Qué algoritmo es implementado por una implementación? ¿Cuales son las implementaciones para un algoritmo? ¿Cual implementación es ejecutada? ¿Cuales son los parámetros para una implementación? ¿Cuales son los parámetros de configuración para parámetros en particular en una ejecución? ¿Cual es la entrada para la ejecución de una implementación? ¿Cual es la salida producida por la ejecución de una implementación? ¿Qué algoritmo realiza una ejecución? ¿Qué tarea realiza una ejecución? ¿Cual es la duración de una ejecución? ¿Sobre que entrada está definida una tarea?

HazardousSituation

En la Figura 2.6 se presenta el diagrama de clases del patrón. El propósito de este patrón es modelar

ra más información sobre el patrón.

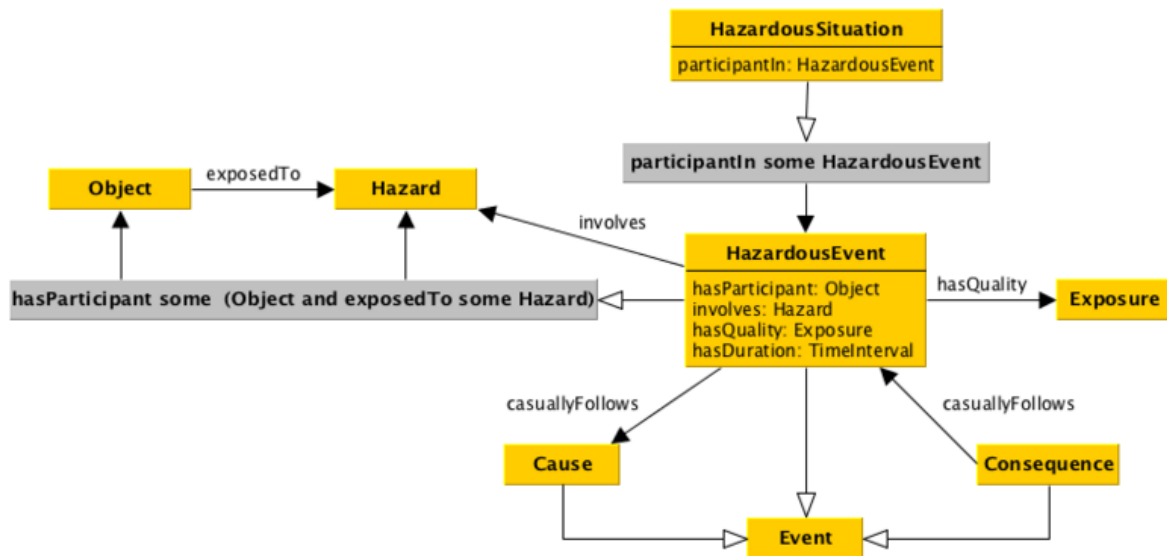


Figura 2.6: Diagrama para el patrón HazardousSituation

situaciones peligrosas y sus eventos peligrosos asociados con objetos que participan de los eventos y los peligros a los que están expuestos los objetos con un valor de exposición. Las preguntas de competencia asociadas al patrón son las siguientes: ¿Qué objeto está expuesto a un peligro? ¿A que peligro está expuesto un objeto? ¿Qué eventos peligrosos están asociados con una situación peligrosa? ¿Cual es la causa de un evento peligroso? ¿Cual es la consecuencia de un evento peligroso? ¿Cual es el valor de exposición de un objeto expuesto a una amenaza?

Course

En la Figura 2.7 se presenta el diagrama de clases del patrón.

El propósito de este patrón es modelar los atributos principales de un curso y las relaciones básicas de dicho curso en una institución educacional. Las preguntas de competencia asociadas al patrón son las siguientes: ¿Cuál es el nombre de un curso? ¿Cuál es el número de un curso? ¿Quién es el instructor de un curso? ¿Quién está realizando un curso?

