



UNIVERSIDAD NACIONAL DE COLOMBIA

Generación de diagramas de clase y casos de uso a partir de historias de usuario utilizando procesamiento de lenguaje natural

Miguel Ángel Tovar Onofre

Universidad Nacional de Colombia

Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial

Bogotá D.C., Colombia

2023

Generación de diagramas de clase y casos de uso a partir de historias de usuario utilizando procesamiento de lenguaje natural

Miguel Ángel Tovar Onofre

Trabajo de grado presentado como requisito parcial para optar al título de:
Magíster en Ingeniería de Sistemas y Computación

Director(a):

Ph.D. Jorge Eliecer Camargo

Línea de Investigación:

Ingeniería de software

Grupo de Investigación:

UnSecureLab

Universidad Nacional de Colombia

Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial

Bogotá D.C., Colombia

2023

A mis padres que me han apoyado y han creído en mi en cada proyecto. A Kikiz que con su paciencia, amor y ayuda me dio el empujón para continuar cada día. A cada persona que con un grano de arena aportó a que este trabajo sea una realidad.

Resumen

Generación de diagramas de clase y casos de uso a partir de historias de usuario utilizando procesamiento de lenguaje natural

El presente trabajo busca el desarrollo de un modelo computacional para la generación de diagramas UML a partir de historias de usuario en español, por medio de la aplicación de patrones gramaticales y procesamiento de lenguaje natural. Como conjunto de datos se tomaron diferentes conjuntos de historias de usuario traducidas al español y sus correspondientes diagramas generados manualmente. Los patrones aplicados fueron construidos con base en reglas establecidas para este proceso en idioma inglés, las cuales fueron adaptadas al idioma español y con base en los componentes extraídos, se construyen los diagramas. La evaluación del modelo computacional indica que es capaz de detectar los componentes como clases y actores, alcanzando un recall de hasta 0.8 en algunos casos. Sin embargo, presenta problemas de precisión al momento de extraer sus atributos, métodos o casos de uso, llegando a presentar valores inferiores a 0.1 en algunos componentes. Finalmente el modelo establece una base para guiar a los diseñadores y/o analistas en la implementación de proyectos de software.

Palabras clave: Reconocimiento de patrones, UML, Modelos computacionales, Procesamiento de lenguaje natural, Aprendizaje de maquina, Análisis de requerimientos, historias de usuario.

Abstract

Generating class diagrams and use cases from user stories using natural language processing

The present work seeks to develop a computational model for the generation of UML diagrams from Spanish user stories by means of the application of grammatical patterns and natural language processing. Different sets of user stories translated into Spanish and their corresponding manually generated diagrams were taken as a dataset. The applied patterns were constructed based on rules established for this process in English language, which were adapted to Spanish language and based on the extracted components, the diagrams were constructed. The evaluation of the computational model indicates that it is capable of detecting components such as classes and actors, reaching a recall of up to 0.8 in some cases. However, it presents precision problems when extracting attributes, methods or use cases, presenting values lower than 0.1 in some components. Finally, the model establishes a basis to guide designers and/or analysts in the implementation of software projects.

Keywords: Patterns recognition, UML, Computational models, Natural Language Processing, Machine Learning, Requirements analysis, User stories

Este Trabajo Final de maestría fue calificado en marzo de 2023 por el siguiente evaluador:

Felipe Restrepo Calle PhD.

Profesor Facultad de Ingeniería

Universidad Nacional de Colombia

Lista de Figuras

2-1. Modelo tradicional de proceso de ingeniería de requerimientos [1]	7
2-2. Ejemplo de notación UML para un diagrama de casos de uso [2]	10
2-3. Diagrama de clases [3]	11
2-4. Proceso de Entendimiento de Lenguaje Natural [4]	12
2-5. Flujo de proceso para extracción de diagramas de clase y casos de uso [5] . .	13
2-6. Flujo de proceso de extracción y generación de casos de uso [6]	15
2-7. Ejemplo de escenario de casos de uso generado por medio de reglas lingüísticas y PLN [7]	16
2-8. Flujo de proceso para la generación de diagramas de clase a partir de historias de usuario [8]	17
3-1. Flujo de proceso para la construcción de modelo computacional y evaluación de resultados [Autor]	22
3-2. Flujo de proceso para la extracción de entidades [Autor]	28
3-3. Flujo de proceso para la extracción de clases, atributos, métodos y relaciones [Autor]	32
3-4. Flujo de proceso para la extracción de casos de uso [Autor]	39
3-5. Flujo de proceso para la generación de diagramas de UML a partir de los componentes extraídos [Autor]	43
3-6. Ejemplo de las entidades más frecuentes obtenidas para el archivo <i>US25</i> [Autor] 44	
3-7. Ejemplo de estructuración de clases para su análisis con PlantUML [Autor] .	47
3-8. Ejemplo de diagrama de clases creado por medio de PlantUML [Autor] . . .	48

3-9. Ejemplo de estructuración de casos de uso para su análisis con PlantUML [Autor]	48
3-10. Ejemplo de diagrama de casos de uso creado por medio de PlantUML [Autor]	48
4-1. Flujo de comparación de resultados obtenidos en extracción de entidades [Autor]	53
4-2. Gráficas de precisión, <i>Recall</i> , <i>Accuracy</i> y F1 correspondientes a la cantidad de entidades tomadas del resultado del sistema	54
4-3. Flujo de comparación de resultados obtenidos en extracción de clases [Autor]	56
4-4. Gráficas de precisión, <i>Recall</i> , <i>Accuracy</i> y F1 correspondientes a la evaluación #1 de las clases [Autor]	58
4-5. Gráficas de precisión, <i>Recall</i> , <i>Accuracy</i> y F1 correspondientes a la evaluación #2 de las clases [Autor].	59
4-6. Gráficas de precisión, <i>Recall</i> , <i>Accuracy</i> y F1 correspondientes a la evaluación #3 de las clases [Autor].	60
4-7. Flujo de comparación de resultados obtenidos en extracción de atributos de la clase [Autor]	63
4-8. Gráficas de precisión, <i>Recall</i> , <i>Accuracy</i> y F1 correspondientes a la evaluación #1 de los atributos [Autor].	64
4-9. Gráficas de precisión, <i>Recall</i> , <i>Accuracy</i> y F1 correspondientes a la evaluación #2 de los atributos [Autor].	65
4-10. Flujo de comparación de resultados obtenidos en extracción de métodos de la clase [Autor]	67
4-11. Gráficas de precisión, <i>Recall</i> , <i>Accuracy</i> y F1 correspondientes a la evaluación #1 de los métodos [Autor].	68
4-12. Gráficas de precisión, <i>Recall</i> , <i>Accuracy</i> y F1 correspondientes a la evaluación #2 de los métodos [Autor].	70
4-13. Flujo de comparación de resultados obtenidos en extracción de relaciones entre las clases [Autor]	71
4-14. Gráficas de precisión, <i>Recall</i> , <i>Accuracy</i> y F1 correspondientes a la evaluación # 1 de las relaciones entre clases [Autor].	73

4-15.	Gráficas de precisión, <i>Recall</i> , <i>Accuracy</i> y F1 correspondientes a la evaluación # 2 de las relaciones entre clases [Autor].	74
4-16.	Flujo de comparación de resultados obtenidos en extracción de actores [Autor]	76
4-17.	Gráficas de precisión, <i>Recall</i> , <i>Accuracy</i> y F1 correspondientes a los actores obtenidos por el sistema [Autor].	77
4-18.	Flujo de comparación de resultados obtenidos en extracción de casos de uso de los actores [Autor]	79
4-19.	Gráficas de precisión, <i>Recall</i> , <i>Accuracy</i> y F1 correspondientes a los casos de uso obtenidos por el sistema [Autor].	80
5-1.	Diagrama esquemático de software construido para la generación de diagramas UML a partir de historias de usuario [Autor].	83

Lista de Tablas

2-1. Tabla comparativa de sistemas previos	18
3-1. Tabla de archivos obtenidos en el conjunto de datos	24

Contenido

Resumen	vii
Lista de figuras	ix
Lista de tablas	xii
1. Introducción	2
1.1. Identificación del problema	2
1.2. Pregunta de investigación	4
1.3. Objetivo general	4
1.4. Objetivos específicos	4
1.5. Contribución	5
1.5.1. Repositorio de conjuntos de historias de usuario en español	5
1.5.2. Modelo computacional para la generación de diagramas UML a partir de historias de usuario	6
1.5.3. Artículo de extracción de clases a partir de historias de usuario	6
2. Estado del arte	7
3. Modelo computacional para la generación de diagramas de clase y casos de uso	21
3.1. Obtención de conjunto de datos y diagramas UML generados manualmente	21
3.1.1. Obtención de historias de usuario	21
3.1.2. Construcción de diagramas de clase y casos de uso manualmente	26
3.2. Extracción de entidades	27
3.2.1. Procedimiento de aplicación de patrones	30

3.3.	Extracción de clases, atributos, métodos y relaciones	31
3.3.1.	Procedimiento de aplicación de patrones	35
3.4.	Extracción de actores y casos de uso	39
3.4.1.	Procedimiento de aplicación de patrones	41
3.5.	Generación de diagramas de clase y casos de uso	42
3.5.1.	Entidades	43
3.5.2.	Clases, atributos, métodos y relaciones	44
3.5.3.	Actores y casos de uso	45
4.	Evaluación de resultados	49
4.1.	Extracción de entidades	52
4.2.	Extracción de clases, atributos, métodos y relaciones	56
4.2.1.	Clases	56
4.2.2.	Atributos	62
4.2.3.	Métodos	66
4.2.4.	Relaciones	71
4.3.	Extracción de actores y casos de uso	76
4.3.1.	Actores	76
4.3.2.	Casos de uso	78
5.	Prototipo de software	82
5.1.	PyUNML	83
5.2.	Extractor de entidades	84
5.3.	Extractor de clases	85
5.4.	Extractor de actores y casos de uso	85
5.5.	Generador de diagramas UML	86
6.	Conclusiones y trabajo futuro	87
6.1.	Conclusiones	87
6.2.	Trabajo futuro	90

A. Anexo: Prototipo

92

Bibliografía

93

1. Introducción

En este documento se presenta el proceso de construcción de un sistema computacional con la capacidad de extraer la información necesaria para la generación de diagramas UML a partir de un conjunto de historias de usuario en idioma español. Específicamente, se adaptará al español un conjunto de reglas o patrones gramaticales establecidos en idioma inglés para la extracción de componentes de diagramas de clase y casos de uso. Los patrones adaptados serán aplicados sobre diferentes conjuntos de historias de usuario en español, a los cuales previamente se les han generado sus respectivos diagramas de clase y casos de uso manualmente. Los componentes extraídos por medio del sistema computacional para un conjunto de historias de usuario son comparados con los diagramas recolectados manualmente para dicho conjunto y se determina la precisión del sistema frente a los diagramas recolectados. A continuación, se describen los principales pasos realizados durante el proceso de construcción del sistema, las reglas gramaticales establecidas, los resultados obtenidos y la discusión de los mismos.

1.1. Identificación del problema

En la actualidad, el análisis de requerimientos es una de las tareas más importantes en el desarrollo de software, al punto que podría decirse que es uno de los pilares que sostienen los proyectos en esta área, pues en esta tarea se modelan los diagramas UML que representan la estructura del sistema y se definen los requerimientos del sistema para poder cumplir con los deseos del usuario y tener un correcto funcionamiento. En este ámbito, el desarrollo de software ágil ha representado una mejora en el proceso de análisis al hacer uso de historias de usuario para expresar los requerimientos del usuario, debido a su representación en lenguaje

natural y su corto formato (*Como [Actor], deseo/necesito [Acción], para [Beneficio]*), son fácilmente comprensibles para las partes interesadas [9].

Sin embargo, el análisis de requerimientos es una de las tareas más complejas, dado que los desarrolladores deben realizar un análisis detallado de los requerimientos textuales suministrados por el usuario, lo que en gran parte de los casos -si no es en todos- ocupa gran cantidad de tiempo al ser un proceso manual [10] o está sujeto a errores en la interpretación de los analistas. Esto a causa de múltiples interpretaciones que podrían presentarse por un mismo término dentro de las historias de usuario, dando paso a ambigüedades al interpretar los requerimientos. Así mismo, los requerimientos podrían presentar inconsistencias en su redacción o ausencia de información primordial [11] para su interpretación y análisis, conllevando a que los resultados de este proceso o los diagramas UML generados a partir del mismo no contengan la información esencial para brindar una estructura sólida al sistema a construir. Aunque el uso de historias de usuario mitiga las inconsistencias y la ausencia de información en los requerimientos, requieren de comunicación continua con el usuario y por consiguiente, de mayor cantidad de tiempo destinado al análisis y comprensión de las mismas. En busca de automatizar este proceso y mitigar las situaciones mencionadas, durante la última década se han implementado distintos sistemas de generación automática de diagramas UML a partir de requerimientos expresados en lenguaje natural, pero estos son enfocados al idioma inglés, en algunos casos con determinadas especificaciones en el formato o palabras usadas [11] y no se cuenta con un enfoque en el idioma español.

Como consecuencia, el análisis de requerimientos en idioma español aun se ve forzado a ser realizado de manera manual, manteniéndose expuesto a las situaciones presentadas y con ello, a errores en el modelado de los diagramas UML por incongruencias o falta de información para su correcta construcción. Todo esto puede verse reflejado en las etapas posteriores de desarrollo de software [12] y afectar de manera negativa el desarrollo del proyecto, aumentando los tiempos estipulados y los costos establecidos para su ejecución a causa de la corrección de errores.

1.2. Pregunta de investigación

A raíz de estas problemáticas, se busca la implementación de un sistema de generación automática de diagramas UML a partir de requerimientos de software expresados en idioma español con el cual se pueda obtener la estructura base para la construcción del sistema objetivo. Debido a la mejora al proceso de análisis de requerimientos por medio de las metodologías ágiles, el sistema a implementar se enfocará en el análisis de requerimientos expresados por medio de historias de usuario. Así mismo, estará orientado a la generación de diagramas de clase y diagramas de casos de uso dada su utilidad al momento de construir el sistema objetivo y mapear las diferentes funcionalidades con las cuales contará.

Por consiguiente, en este trabajo se desarrollará la siguiente pregunta: ¿Cuál es el proceso a seguir para apoyar la generación automática de diagramas de clases y casos de uso a partir de las historias de usuario del software en idioma español?

Investigaciones e implementaciones realizadas durante los últimos años para la generación de diagramas UML partiendo de historias de usuario sugieren el uso de procesamiento de lenguaje natural y búsqueda de patrones para la extracción de los componentes y la información requerida para la construcción de los diagramas objetivo.

1.3. Objetivo general

Por esta razón, el objetivo principal de este trabajo fue:

- Desarrollar un modelo computacional para la generación automática de diagramas de clases y casos de uso a partir de las historias de usuario del software objetivo, utilizando técnicas de procesamiento de lenguaje natural y reconocimiento de patrones.

1.4. Objetivos específicos

El cual fue dividido en cuatro objetivos específicos para su ejecución:

- Identificar las entidades presentes dentro del sistema por medio de procesamiento de lenguaje natural partiendo de las historias de usuario expresadas en idioma español, con el objetivo de realizar un reconocimiento preliminar de los posibles componentes del sistema.
- Diseñar modelo computacional de clasificación en clases, atributos, relaciones, actores y casos de uso, usando modelos de clasificación basados en reconocimiento de patrones en la estructura del texto, con el fin de extraer los principales componentes de los diagramas objetivo.
- Generar los diagramas de clase y de casos de uso a partir de los componentes obtenidos por medio de la clasificación realizada, usando librerías de transformación de archivos de texto a diagramas UML, con el propósito de visualizar los resultados obtenidos por medio del modelo computacional.
- Evaluar los resultados del modelo computacional frente a la generación de los diagramas manualmente, usando métricas de *Machine Learning* tales como la precisión y *Recall*, con la finalidad de determinar la exactitud de los diagramas obtenidos por el modelo computacional frente a los diagramas recolectados.

1.5. Contribución

1.5.1. Repositorio de conjuntos de historias de usuario en español

Se brinda un repositorio de archivos de texto, que cuentan con distintos conjuntos de historias de usuario traducidas al español, las cuales describen distintos sistemas de software y puede ser accedido desde el repositorio en línea indicado en el anexo A y cada conjunto de historias de usuario cuenta con más de 50 historias. El objetivo de este repositorio es que pueda ser utilizado en futuros trabajos para la implementación de modelos computacionales más precisos o en la solución de problemáticas similares, debido a la ausencia de conjuntos de datos similares enfocados al idioma español.

1.5.2. Modelo computacional para la generación de diagramas UML a partir de historias de usuario

El código fuente del modelo computacional y su instalación se encuentran indicados en el anexo A para aquellos interesados en experimentación y/o conocer el proceso realizado para ser replicado en futuros trabajos similares.

1.5.3. Artículo de extracción de clases a partir de historias de usuario

Con base en resultados preliminares obtenidos por modelo (Evaluación #1 de la extracción de clases), se redactó un artículo científico el cual fue aceptado en la conferencia *SmartTech-IC 2022*:

Tovar, M & Camargo, J. (2022). Automatic class extraction from Spanish text of user stories using natural language processing. In: Narváez, F. (eds) Smart Technologies, Systems and Applications. SmartTech-IC 2022. Communications in Computer and Information Science. Springer, Cham. (En proceso de publicación)

2. Estado del arte

La ingeniería de requerimientos es un área de la ingeniería de software en la cual se aplican diferentes métodos y técnicas para analizar los requerimientos que debe cumplir el software objetivo y poder determinar su viabilidad, necesidad, consistencia y alcance [13]. En esta se involucran distintos procesos que van desde la obtención de los requisitos, hasta la validación y gestión de los mismos (Figura 2-1.), siendo una de las etapas más importantes para el desarrollo de software, pues con base en los requerimientos se determinan las principales funcionalidades que tendrá el software, la estructura y capacidades técnicas del mismo [1].

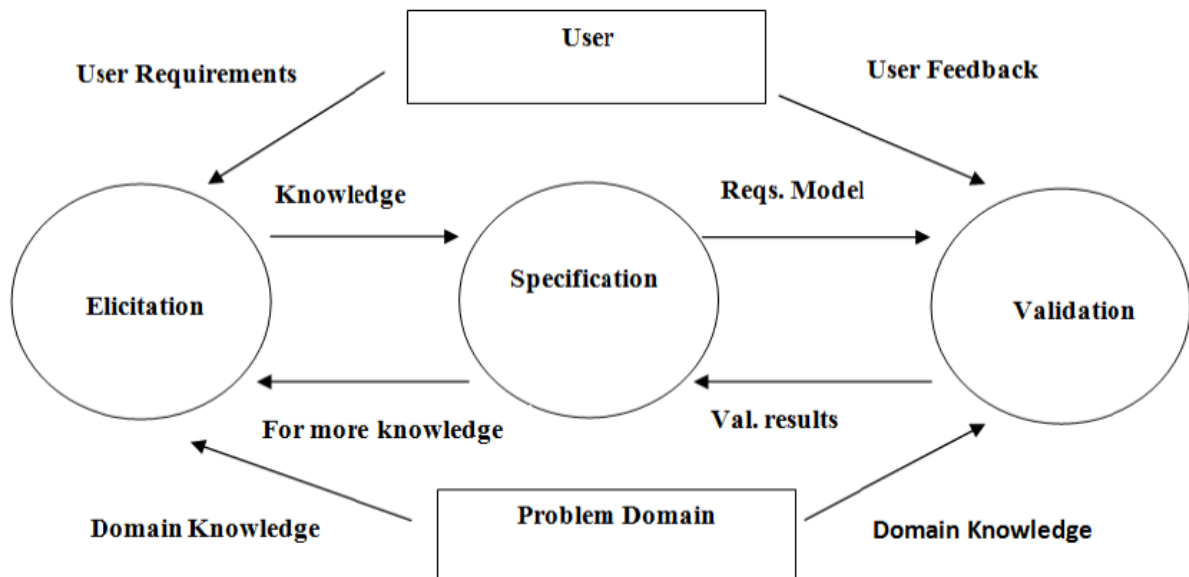


Figura 2-1.: Modelo tradicional de proceso de ingeniería de requerimientos [1]

No obstante, tiende a ser un proceso engorroso y puede estar propenso a errores, pues los ingenieros de requerimientos realizan lluvias de ideas y entrevistas con los clientes para

determinar los requerimientos del software a construir [1], lo cual puede conllevar demasiado tiempo y en algunos casos, los requerimientos extraídos no cuentan con la solidez necesaria para realizar el análisis adecuadamente, pues pueden ausentarse términos esenciales para su interpretación o algunos términos podrían tener distintas interpretaciones, por lo cual se podría presentar una ambigüedad y con esto, una interpretación errónea del requerimiento y las necesidades que cubre.

A raíz de esto, en los últimos años ha aumentado el uso de metodologías ágiles con el objetivo de reducir el tiempo empleado en el análisis de requerimientos y optimizar los procesos de desarrollo de software. En este enfoque, se da mayor importancia al correcto funcionamiento del software y satisfacción de las necesidades del usuario que al progreso en el desarrollo del mismo [1]. Para cumplir con dicho objetivo, desde el análisis de requerimientos se han estado usando nuevos mecanismos y herramientas para garantizar la calidad de los requerimientos y con esto, la estructura y funcionamiento del software a construir. Entre los cambios realizados respecto a las metodologías tradicionales donde se realizaban reuniones con el cliente para extraer los requerimientos, en estas nuevas metodologías son proporcionados por el mismo cliente o por el *Product Owner* por medio de historias de usuario.

Las historias de usuario son especificaciones semi-estructuradas cortas escritas en lenguaje natural que expresan los requerimientos desde la perspectiva del usuario, y consisten de tres aspectos: *Quien* correspondiente al actor, *Que* correspondiente a la acción deseada o necesaria para el actor y *Por que* correspondiente a la razón de dicho deseo o necesidad (Esta parte es opcional) [14]. Estas partes se involucran en la historia de usuario de una de las siguientes maneras:

- Como [Actor], quiero/necesito/puedo [Acción], para [Razón].
- Con el fin de [Razón], como [Actor], deseo [Acción]

La estructura concreta de las historias de usuario permite a los analistas comprender con mayor facilidad los deseos y necesidades del usuario, mitigando los posibles errores por mal interpretación que se podrían presentar durante el proceso de análisis. Sin embargo, dada su estructura concreta, en ocasiones los analistas requieren una perspectiva más detallada,

por lo que se deben realizar reuniones constantes con el cliente para poder obtener la información necesaria [15], conllevando a tomarse demasiado tiempo en el entendimiento de las necesidades del usuario, que podría ser aprovechado en etapas posteriores del desarrollo.