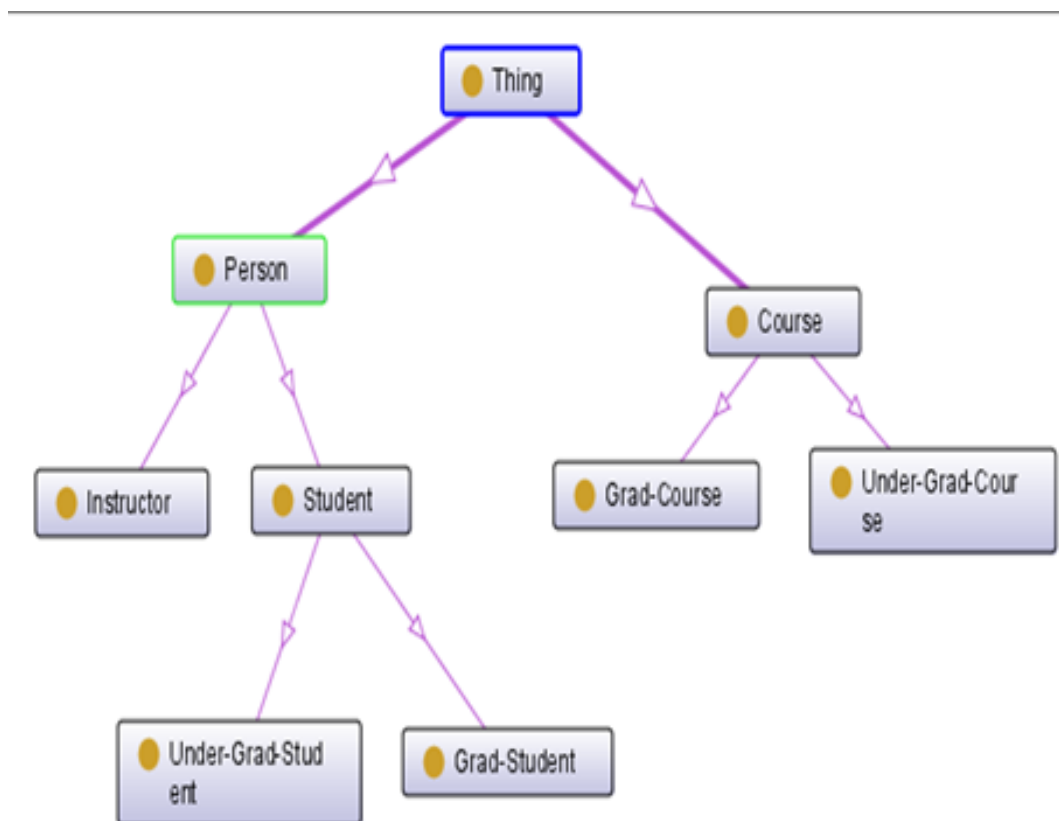


Figura 2.7: Diagrama para el patrón Course

Capítulo 3

Diseño del Prototipo

En este capítulo se presentará el diseño de los componentes que conforman la herramienta de detección de patrones. Debido a que se utilizará una solución orientada a objetos, se hará uso del lenguaje unificado de modelado (*UML*) para graficar los distintos componentes, haciendo uso de distintos diagramas, como los diagramas de componentes, de clases y de secuencia. Primero, se mostrará el diseño general de la API y luego el diseño de los módulos de Preprocesamiento de Patrones y luego el diseño del módulo de Análisis de Patrones con la ontología del usuario.

3.1. Arquitectura del Sistema

El Servidor Web estará conformado por dos componentes llamados `Pattern Preprocessor Web Component` y `Analyzer Web Component`. El primero realizará todas las tareas relacionadas con el Preprocesamiento de Patrones. Esto incluye: tomar como entrada la ontología del patrón y una lista de sinónimos para los nombres que ocurren en dicha ontología, extraer todos los nombres relevantes del patrón, obtener todos los axiomas presentes, almacenar dicha información para ser utilizada posteriormente. El segundo componente realizará todas las actividades relacionadas con el Análisis de Ontologías. Tomará como entrada la ontología del usuario y generará como salida un listado de todos los patrones identificados en dicha ontología. Para lograr esto, se ha diseñado la API `PreprocessorResultsAPI`, que será utilizada internamente dentro del Servidor Web por este componente, para obtener los resultados elaborados por el componente `Pattern Preprocessor Web Component` de todos los patrones analizados. Esta información será utilizada para realizar el análisis de la ontología, comparando dicha información con la obtenida por el primer componente y de esta forma, poder encontrar ocurrencias de patrones dentro de la ontología del usuario.

Se han diseñado dos APIs para la comunicación de los clientes con el Servidor Web. La API `PatternPreprocessorAPI` proveerá mensajes para agregar nuevos, eliminar del servidor patrones agregados previamente, obtener una lista con todos los patrones y encontrar un patrón en particular. La API `AnalyzerAPI` aceptará el mensaje para poder sugerir patrones al usuario a partir de una ontología de entrada.