Ahora bien, los analistas y desarrolladores de software deben buscar estrategias o herramientas que les permitan tener una perspectiva global del software con base en los requerimientos obtenidos y como se relacionan estos para comprender el comportamiento que tendrá el sistema en diversas situaciones o casos. Para esta tarea, los diagramas UML suelen ser una herramienta bastante importante y útil, siendo utilizada y conocida actualmente por la gran mayoría -si no son todos- los involucrados en la ingeniería de software.

Unified Modeling Language (UML) es un lenguaje gráfico de modelado, estandarizado en 1997 por el *Object Management Group* (OMG), usado mayormente para la visualización, documentación y análisis de componentes del software y como se relacionan para el adecuado comportamiento del sistema [3].

Entre los diagramas construidos con más frecuencia por medio de este lenguaje se encuentran:

- **Casos de uso:** Define las acciones que puede realizar un actor o usuario dentro del sistema, determinando su comportamiento por medio de funcionalidades. Un ejemplo de su notación se puede observar en la figura **2-2**, donde el actor está relacionado con los casos de uso y los casos pueden estar relacionados entre ellos con *extend* o *include*.
- **Diagramas de secuencia:** Presenta la interacción entre el actor y el sistema a través del tiempo, representado por medio de los mensajes enviados entre ambas partes.
- **Diagramas de clase:** Determina las clases y las relaciones existentes entre ellas para comprender como interactúan entre si cada uno de los componentes del sistema y su funcionamiento. Un ejemplo de las clases y los diferentes aspectos de sus relaciones se pueden encontrar en la figura **2-3**.
- **Diagramas de estados:** Representa los diferentes estados en los cuales puede estar una entidad del sistema y cuales son las condiciones que se deben cumplir para pasar de un estado a otro según el caso.

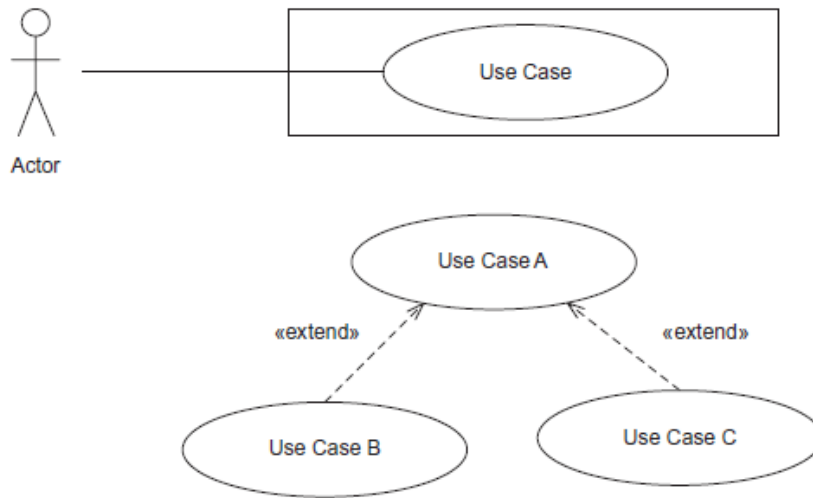


Figura 2-2.: Ejemplo de notación UML para un diagrama de casos de uso [2]

A raíz de los múltiples diagramas UML que pueden ser construidos a partir de los requerimientos de software, durante los últimos años se han venido realizando diferentes trabajos con el objetivo de automatizar este proceso haciendo uso de técnicas de Procesamiento de Lenguaje Natural.

Procesamiento de Lenguaje Natural (PLN) es un campo de la inteligencia artificial en el cual se ven involucradas distintas disciplinas tales como computación, lingüística, psicología u ontología, con el objetivo de comprender el lenguaje natural humano.

Entre sus principales aplicaciones se encuentran:

- Traducción automática.
- Análisis de sentimientos.
- Extracción de información.
- Diálogos automáticos.

Dentro de cada una de estas aplicaciones, se ejecutan distintos procesos de fragmentación y comprensión de cada una de las partes del texto a analizar, pues se realiza un análisis desde la parte semántica como sintáctica del mismo.

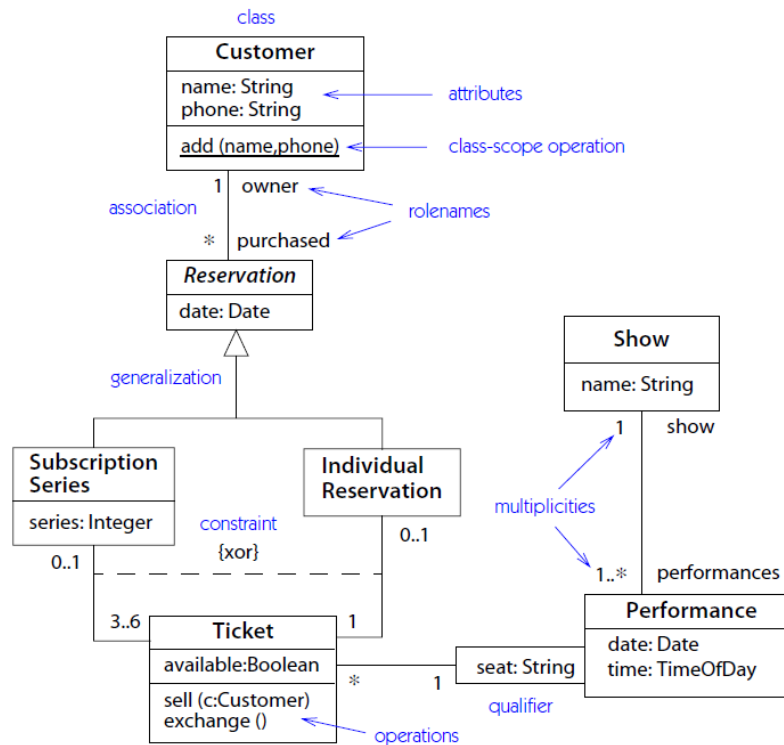


Figura 2-3.: Diagrama de clases [3]

Entre los principales componentes de PLN se encuentra el *Entendimiento del Lenguaje Natural*, el cual se enfoca en reconocer y comprender el lenguaje hablado (o escrito). Como se puede observar en la figura 2-4, este componente cuenta con cuatro procesos principales: Reconocimiento de voz, análisis sintáctico, análisis semántico y análisis pragmático [4].

Reconocimiento de voz

Su objetivo es dividir fragmentos del texto o lenguaje hablado en conjuntos de términos o tokens correspondientes a párrafos, oraciones y palabras.

Análisis sintáctico

En este proceso hay dos objetivos:

1. Determinar si la oración a analizar está bien formada.

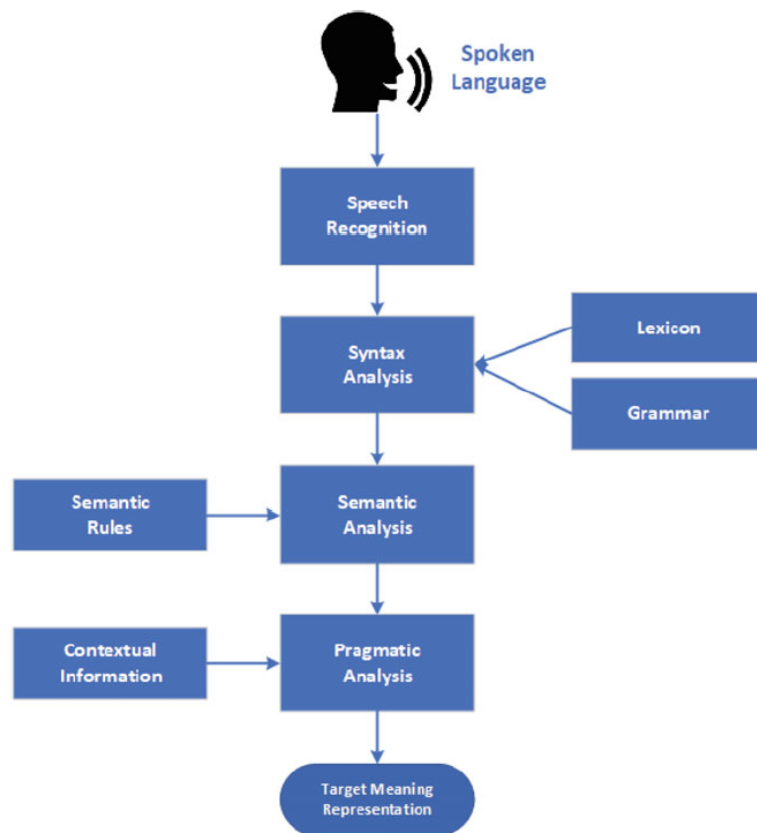


Figura 2-4.: Proceso de Entendimiento de Lenguaje Natural [4]

2. Dividir la oración en una estructura sintáctica que permita observar las relaciones existentes entre cada uno de sus términos.

Análisis semántico

En esta fase, se extrae el significado del texto con base en las relaciones encontradas en sus términos. En este proceso, se rechazan las oraciones o textos sin sentido y se comprende el significado exacto de las oraciones recibidas.

Análisis pragmático

Esta fase es la más compleja, pues su objetivo es reinterpretar el texto u oración recibida para comprender el significado que se pretendía dar a la oración basado en el conocimiento del mundo real y no en el significado literal del término. Estos son casos recurrentes en frases

donde se pretende dar un significado diferente o basado en el contexto al término a fin de expresar una determinada idea.

Durante los últimos años, se han desarrollado diferentes trabajos enfocados a la automatización del análisis de requerimientos a partir de la especificación textual del software o historias de usuario, utilizando diferentes metodologías.

En 2015, en [16] se presentó un sistema capaz de generar diagramas Entidad - Relación partiendo de la descripción de requerimientos. En este trabajo, se segmentó el texto para extraer cada una de sus oraciones por separado y analizarlas por medio de *Parsing Tree* haciendo uso de las etiquetas *Part of Speech* (PoS) de cada término, con el objetivo de extraer cada una de las entidades presentes en el texto y como se encuentran relacionadas. Sin embargo, el texto recibido con la descripción de requerimientos puede ser ambiguo y/o presentar múltiples árboles de análisis.

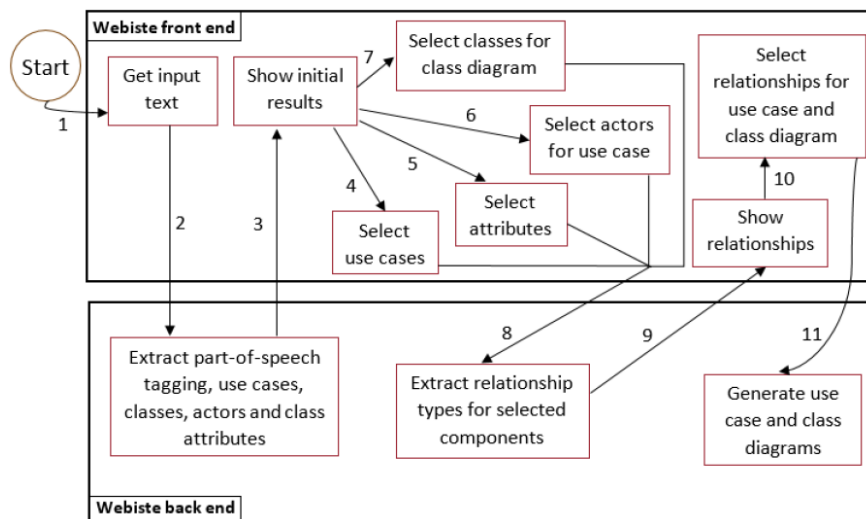


Figura 2-5.: Flujo de proceso para extracción de diagramas de clase y casos de uso [5]

Dos años más tarde, en 2017, se implementó una aplicación web en [5], que recibe los requisitos como descripción textual del software en lenguaje natural. Esta descripción es preprocesada pasando por la etapa de tokenización y PoS para determinar las etiquetas PoS que corresponden a cada token. Este trabajo tiene un enfoque basado en la aplicación de reglas XML, por lo cual, dichas reglas son aplicadas sobre el resultado del preprocesamiento para extraer las clases, actores y atributos del sistema. Los casos de uso son extraídos de

la conjunción entre verbos y sustantivos encontrados en las oraciones donde los actores son los sujetos de la misma. Una particularidad de este sistema es que al ser aplicación web, el usuario puede elegir cuales elementos de los encontrados (Clase, actores, atributos o casos de uso) conservar al generar los diagramas de clase y casos de uso. Una vez que el usuario haya elegido los elementos (o haya omitido la elección), el sistema procede a generar los diagramas de clase y casos de uso teniendo en cuenta las relaciones entre los diferentes elementos finales. El proceso completo puede ser observado en la figura **2-5**. El autor de este trabajo testeo el sistema con más de 20 escenarios diferentes, dando como resultado una exactitud que ronda el 70 %, mostrando que a pesar de tener un resultado cercano a lo esperado en el proceso de análisis de requerimientos habitual, aun hay irregularidades en la detección de algunos elementos.

Casi de manera paralela, ese mismo año en [6] se presentó una de las primeras metodologías basadas en probabilidad. Recibe como entrada un documento de requerimientos de usuario, el preprocesado por medio de Tokenización y PoS para identificar los sustantivos y verbos de cada oración. Los resultados de este preprocesamiento son enviados a dos clasificadores bayesianos: Un clasificador de actores y un clasificador de casos de uso. Ambos clasificadores fueron entrenados por medio de aprendizaje supervisado basado en las características de las palabras, las cuales son clasificadas con base a su probabilidad de pertenecer a diferentes clases. El clasificador de actores retorna el conjunto de actores detectados en el texto y el clasificador de casos de uso retorna el conjunto de casos de uso sintetizado y de la forma Verbo + Sustantivo. Posteriormente, el diagrama de casos de uso se construye con base en los resultados de los clasificadores. El proceso realizado por el sistema se puede observar en la figura **2-6**. Al evaluar los resultados, se evidencia que el sistema tiene un 70 % de precisión y un recall de 80 %, demostrando que los resultados son buenos y cercanos a los que se podrían obtener en el análisis de requerimientos habitual. Sin embargo, se siguen presentando componentes detectados erróneamente o que no fueron detectados.

Para 2018, en [17] se presentó un sistema enfocado al uso de reglas de transformación para la extracción de elementos. El sistema recibe un archivo de texto con el conjunto de historias de usuario a analizar. Se preprocesa el conjunto recibido, removiendo las palabras innecesarias

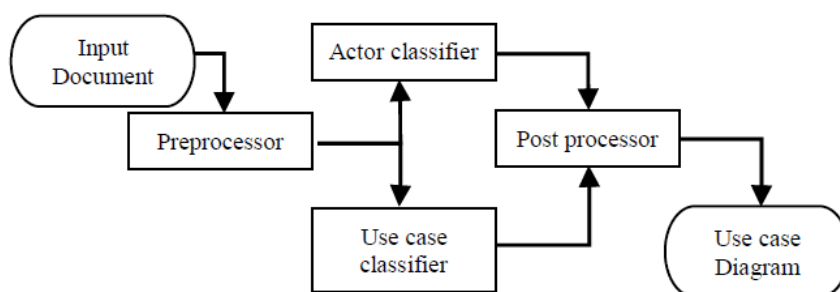


Figura 2-6.: Flujo de proceso de extracción y generación de casos de uso [6]

y generando el *Parse Tree* a cada historia de usuario, de modo que se pueda identificar la etiqueta PoS asignada a cada uno de los tokens de la oración. Una vez que se obtienen las etiquetas PoS, se aplican reglas de transformación basadas en dichas etiquetas, las cuales extraen los actores y casos de uso del sistema de acuerdo a la estructura identificada por los *Parse Tree*. Finalmente se generan los diagramas de casos de uso de acuerdo a los elementos extraídos por medio de las reglas de transformación. Este sistema tiene un 98 % de precisión para la detección de actores y un 87 % de precisión en la extracción de casos de uso, por lo cual es uno de los mejores y más precisos en los últimos años.

Ese mismo año, en [7] se presentó un enfoque generativo orientado al uso de PLN para la tokenización y construcción *Parse Tree* para cada una de las historias de usuario, sobre las cuales se aplican posteriormente reglas lingüísticas con el objetivo de extraer entidades nombradas. En la fase de pos-procesamiento se identifican elementos co-referenciados, que son convertidos en diagramas de objetos robustos. Estos diagramas en conjunto con las entidades nombradas son procesados para detectar las relaciones entre ellos y producir un diagrama robusto final con escenarios de casos de uso (Figura 2-7), generado por medio de PlantUML¹.

Posteriormente, en 2019 se presentó una metodología nueva por medio de [18], donde las historias de usuario son fragmentadas en: *Quien, Que, Porque, pre-condición, acción y pos-condición*. A cada una de estas partes le son identificadas sus respectivas etiquetas PoS, con las cuales se extraen los principales componentes como Agente, objeto, acción, estado

¹Ver: plantuml.com/es/

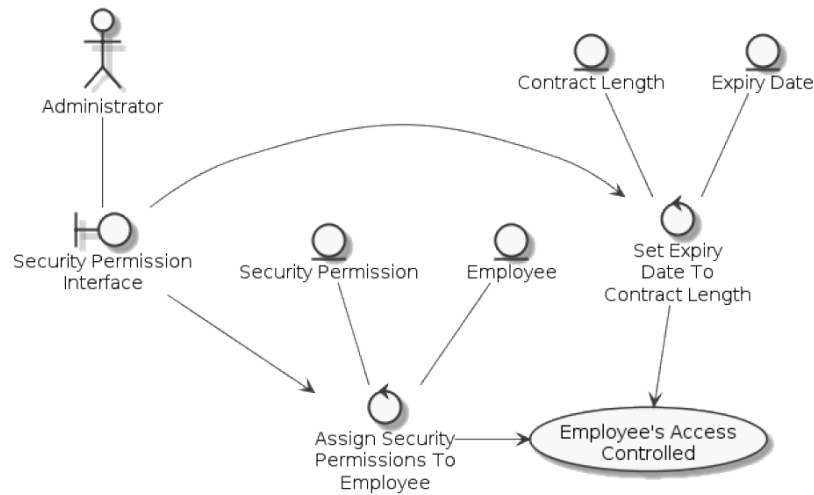


Figura 2-7.: Ejemplo de escenario de casos de uso generado por medio de reglas lingüísticas y PLN [7]

de pre-condición, etc. y una vez que se garantiza que los componentes están presentes en todas o la gran mayoría de historias de usuario, se procede a la construcción del diagrama Entidad-Relación o de casos de uso.

Un nuevo sistema fue presentado en 2020 en [8], enfocado a la generación de diagramas de clase a partir de historias de usuario. El sistema toma cada historia de usuario y extrae las etiquetas PoS correspondientes. Sin embargo, en este sistema también se identifica la palabra raíz de cada uno de los tokens con el objetivo de evitar tomar diferentes formas de la misma palabra como si fueran distintas palabras. Posteriormente se aplican reglas o patrones sobre la estructura de las historias de usuario para extraer los actores y clases, con las cuales se podrán identificar las relaciones de acuerdo la posición o estructura de las clases encontradas. Los atributos de estas pueden ser obtenidos por medio de las relaciones de composición entre las clases. Los elementos extraídos son incorporados dentro de un archivo XML, el cual es procesado para finalmente generar el diagrama de clases. El flujo de este proceso puede ser observado en la figura 2-8. Los diagramas de clase generados tienen un 98 % de exactitud respecto a los diagramas de clase generados manualmente, siendo bastante satisfactorios y eficientes para el proceso de análisis de requerimientos.

Ese mismo año, en [19] se toma la especificación textual del software y es procesada por

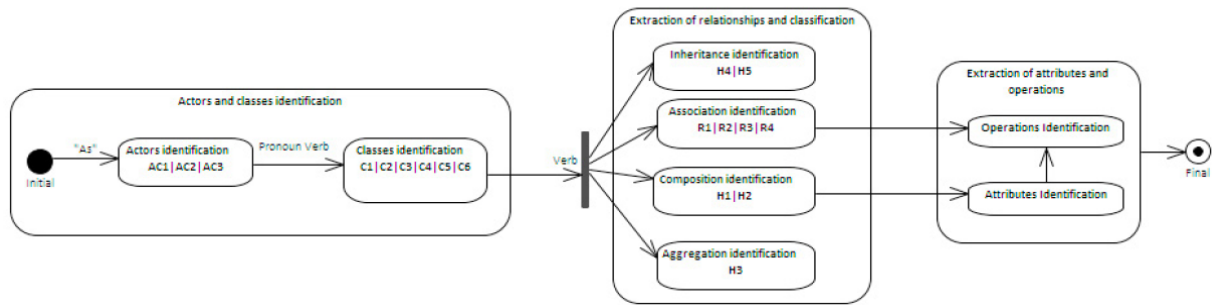


Figura 2-8.: Flujo de proceso para la generación de diagramas de clase a partir de historias de usuario [8]

medio de PLN en conjunto con NER (*Named Entity Recognition*) para extraer las entidades nombradas, nombres y pronombres presentes en el texto. Los resultados de esta extracción son convertidos a vectores por medio de *Word Embeddings* y pasados a través de una red neuronal convolucional, la cual en su última capa utiliza la función *softmax* para clasificar las entradas en la etiqueta más probable. Una vez clasificadas, se utiliza detección de co-referencias para encontrar las múltiples referencias que podrían haber para las mismas entidades y con esto, detectar las relaciones entre clases y Clase-Atributo. Finalmente, se procede a la construcción del diagrama de clases con los elementos obtenidos.

En 2021, se implementó un prototipo para la generación de diagramas de clase y casos de uso en [20], el cual consiste en preprocesar las historias de usuario por medio de PLN y convertir cada palabra en un vector, con el objetivo de medir la similaridad semántica entre las palabras de cada historia y la similaridad semántica entre las historias. Posteriormente, se agrupan las historias de usuario de acuerdo a su similaridad y se etiquetan de acuerdo a las palabras claves contenidas en cada agrupación. Finalmente se genera el diagrama de clases y de casos de uso con base a reglas heurísticas establecidas para la extracción de clases, actores y casos de uso. Sus resultados tienen un 100% de precisión en la extracción de actores y 98% para la extracción de casos de uso, lo cual evidencia resultados muy satisfactorios para la automatización y generación de los diagramas de clase y casos de uso.