3.2. Preprocesamiento de Patrones

En la Figura 3.2 se presenta el diseño del Preprocesamiento de Patrones. Las clases encargadas de del procesamiento de nombres y de axiomas son `NameProcessor` y `AxiomProcessor` respectivamente. Ambas clases redefinen el comportamiento de la clase `Preprocessor`, de acuerdo al tipo de extracción de información que necesiten realizar.

Previamente al preprocesamiento de patrones, es necesario crear las instancias correspondientes de los repositorios de archivos *OWL* y de patrones. De esta tarea se encargará otro componente del sistema, el cual para cada archivo que contenga un patrón, creará una instancia de `OWLontology`. Esta clase está definida dentro de la `OWLAPI` y será utilizada para la extracción de información del patrón. Para

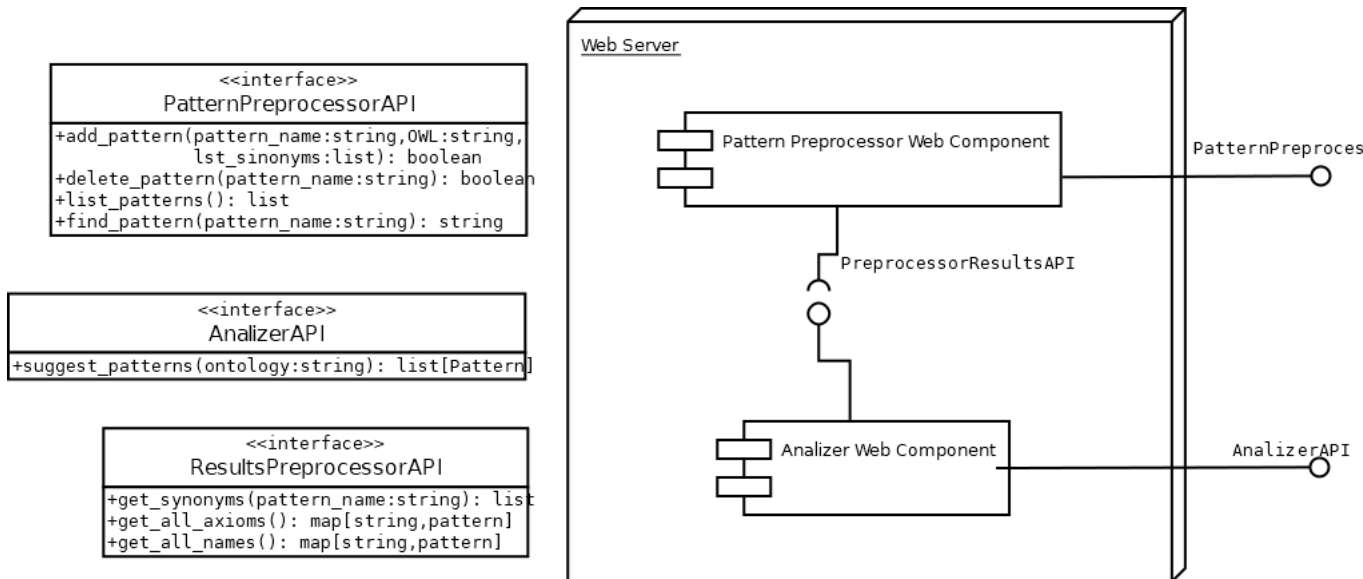


Figura 3.1: Diagrama de Componentes de la API

cada una de estas instancias, se creará una instancia de la clase `Pattern`, la cual contendrá el nombre del patrón y las consultas de nombres y de axiomas correspondientes. Dichas consultas se obtendrán a medida que se realice el preprocesamiento de los patrones.

EL funcionamiento general tanto de `NameProcessor` como de `AxiomProcessor` consiste en analizar cada uno de los objetos `OWLontology` almacenados en `OWLontologyRepository`, extraer la información necesaria y almacenar dicha información en la instancia correspondiente de la clase `Pattern`. Debido a que las consultas de axiomas son más complejas que la de nombres, se diseñó una clase `AxiomQuery` que contiene la información de un axioma detectado durante el preprocesamiento.

3.3. Diseño del Procesador de Nombres

En esta sección se mostrarán las decisiones de diseño realizadas para el Procesador de Nombres de la herramienta. En la Figura 3.3 se muestra el diagrama de secuencia para el método `processPatterns()` redefinido por la clase `NameProcessor` para la extracción de nombres de los patrones.