Como se puede observar, se han realizado múltiples trabajos para automatizar el análisis de requerimientos ya sean especificaciones textuales o historias de usuario y llevarlas a diagramas

UML por medio de diferentes metodologías. Sin embargo, algo que tienen en común todos es que están orientados al idioma inglés y no hay un enfoque al idioma español, manteniendo la dificultad para los analistas en dicho idioma.

En la tabla **2-1** se encuentra un comparativo de los sistemas orientados a la automatización del análisis de requerimientos que fueron desarrollados en los últimos años.

Tabla 2-1.: Tabla comparativa de sistemas previos

Software	Entrada	Metodología	Salida
Generador de Diagramas Entidad-Relación [16]	Descripción textual del software	PLN y uso de Parsing Tree para visualizar la estructura de las oraciones y sus relaciones.	Diagrama Entidad-Relación
Generador de diagramas de casos de uso [6]	Documento de requerimientos de usuario	PLN y clasificadores bayesianos para la extracción de actores y casos de uso	Casos de uso
Generador de diagrama de clases y casos de uso [5]	Especificación textual del software en lenguaje natural	PLN y aplicación de reglas XML para la extracción de componentes con base a la estructura del texto. El usuario puede elegir los componentes que desea antes de proceder a generar los diagramas de clase y de casos de uso.	Diagrama de clase y diagrama de casos de uso
Generador de diagrama de casos de uso [17]	Historias de usuario	PLN, Parse Tree y reglas de transformación aplicadas sobre los token etiquetados	Diagrama de casos de uso
Continúa en la siguiente página			

Tabla 2-1 Continuación de la página anterior

Software	Entrada	Metodología	Salida
Generador de escenarios de casos de uso [7]	Historias de usuario	PLN y aplicación de reglas lingüísticas para la identificación de entidades nombradas y objetos	Escenarios de casos de uso
Generador de diagrama Entidad - Relación [18]	Historias de usuario	Descomposición de las historias de usuario y aplicación de PLN sobre cada una de las partes para la extracción de componentes basados en agentes y objetos.	Diagrama Entidad-Relación
Generador de diagramas de clase [8]	Historias de usuario	PLN y aplicación de reglas basadas en la estructura y relaciones entre los tokens para la extracción de clases y actores	Diagrama de clase
Generador de diagramas de clase [19]	Especificación textual del software	Extracción de entidades nombradas, sustantivos y pronombres por medio de PLN y NER. Entrenamiento de CNN para la clasificación de las palabras.	Diagramas de clases
Continua en la siguiente página			

Tabla 2-1 Continuación de la página anterior

Software	Entrada	Metodología	Salida
Generador de diagramas de casos de uso [20]	Historias de usuario	PLN y Word Embeddings para el análisis de similitud entre las palabras de cada historia. Agrupamiento de las palabras de acuerdo a sus características.	Diagramas de casos de uso

3. Modelo computacional para la generación de diagramas de clase y casos de uso

El presente capítulo describe el proceso realizado para obtener los conjuntos de historias de usuario y los diagramas de clases y casos de uso correspondientes a estos conjuntos. Posteriormente, se explican los pasos y herramientas usadas para la extracción de los componentes requeridos para la construcción automática de los diagramas UML. Finalmente se explica el procedimiento ejecutado para la transformación de los componentes detectados en diagramas UML que describan claramente los sistemas descritos por medio de las historias de usuario. En la figura 3-1 se puede observar el flujo de proceso realizado.

3.1. Obtención de conjunto de datos y diagramas UML generados manualmente

3.1.1. Obtención de historias de usuario

Para el desarrollo de este trabajo, se requiere un conjunto de múltiples historias de usuario agrupadas de acuerdo al sistema que describen, con el objetivo de poder identificar el sistema descrito por cada grupo de historias de usuario y de este modo, generar los diagramas UML correspondientes de manera automática por medio del modelo computacional y de

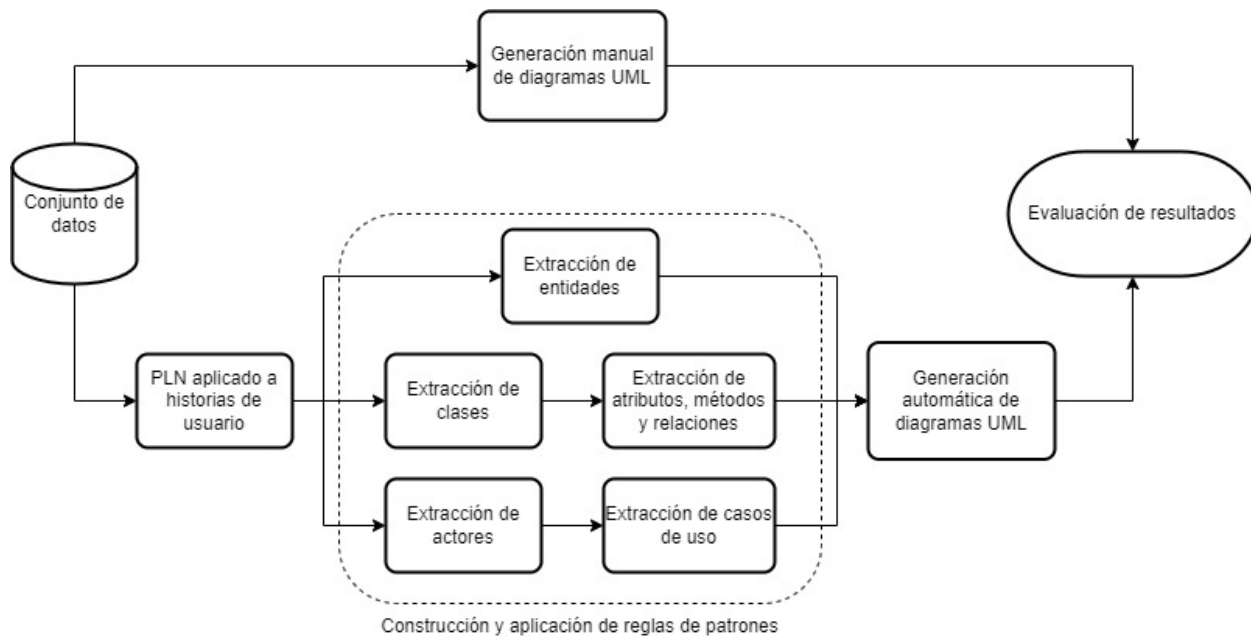


Figura 3-1.: Flujo de proceso para la construcción de modelo computacional y evaluación de resultados [Autor]

manera manual para realizar el análisis de resultados. Dado que el enfoque que se ha dado a la automatización del análisis de requerimientos se ha orientado al idioma inglés, no hay un conjunto de datos público en el cual se puedan encontrar historias de usuario en idioma español o aquellos que se encuentran no son claros respecto al sistema que describen sino que tienden a tener múltiples historias de usuario que pueden pertenecer a la descripción de múltiples sistemas.

A raíz de esto, se optó por el uso de dos conjuntos de datos compuestos por historias de usuario en idioma inglés, las cuales describen los principales requerimientos para sistemas computacionales con distintos objetivos.

El primer conjunto de datos [21] está compuesto por 22 archivos de texto con más de 50 historias de usuario cada uno. Cada archivo corresponde a la descripción de requisitos del usuario para un software diferente. El segundo conjunto de datos [22] está conformado por 15 grupos de archivos en los cuales se encuentran las historias de usuario o casos de uso

detallados para tres sistemas diferentes: Un simulador de tráfico urbano (SIM), un sistema de gestión hospitalaria (HOS) y el portal de la asociación internacional de fútbol (IFA).

Todas las historias de usuario en ambos conjuntos cumplen con el formato:

As [Actor], I want [Acción], so [Beneficio]

Preprocesamiento

El segundo conjunto de datos [22] contiene archivos PDF con historias de usuario en 11 de los 15 grupos de archivos que lo componen. Cada uno de estos archivos PDF fue transcrito en un archivo TXT en el cual se recopilieron las historias de usuario encontradas en dichos archivos de manera explícita. Al finalizar, se obtuvieron 11 archivos de texto con diferentes historias de usuario correspondientes a la descripción de uno de los tres sistemas: IFA, HOS o SIM. De acuerdo a lo anterior, se cuenta con un total de 33 archivos de texto con diferentes grupo de historias de usuario entre ambos conjuntos de datos.

Acto seguido, dado que ambos conjuntos se encuentran originalmente en idioma inglés, los archivos de texto son traducidos al idioma español haciendo uso del sistema de traducción automática *Google Translate* de manera similar a [23, 24], debido a su mínimo sesgo de traducción y a que sus resultados son muy cercanos a los que se pueden obtener por medio de traducción automática usando *Transformers* [25]. La traducción de cada uno de los archivos es almacenada en un archivo de texto aparte, que fue nombrado por medio de un alias (Tabla **3-1**) con el objetivo de reconocerlo con mayor facilidad durante el análisis de resultados. Para el presente trabajo, se optó por continuar con 30 de los 33 archivos traducidos para reducir el posible ruido generado por la traducción y tener un conjunto más concreto de datos para analizar los resultados del modelo computacional.

Tabla 3-1.: Tabla de archivos obtenidos en el conjunto de datos

Conjunto de datos #	Nombre original del archivo	Alias	¿Usado en procesamiento?
1	FederalSpending	US1	SI
1	Loudoun	US2	SI
1	Recycling	US3	SI
1	Openspending	US4	SI
1	Frictionless	US5	SI
1	Scrumalliance	US6	SI
1	NSF	US7	SI
1	CamperPlus	US8	NO
1	PlanningPoker	US9	SI
1	Datahub	US10	SI
1	MIS	US11	SI
1	CASK	US12	SI
1	Neurohub	US13	SI
1	Alfred	US14	SI
1	Badcamp	US15	SI
1	Rdadmp	US16	NO
1	ArchivesSpace	US17	SI
1	Unibath	US18	SI
1	Duraspace	US19	NO
1	Racdam	US20	SI
1	Culrepo	US21	SI
1	Zooniverse	US22	SI
2	g1	US23	SI
2	g2	US24	SI
Continua en la siguiente página			

Tabla 3-1 Continuación de la página anterior

Conjunto de datos #	Nombre original del archivo	Alias	¿Usado en procesamiento?
2	g4	US25	SI
2	g5	US26	SI
2	g6	US27	SI
2	g8	US28	SI
2	g9	US29	SI
2	g10	US30	SI
2	g11	US31	SI
2	g12	US32	SI
2	g14	US33	SI

Procesamiento del lenguaje natural

Una vez que se tienen los archivos TXT con las historias de usuario en español, se les aplican técnicas de Procesamiento de lenguaje natural con el objetivo de identificar la estructura de las distintas historias de usuario, las relaciones entre las partes de cada historia y sus principales características gramaticales. Este procedimiento fue realizado por medio del lenguaje de programación *Python*, más específicamente por medio de la librería *Stanza*. Esta fue desarrollada por *Stanford NLP Group*, enfocándose en el análisis de textos en lenguaje natural humano, que transforma en listas de oraciones y extrae sus características morfológicas y etiquetas PoS [26]. Principalmente, el *parser* de la librería recibe un texto suministrado por el usuario y este es analizado con base en el modelo preentrenado configurado por el mismo. Una vez que la librería a concluido el procesamiento del texto, retorna un conjunto bastante completo de características entre las que se encuentran los tokens (Tokenización), lemmas (Lematización), la etiqueta *Part-of-Speech* correspondiente a cada término o expresión contenida en el texto.