El proceso de extracción de nombres inicia obteniendo una lista con todos los objetos `OWLontology` del repositorio `OWLontologyRepository`. Iterativamente se extrae de cada instancia el nombre del patrón, los nombres de cada una de las clases que conforman la ontología, y los nombres de los roles entre las clases. Luego, utilizando la IRI de la ontología, se obtiene la correspondiente instancia de `Pattern`, para agregar la información extraída. Los sinónimos de cada patrón serán insertados en un archivo para su posterior extracción. El Procesador de Nombres es el encargado de extraer los sinónimos de cada patrón del archivo e insertarlos en la instancia de `Pattern` correspondiente. Para realizar esto, se utiliza la IRI obtenida del objeto `OWLontology` para buscar la lista de sinónimos en el archivo de sinónimos y extraerlos.

3.4. Diseño del Procesador de Axiomas

3.5. API del Prototipo

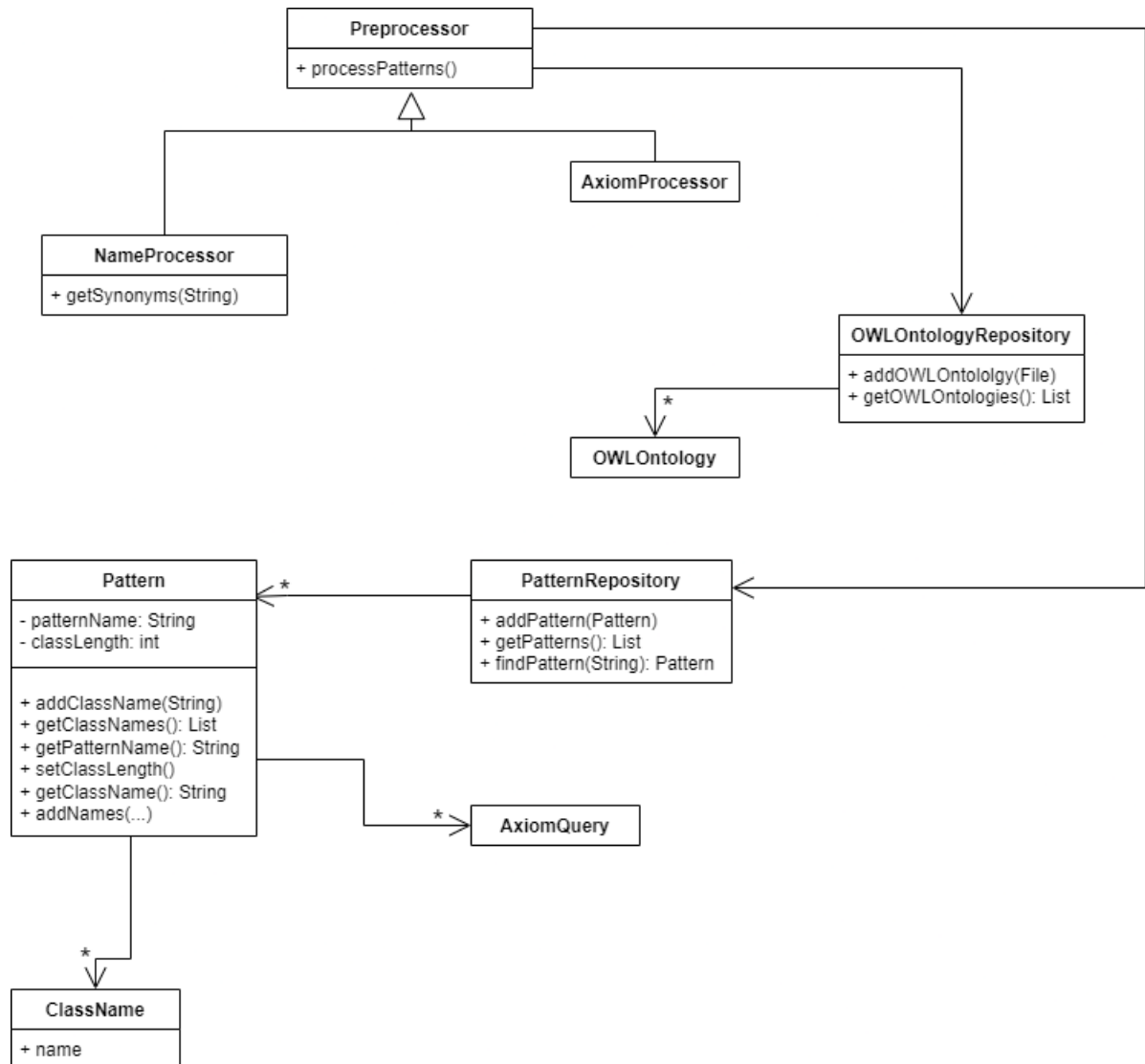


Figura 3.2: Diagrama de clases del Preprocesador

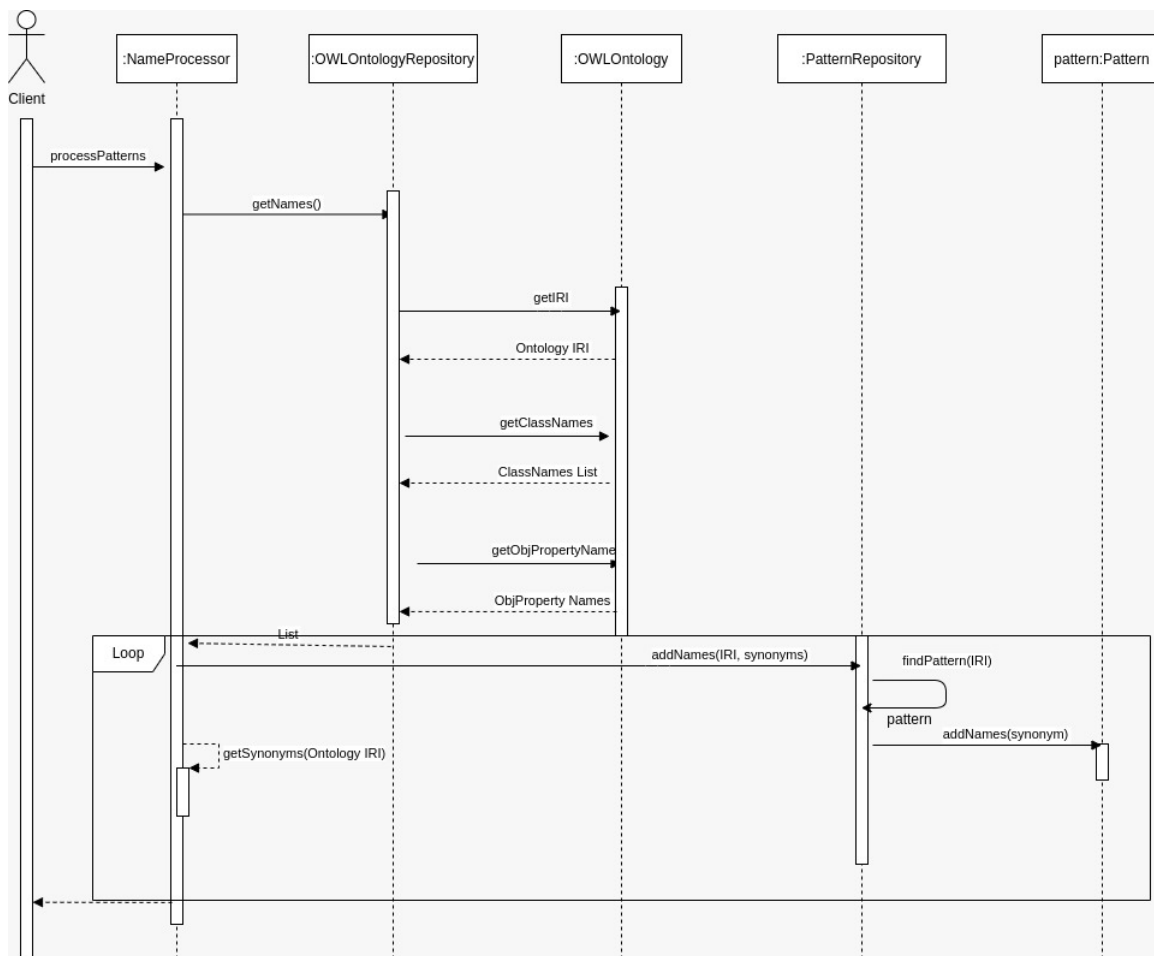


Figura 3.3: Diagrama de secuencia para el procesamiento de nombres

Capítulo 4

Implementación del Prototipo

4.1. Herramientas y Tecnologías a utilizar

El desarrollo de la herramienta fue realizado utilizando el lenguaje de programación *Java* con el framework de desarrollo *Spring Boot*¹. La creación y gestión del proyecto fue realizado con *Apache Maven*², el cual es una herramienta de desarrollo de software de código abierto para la construcción y manejo de proyectos con *Java*. Además, para el manejo de ontologías, tanto de usuarios como de los patrones de diseño ontológicos, se utilizó *OWL API*³, una API de Java para la creación, manipulación y serialización de ontologías.