Dado que el enfoque del trabajo es en el idioma español, se utilizó el modelo preentrenado *Ancora*, proporcionado por *Stanza* y el cual está entrenado con un amplio conjunto de pa-

labras en este idioma. Se eligió esta herramienta para realizar el procesamiento de lenguaje natural debido a que brindaba una mayor precisión al momento de determinar los lemmas y etiquetas PoS con base en el contexto de cada palabra (Como la palabra *aplicación* la cual puede ser considerada un sustantivo o un verbo dependiendo del contexto), a diferencia de otras herramientas (La palabra *aplicación* es considerada verbo sin tener en cuenta el contexto o estructura de la oración que la contiene).

Las principales características extraídas por cada palabra usando *Part of Speech* son:

- **Token:** Es la palabra original en la historia de usuario.
- **upos:** Es la etiqueta PoS correspondiente a la palabra.
- **Características (Features):** Son las características de la palabra, entre las que se encuentran su genero, si es posesivo o su número gramatical.
- **lemma:** Es la forma más básica de la palabra en cuestión.

Con la intención de optimizar el análisis de resultados, la estructura de cada historia de usuario, sus etiquetas PoS y sus correspondientes características son almacenadas en una base de datos en el servicio en la nube *FireBase*¹ agrupadas de acuerdo al conjunto de datos y archivo al que pertenece cada historia de usuario.

3.1.2. Construcción de diagramas de clase y casos de uso manualmente

Dado que los conjuntos de historias de usuario no cuentan con sus correspondientes diagramas UML, estos deben ser construidos con el fin de tener una base que nos permita determinar si el modelo computacional genero los diagramas de manera adecuada.

Para este proceso, se solicitó la construcción de los diagramas UML correspondientes a las historias de usuario a estudiantes de carreras a fines a la ingeniería de software. A cada

¹firebase.google.com

estudiante se le asignaron 3 archivos diferentes los cuales fueron enviados vía mail y de acuerdo al análisis realizado por ellos a los requerimientos descritos en las historias de usuario de cada archivo, construyeron los diagramas de clase y casos de uso correspondientes. Para la construcción de los diagramas UML, se tenían los siguientes lineamientos:

- Las clases, entidades y actores extraídos deben mantenerse en singular.
- Las clases, atributos, métodos o actores que se compongan por varias palabras, deben separarlas por guión bajo. Ej: color_cabello, director_hospital, enviar_archivo().
- Los diagramas deben ser consistentes y tan completos cómo sea posible.

Al final de este proceso, cada estudiante entrega un conjunto de archivos agrupados por los archivos analizados. Cada grupo de archivos dentro del conjunto corresponde al análisis realizado a uno de los tres archivos asignados y está compuesto por:

- Un archivo TXT con la lista de entidades presentes.
- Un archivo PDF con el diagrama de clases
- Uno o varios archivos PDF correspondientes a los casos de uso construidos.

Como resultado de este proceso, se obtienen los diagramas UML y las listas de entidades extraídas manualmente para cada uno de los archivos del conjunto de datos.

Acto seguido, se procede a la construcción de las reglas de patrones, que serán aplicadas a las historias de usuario para la extracción de entidades, clases, atributos, métodos, actores y casos de uso para la construcción de los correspondientes diagramas.

3.2. Extracción de entidades

Esta primera etapa consiste de la construcción de patrones que son usados para extraer las entidades presentes en las historias de usuario y el procedimiento de aplicación de los mismos sobre el conjunto de historias. En la figura **3-2** describe el flujo del procedimiento.

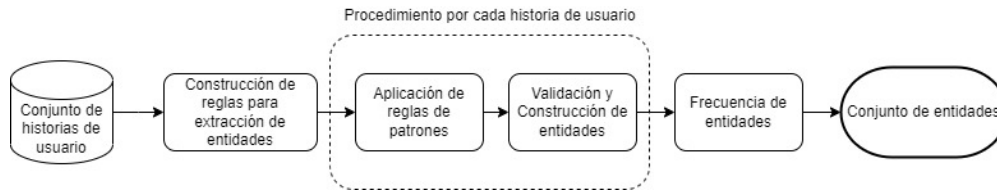


Figura 3-2.: Flujo de proceso para la extracción de entidades [Autor]

Para la construcción de las reglas de patrones enfocadas a la detección de entidades, se tomaron como base las reglas establecidas en [16] para textos escritos en inglés y fueron adaptadas al idioma español teniendo en cuenta los cambios gramaticales entre ambos idiomas y el cambio de formato a historias de usuario.

Como resultado de dicha transformación, se establecieron las siguientes reglas:

1. Un sustantivo puede ser una entidad.
2. Un nombre propio puede representar una entidad.
3. Si hay dos sustantivos separados por adjetivos o determinantes, se debe revisar el primer sustantivo. Si este no se encuentra entre alguna de las siguientes palabras:

*Atributos = ['número', 'no', 'código', 'fecha', 'tipo', 'volumen', 'nacimiento', 'id',
'dirección', 'nombre']*

Puede representar una entidad, de otro modo, podría representar un atributo de la entidad.

4. Si el sustantivo se encuentra entre el siguiente conjunto de palabras:

*Sistema = [base_de_datos', 'base_de_dato', 'base_dato', 'registro', 'sistema',
'información', 'organización', 'detalle', 'cosa']*

Dado que representan entidades para el funcionamiento propio del sistema, mas no entidades que se encuentren del objetivo principal del mismo.

Para extraer aquellos sustantivos que cumplen con estos criterios, se formaron los siguientes patrones:

$$E1: \{<NOUN|PROPN><ADJ><DET>?<NOUN|PROPN>*\}$$
$$E2: \{<NOUN|PROPN>*\}$$

En estos patrones, *NOUN* representa los sustantivos comunes, *PROPN* representa los nombres propios, *ADJ* representa los adjetivos y *DET* representa los determinantes. Estos patrones fueron refinados por medio de un proceso de observación de los resultados preliminares, frente a las entidades obtenidas manualmente en los conjuntos de datos. Durante este proceso, se observaron cuales entidades podían omitirse, aquellas palabras con determinadas etiquetas PoS que impedían detectar algunos sustantivos (Como adjetivos y determinantes) o las estructuras de las historias que no eran cubiertas por estas reglas y de este modo, poder realizar las modificaciones necesarias para tener en cuenta dichos casos y obtener mejores resultados.

Con el patrón *E1*, se extraen aquellos sustantivos compuestos, los cuales constan de dos (o más) sustantivos relacionados por medio de al menos un adjetivo o determinante y cuyo significado en conjunto brinda un contexto diferente al que podría dar cada sustantivo por separado.

Con el patrón *E2*, se extraen aquellos sustantivos simples que se encuentran en las historias de usuario y no están relacionados por medio de adjetivos o determinantes a otros sustantivos.

Una vez que se cuenta con las reglas de patrones para la extracción de las entidades, estas son aplicadas sobre cada una de las historias de usuario para determinar aquellos sustantivos compuestos o simples que cumplen con las mismas.

Para ello, se hace uso de la librería *NLTK* del lenguaje *Python*. Esta librería cuenta con la función *RegexParser*, la cual genera un parser o analizador sintáctico a partir de los patrones *E1* y *E2*. Cabe mencionar que el orden de los patrones tiene gran importancia al

momento de enviarlos a la función, pues serán validados en el orden que sean suministrados y en caso de que un fragmento de texto cumpla con dos o más patrones, será tenido en cuenta solo en el primer patrón que lo haya detectado.

3.2.1. Procedimiento de aplicación de patrones

Cada una de las historias de usuario es analizada por medio de este parser, el cual extrae árboles sintácticos conformados por aquellos fragmentos de las historias que cumplen con al menos uno de los patrones determinados. En este caso, dado el objetivo de los patrones, los árboles sintácticos extraídos están conformados por los sustantivos simples o compuestos encontrados y están etiquetados con el identificador dado a la respectiva regla por medio de la cual fueron obtenidos cada uno de los árboles sintácticos.

Cada árbol sintáctico encontrado es recorrido para observar cada uno de los términos que lo componen, sus principales características y unificarlos para formar una sola expresión que representa al sustantivo compuesto o al sustantivo simple extraído en el árbol. Dado que se pueden presentar casos en los cuales el mismo sustantivo puede encontrarse como plural o singular en diferentes partes del conjunto de historias de usuario, durante este recorrido se determina si el término se encuentra en plural o singular. Si el término está en plural, es convertido a su forma singular por medio del *lemma* obtenido al aplicar PLN. Si el término está en su forma singular, no es requerido realizar modificaciones sobre el mismo. De este modo, se garantiza que un mismo sustantivo no pueda encontrarse de diferentes maneras dentro de los resultados y así, evitar la duplicación de términos.

Acto seguido, todos los términos obtenidos son verificados de manera separada y unificada para asegurar que no se encuentran entre los conjuntos de palabras establecidos en las reglas 3 y 4. Al finalizar este proceso, si todos los términos aprobaron la verificación, estos se encuentran en su forma singular y son unificados por medio del carácter «_» en el caso de los términos que conforman un mismo árbol sintáctico.

Una vez que todas las historias de usuario han sido analizadas, se cuenta con un conjunto de todos los términos extraídos por medio de los patrones. Cada uno de estos corresponde a un sustantivo simple o un sustantivo compuesto y representa una entidad. Todas las entidades son concatenadas para conformar una cadena de texto con los resultados de todas las extracciones, la cual es procesada por medio de *CountVectorizer*.

CountVectorizer es una implementación de la librería *Sklearn* que permite determinar la frecuencia TF-IDF (*Term Frequency – Inverse Document Frequency*) de diferentes términos dentro de un mismo documento o cadena de texto. La concatenación de todas las entidades es procesada por medio de esta herramienta y acto seguido, se obtiene el listado de las entidades más importantes o nombradas dentro del conjunto de historias de usuario.

Como resultado de esta extracción, se obtiene el listado de entidades presentes en el conjunto de historias de usuario, ordenadas desde la más a la menos frecuente y con esto, se cumple el primer objetivo específico del trabajo, el cual consiste en la identificación de las entidades presentes en el sistema por medio de PLN. Cabe mencionar que durante este procedimiento se contempló el uso de NER (*Named Entity Recognition*) para la detección automática de entidades, sin embargo, no se logró establecer un resultado satisfactorio con ninguna herramienta, dado que no se logran detectar gran parte de las entidades que se obtuvieron de manera manual, sino solo aquellas que podrían ser muy reconocidas como nombres de organizaciones o lugares.

3.3. Extracción de clases, atributos, métodos y relaciones

En esta etapa, en esencia, se realiza un proceso similar. Sin embargo, se ejerce un mayor procesamiento de las historias de usuario y validaciones sobre los componentes obtenidas a partir de estas. El flujo del proceso se puede observar en la figura 3-3.

En [8] se establecen una serie de reglas para la extracción de clases y sus respectivos componentes en idioma inglés. Así mismo, se dan las pautas para la identificación de las relaciones entre estas. Entre las reglas establecidas se encuentran:

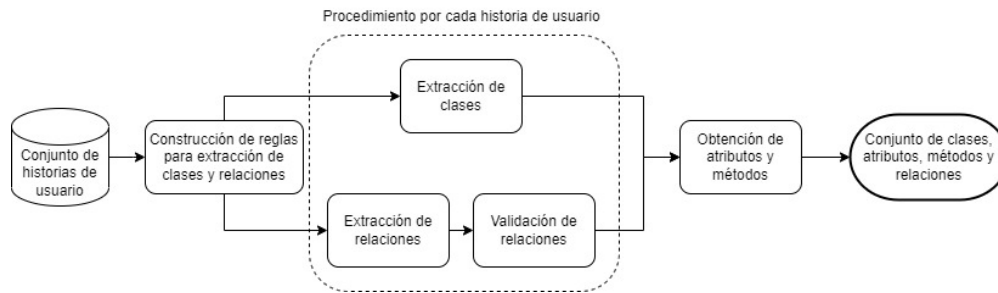


Figura 3-3.: Flujo de proceso para la extracción de clases, atributos, métodos y relaciones
[Autor]

Reglas para extracción de clases:

- El sujeto de una oración es una clase, este es un sustantivo y no un pronombre.
- El objeto directo de una oración es una clase.
- El sustantivo que precede un apóstrofe posesivo es una clase.
- El sustantivo después de las preposiciones *de*, *por* y *a* es una clase.
- La conjunción entre clases.

Reglas para extracción de relaciones:

- El verbo es una asociación entre el sujeto y el objeto directo.
- Las preposiciones representan una asociación.
- Los casos posesivos determinan una asociación.
- Los verbos *Contener*, *es parte de*, *Consistir*, *Incluir*, *Tener* indican una relación de composición.
- Si un sustantivo es compuesto y los sustantivos que lo componen son clases, indica una relación de composición.

- Si el sujeto y el objeto directo de una oración son clases y están relacionados por una conjugación del verbo *Ser*, indica una relación de herencia.
- Las frases «... es un tipo de ...» o «... es una clase de ...» indican una relación de herencia.

Estas reglas fueron adaptadas para ser aplicables al idioma español y reglas de patrones, las cuales fueron refinadas por medio de múltiples pruebas en las cuales se observaban las clases o los componentes obtenidos de las mismas y su similaridad frente a los diagramas de clase obtenidos manualmente en el conjunto de datos. Como resultado, se obtuvieron los siguientes patrones:

Para la extracción de posibles clases, se aplicó el siguiente patrón:

CLASS: (<NOUN>|<PROPN>)(<ADJ>|<DET>)*(<NOUN>|<PROPN>)*

De este modo, podemos obtener las posibles clases que este compuestas por un sustantivo simple o sean un sustantivo compuesto.

Para la extracción de posibles métodos, atributos y las relaciones entre clases, se construyeron los siguientes patrones:

Patrones:

R1: {<SUST>(<EXC_TERM.*>)*(<EXC_TERM3>|<EXC_TERM4>)*
 <VERB><CCONJ>*<VERB>*(<EXC_TERM1>|<EXC_TERM2>)*<SUST>}

R3: {(<SUST>)(<EXC_TERM2>)*<ADP>(<EXC_TERM2>)*(<SUST>)}

H4: {(<SUST>)(<EXC_TERM1>)*<AUX>(<EXC_TERM1>)*(<SUST>)}

EXC_TERM1: {(<ADP>|<DET>|<ADJ>)*}

EXC_TERM2: {(<ADJ>|<DET>)*}

EXC_TERM3: {<PUNCT><VERB>(<AUX>|<SCONJ>|<PRON>)*}

EXC_TERM4: {(<PROPN>|<PRON>|<ADV>)*}

SUST: {(<SUST_1>|<SUST_2>)+}

SUST_1: {(<NOUN>|<PROPN>)((<EXC_TERM1>)+(<NOUN>|<PROPN>)+)+}

SUST_2: {(<NOUN>|<PROPN>)+}

,
Reglas de exclusión : [
 'EXC_TERM1',
 'EXC_TERM2',
 'EXC_TERM3',
 'EXC_TERM4',
],

Reglas de inclusión:[
 'R1',
 'R3',
 'H4'
],

Verbos Ser :[
 'soy',
 'eres',
 'es',
 'somos',
 'son',
 'sean'
],

Sinónimos Tipo:[
 'tipo',
 'clase',
 'categoría',
 'especie',
 'genero',
 'variedad'
],

Verbos Inclusión :[

```
'comprenden',  
'comprende',  
'consiste',  
'consisten',  
'tiene',  
'tienen',  
'incluyen',  
'incluye',  
'abarcan',  
'abarca',  
'encierran',  
'encierra',  
'engloban',  
'engloba',  
'abrazan',  
'abraza',  
'contienen',  
'contiene'  
]
```

En este caso, los patrones para la extracción de relaciones requieren el reconocimiento de ciertas palabras particulares para determinar el cumplimiento de ciertas palabras (Verbos Ser, Verbos Inclusión y Sinónimos de Tipo) o conocer explícitamente cuales reglas debe excluir o incluir al momento de hacer el procesamiento de las historias de usuario.

3.3.1. Procedimiento de aplicación de patrones

Para aplicar los patrones construidos sobre las historias de usuario se realiza el siguiente procedimiento:

Primero, cada historia de usuario es analizada por un parser con el patrón de detección de clases. Cada árbol sintáctico obtenido por medio de este análisis es recorrido con el objeti-

vo de construir la clase detectada. Durante esta construcción se valida que ninguno de los sustantivos encontrados dentro del árbol pertenezcan a los conjuntos *Atributos* o *Sistema* y de ser así, el nombre de la clase es construido concatenando los sustantivos encontrados dentro del árbol sintáctico por medio de guión bajo «_», respetando el orden en el cual fueron encontrados.

Todas las posibles clases encontradas dentro de la historia de usuario son almacenadas dentro de un conjunto total de clases del conjunto de historias de usuario.

Posteriormente, los patrones de extracción de relaciones son aplicados sobre la historia de usuario por medio de un parser. Al igual que en el procedimiento anterior, cada uno de los arboles obtenidos por el parser es recorrido para poder analizar los tokens o términos que lo componen. En esta ocasión, se realizan múltiples validaciones sobre los árboles sintácticos encontrados para determinar las reglas que cumplen y las relaciones a construir.

La primera validación consta de determinar si en el árbol sintáctico se encontró algún sustantivo compuesto. De ser así, este es separado en cada uno de los sustantivos que lo componen y se crea una relación entre estos como posible relación entre dos clases. Así mismo, se valida si entre los posibles sustantivos encontrados en el árbol se presentan preposiciones que puedan representar una asociación entre posibles clases. En caso de encontrarse este caso, se crea y almacena la correspondiente relación en el conjunto de relaciones.

Acto seguido se valida si el árbol contiene alguno de los verbos contenidos en el conjunto *Verbos Inclusión* para construir la relación de composición o en los conjuntos *Verbos Ser* o *Sinónimos de tipo* para construir la relación de herencia. En caso de que el árbol sintáctico no corresponda a ninguno de los casos anteriores, se crea una sola relación con los sustantivos y verbos encontrados en la historia.

Dado que en una oración pueden presentarse múltiples verbos (Ej. «Quiero ver y coordinar mis envíos»), las relaciones obtenidas son recorridas para determinar cuales de ellas poseen

múltiples verbos y son separadas de modo que haya una relación por cada verbo encontrado en las mismas, con los mismos sustantivos. Las relaciones obtenidas al finalizar este proceso tienen la forma [Posible Clase 1, Verbo o unión, Posible clase 2] y son agrupadas de acuerdo al tipo de relación al cual pertenecen (Asociación, agregación, herencia, composición o dependencia) y el identificador del patrón con el que fue obtenida la relación.

Una vez que se han procesado todas las historias de usuario, las relaciones obtenidas se recorren con el objetivo de buscar posibles relaciones duplicadas y retirarlas, evitando su repetición en la construcción del diagrama objetivo.

Acto seguido, las clases obtenidas anteriormente y las posibles clases de las relaciones son comparadas y se retiran aquellas clases que no se encuentren tanto en las relaciones como en el conjunto de clases obtenido, reduciendo la cantidad de posibles clases detectadas.

Después de esto, las relaciones de composición son analizadas y de acuerdo al patrón con el cual hayan sido obtenidas, las clases relacionadas a la clase principal de cada relación son transformadas en atributos de dichas clases, es decir, en la relación [Clase 1, verbo, Clase 2], la *Clase 2* es transformada en atributo de la *Clase 1*. Esto debido a que en este tipo de relaciones se tienden a expresar los componentes o características de las diversas clases.

Para la obtención de los métodos, se recorren las relaciones asociativas y si su clase principal se encuentra dentro del arreglo de clases obtenido anteriormente, se concatenan su verbo y clase o atributo relacionado por medio de «_», dejando este tipo de relaciones con la forma [Clase 1, Verbo_Clase2], siendo la última posición de esta la que contiene el nombre del método detectado.

Por último, las clases que cuenten con atributos o métodos son organizadas en un conjunto con la forma:

```
{  
  Clases:  
    Clase 1:  
      Atributos:  
        Atributo 1,  
        Atributo 2,  
        ...  
      Métodos:  
        Método 1,  
        Método 2,  
        ...  
  
    Clase 2:  
      ...  
  
  Relaciones:  
    [Clase 1, Clase 2, ASOC],  
    ...  
}
```

Esto nos da como resultado un conjunto con todas las clases detectadas para la construcción del diagrama de clases, junto con sus atributos y métodos. Adicionalmente, se tiene el conjunto de relaciones entre dichas clases y el posible tipo de relación representado en las siguientes siglas:

- **ASOC**: Asociación

- **DEP**: Dependencia

- **HER**: Herencia

- **COMP**: Composición

- **AGR:** Agregación

3.4. Extracción de actores y casos de uso

En esta etapa, se construyen las reglas de patrones para la extracción de casos de uso y el procedimiento de aplicación de las mismas sobre las historias de usuario. El proceso puede ser visualizado en la figura 3-4.

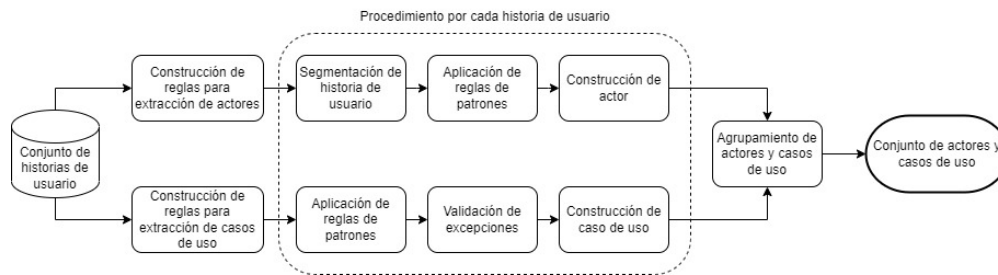


Figura 3-4.: Flujo de proceso para la extracción de casos de uso [Autor]

En [17] se establecen las siguientes reglas para la extracción de actores y casos de uso:

1. El primer sustantivo o sustantivo compuesto de la historia de usuario corresponde a un actor.
2. Los casos de uso son predicados de la forma Verbo + Sustantivo dentro de las historias de usuario, las cuales están asociadas a un actor.
3. Cada caso de uso extraído en una historia de usuario está asociado al actor de la misma.

Dado que estas reglas son aplicables a las historias de usuario en español, se construye un conjunto de patrones para la extracción de actores y un conjunto de patrones para la extracción de los casos de uso a partir de dichas reglas. Dado que los casos de uso se asocian al actor de la misma historia de usuario, al extraer los casos, se crea la relación con el actor correspondiente.