Para testear dicha API, se utilizó la plataforma *Postman*⁴, que permite definir un cliente HTTP para realizar pruebas del tipo *HTTP requests* y por lo tanto hacer pruebas automatizadas sobre APIs en desarrollo.

4.2. Implementación del Procesador de Axiomas

4.3. Cómo se ejecuta el servidor

Para la compilación y ejecución del servidor se necesitará contar con las siguientes herramientas:

- Java JDK (17.0.1)
- Apache Maven (3.8.3)

Luego, se deberá clonar el repositorio y ejecutar los siguientes comandos en desde la carpeta raíz del proyecto:

- `$ mvn clean install`
- `$ mvn spring-boot:run`

¹<http://spring.io/projects/spring-boot>

²<http://maven.apache.org/>

³<https://www.postman.com/>

⁴<http://ontologydesignpatterns.org/wiki/Category:ProposedContentOP>

Referencias Bibliográficas

- [1] Christopher Alexander. *A pattern language: towns, buildings, construction*. Oxford university press, 1977 (vid. pág. 1).
- [2] Franz Baader, Ian Horrocks, Carsten Lutz y Uli Sattler. *Introduction to description logic*. Cambridge University Press, 2017 (vid. págs. 4, 5).
- [3] Frank Buschmann, Kevlin Henney y Douglas C Schmidt. *Pattern-oriented software architecture, on patterns and pattern languages*. Vol. 5. John wiley & sons, 2007 (vid. pág. 1).
- [4] *Category:ProposedContentOP*. <http://ontologydesignpatterns.org/wiki/Category:ProposedContentOP>. Visitado: 2021-06-20 (vid. pág. 11).
- [5] R de A Falbo, Giancarlo Guizzardi, Aldo Gangemi y Valentina Presutti. «Ontology patterns: clarifying concepts and terminology». En: *Proceedings of the 4th Workshop on Ontology and Semantic Web Patterns*. 2013 (vid. pág. 1).
- [6] *FOAF Vocabulary Specification 0.99*. <http://xmlns.com/foaf/spec/>. Visitado: 2021-06-19 (vid. pág. 2).
- [7] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides y Design Patterns. «Elements of Reusable Object-Oriented Software». En: *Design Patterns. massachusetts: Addison-Wesley Publishing Company* (1995) (vid. pág. 1).
- [8] Aldo Gangemi y Valentina Presutti. «Ontology design patterns». En: *Handbook on ontologies*. Springer, 2009, págs. 221-243 (vid. págs. 2, 3).
- [9] Pascal Hitzler, Aldo Gangemi y Krzysztof Janowicz. *Ontology engineering with ontology design patterns: foundations and applications*. Vol. 25. IOS Press, 2016 (vid. pág. 1).
- [10] Liebig, Thorsten and Luther, Marko and Noppens, Olaf and Wessel, Michael. «OWLlink». En: *Semantic Web – Interoperability, Usability, Applicability 2.1* (2011), págs. 23-32 (vid. pág. 14).
- [11] *OWL Web Ontology Language - Overview*. <https://www.w3.org/TR/owl-features/>. Visitado: 2022-01-10 (vid. pág. 9).
- [12] Valentina Presutti, Aldo Gangemi, Stefano David, Guadalupe Aguado de Cea, Mari Carmen Suárez-Figueroa, Elena Montiel-Ponsoda y Maria Poveda. «A library of ontology design patterns: reusable solutions for collaborative design of networked ontologies». En: *NeOn deliverable D 2* (2008) (vid. pág. 3).
- [13] *Submissions:ObjectRole*. <http://ontologydesignpatterns.org/wiki/Submissions:Objectrole>. Visitado: 2021-06-15 (vid. pág. 6).
- [14] *Submissions:Participant*. <http://ontologydesignpatterns.org/wiki/Submissions:Participation>. Visitado: 2021-06-15 (vid. pág. 6).
- [15] Denny Vrandečić y York Sure. «How to design better ontology metrics». En: *European Semantic Web Conference*. Springer. 2007, págs. 311-325 (vid. pág. 7).
- [16] Maleeha Arif Yasvi y Raghava Mutharaju. «ODPREco-A Tool to Recommend Ontology Design Patterns». En: *WOP@ ISWC*. 2019 (vid. pág. 11).