Para la extracción de actores, se usaron los siguientes patrones:

$$\begin{aligned} \text{EXC_ACT1: } & \{(\langle\text{ADP}\rangle|\langle\text{DET}\rangle|\langle\text{ADJ}\rangle)^*\} \\ \text{ACT: } & \{(\langle\text{ACT_1}\rangle|\langle\text{ACT_2}\rangle)^+\} \\ \text{ACT_1: } & \{(\langle\text{NOUN}\rangle|\langle\text{PROPN}\rangle)((\langle\text{EXC_ACT1}\rangle)+(\langle\text{NOUN}\rangle|\langle\text{PROPN}\rangle)^+)^+\} \\ \text{ACT_2: } & (\langle\text{NOUN}\rangle|\langle\text{PROPN}\rangle)^+ \end{aligned}$$

Así mismo, para la extracción de casos de uso se usaron los patrones mencionados a continuación:

$$\begin{aligned} \text{CU: } & \{\langle\text{ACT}\rangle(\langle\text{EXC_TERM.}^*\rangle)^*(\langle\text{EXC_TERM3}\rangle|\langle\text{EXC_TERM4}\rangle)^* \\ & \langle\text{VERB}\rangle\langle\text{CCONJ}\rangle^*\langle\text{VERB}\rangle^*(\langle\text{EXC_TERM1}\rangle|\langle\text{EXC_TERM2}\rangle)^*\langle\text{ACT}\rangle\} \\ \text{EXC_TERM1: } & \{(\langle\text{ADP}\rangle|\langle\text{DET}\rangle|\langle\text{ADJ}\rangle)^*\} \\ \text{EXC_TERM2: } & \{(\langle\text{ADJ}\rangle|\langle\text{DET}\rangle)^*\} \\ \text{EXC_TERM3: } & \{\langle\text{PUNCT}\rangle\langle\text{VERB}\rangle(\langle\text{AUX}\rangle|\langle\text{SCONJ}\rangle|\langle\text{PRON}\rangle)^*\} \\ \text{EXC_TERM4: } & \{(\langle\text{PROPN}\rangle|\langle\text{PRON}\rangle|\langle\text{ADV}\rangle)^*\} \\ \text{ACT: } & \{(\langle\text{ACT_1}\rangle|\langle\text{ACT_2}\rangle)^+\} \\ \text{ACT_1: } & \{(\langle\text{NOUN}\rangle|\langle\text{PROPN}\rangle)((\langle\text{EXC_TERM1}\rangle)+(\langle\text{NOUN}\rangle|\langle\text{PROPN}\rangle)^+)^+\} \\ \text{ACT_2: } & (\langle\text{NOUN}\rangle|\langle\text{PROPN}\rangle)^+ \end{aligned}$$

Donde *ACT* extrae los posibles actores o entidades que se pueden encontrar en las historias de usuario, ya sean sustantivos simples (*ACT_1*) o sustantivos compuestos (*ACT_2*). La regla *CU* extrae los casos de uso de las historias de usuario y así mismo, la relación de estos con los diferentes actores. Para esto, la regla está constituida por los actores o entidades encontrado, el verbo que relaciona a estos y una serie de reglas de excepción (*EXC_TERM1*, *EXC_TERM2*, *EXC_TERM3*, *EXC_TERM4*) que permiten determinar que partes de la historia se encuentran en medio de aquellas partes o términos que nos interesan y de esta manera, poder enfocarnos en los datos que realmente necesitamos. Al igual que en los casos anteriores, estos patrones fueron refinados por medio de la comparación de los resultados preliminares con los casos de uso y actores detectados manualmente, con el fin de detectar los casos excepcionales o aquellos que no eran tenidos en cuenta por los patrones y hacer los ajustes requeridos para incluirlos, con el fin de mejorar los resultados.

3.4.1. Procedimiento de aplicación de patrones

Para la aplicación de las reglas de patrones sobre las historias de usuario, se realizan dos procedimientos diferentes por cada historia:

Primer procedimiento: Se extrae el actor principal de la historia, donde se segmenta y se toma la primera parte de la historia de usuario, es decir, *Como [Actor]*. Posteriormente, se aplican las reglas de patrones sobre este fragmento para obtener el árbol sintáctico del mismo, y de este modo, extraer el actor presente en el mismo concatenando por medio del carácter «_» la forma singular de los diferentes términos que lo componen siempre y cuando estos sean sustantivos.

Este procedimiento da como resultado el actor correspondiente a la historia de usuario analizada y en caso de que no haya sido obtenido antes, este es almacenado con los demás actores que sean extraídos de las demás historias de usuario.

Segundo procedimiento: Consta de extraer los casos de uso presentes en las historias de usuario y así mismo, su relación con los actores correspondientes. Para esto, se toma la historia de usuario completa y es analizada por medio de todas las reglas de patrones establecidas. Acto seguido, se recorren los árboles sintácticos obtenidos en dicho análisis y si son árboles obtenidos con la regla *CU*, los componentes o términos que lo integran son recorridos.

El objetivo de este recorrido es determinar que componentes corresponden a las excepciones establecidas dentro de la regla y construir el caso de uso con aquellos que no se encuentran en estas excepciones, haciendo uso de la etiqueta proporcionada por *RegexParser* a cada componente de tipo *Árbol* que se encuentra en el recorrido. Si un componente se encuentra dentro de las excepciones de la regla, es ignorado al construir el caso de uso. Si un componente es un árbol de tipo *ACT*, los términos que lo componen son concatenados por medio del carácter «_» para construir el sustantivo correspondiente y almacenado en un arreglo temporal de la historia. Si el componente es un verbo (*VERB*), es almacenado en arreglo

temporal de la historia. En caso de que se encuentre más de un verbo durante el análisis, se construirá un arreglo temporal por cada uno de los verbos encontrados. Cada uno de estos arreglos representa la relación entre el posible actor y los casos de uso, de la siguiente manera: [Posible actor, Verbo, Sustantivo]

Al finalizar, cada arreglo temporal es almacenado en un conjunto de arreglos con los actores o sustantivos extraídos y los respectivos verbos que los relacionan. Debido a la estructura de las historias de usuario, los actores o sustantivos siempre se encontrarán al inicio o final del arreglo, lo cual facilitará su análisis posteriormente.

Al finalizar la aplicación de ambos procedimientos sobre todas las historias de usuario, se recorren los diferentes arreglos obtenidos y se valida si el primer sustantivo de cada arreglo (La primera posición) se encuentra entre la lista de actores obtenidos por medio del primer procedimiento. De ser así, el caso de uso se construye a partir del verbo y el segundo sustantivo del arreglo correspondiente, modificando el arreglo para tener la siguiente forma: [Actor, Caso de uso (Verbo + Sustantivo)] y se almacena en un conjunto final de casos de uso.

Para finalizar, los casos de uso son agrupados de acuerdo al actor al que pertenecen y como resultado, se obtiene un conjunto de casos de uso por cada actor detectado en las historias de usuario.

Con esto, se estaría cumpliendo el segundo objetivo específico, el cual consta de diseñar un modelo computacional de clasificación en clases, atributos, relaciones, actores y casos de uso basado en reconocimiento de patrones en la estructura de texto para la extracción de los principales componentes de los diagramas objetivo.

3.5. Generación de diagramas de clase y casos de uso

Una vez que se han obtenido los componentes de los diagramas objetivo, estos son procesados para la construcción de los respectivos diagramas. Para este proceso se utiliza la

herramienta *OpenSource* PlantUML ², la cual permite generar múltiples tipos de diagramas UML (y no-UML) a partir de archivos de texto con un formato determinado de acuerdo al diagrama que se construirá. En la figura 3-5 se puede observar el flujo de este procedimiento.

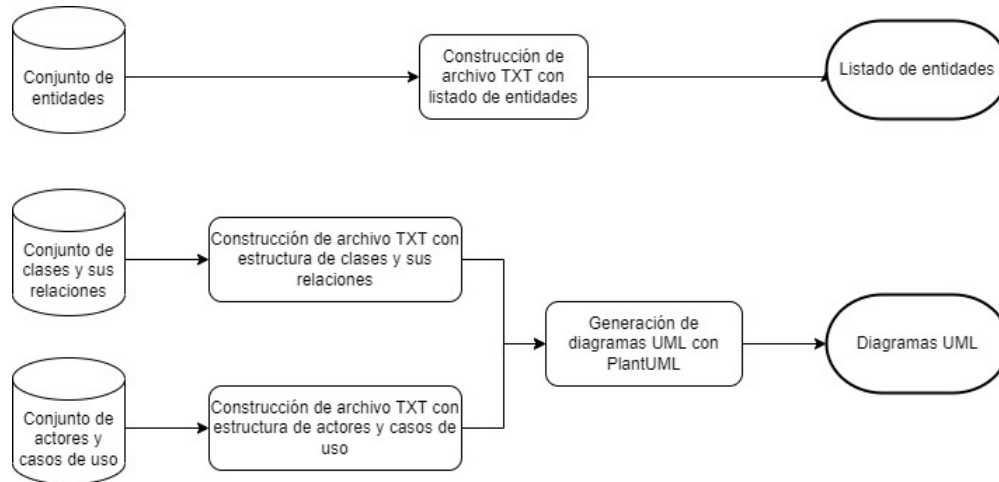


Figura 3-5.: Flujo de proceso para la generación de diagramas de UML a partir de los componentes extraídos [Autor]

Dado que los componentes extraídos se encuentran almacenados en conjuntos, deben ser recorridos e introducidos dentro de archivos de texto que son construidos de acuerdo al formato correspondiente al diagrama que se construirá o la información que se mostrará en dicho archivo.

3.5.1. Entidades

Las entidades no se incorporan en un diagrama UML, sino que son impresas en el archivo de texto en formato lista. Para esto, todas las entidades son impresas en el archivo en orden descendiente de acuerdo a su frecuencia dentro del conjunto de historias, de modo que las entidades más frecuentes estén en los primeros lugares y las menos frecuentes están en los últimos lugares. Este archivo se construye, se almacena y se nombra con el siguiente formato

²Página oficial de PlantUML en español: <https://plantuml.com/es/>

[Nombre del conjunto]-Entidades.txt. En la figura 3-6 se pueden observar algunas de las entidades más frecuentes encontradas en el conjunto de datos del archivo *US25*.

```
----- LISTA DE ENTIDADES -----  
* A continuación encontrara la lista de entidades presentes en las  
historias de usuario. Estan organizadas de la entidad mas frecuente a  
la menos frecuente *  
movilidad_urbana  
planificador  
medida  
efecto  
tráfico  
propuesta  
congestión  
nivel  
solución  
validación  
contaminación  
consecuencia  
efecto_ambiental  
equidad  
evento
```

Figura 3-6.: Ejemplo de las entidades más frecuentes obtenidas para el archivo *US25* [Autor]

3.5.2. Clases, atributos, métodos y relaciones

Estos componentes se encuentran almacenados en un conjunto en el que los métodos y atributos están agrupados según la clase a la que pertenecen, por lo cual se recorre dicho conjunto para construir el archivo de texto correspondiente y la información de cada clase de adiciona en el siguiente formato:

```
class NombreClase1 {  
    Atributo1  
    Atributo2  
    ...  
    Metodo1()  
    Metodo2()  
}
```

Posteriormente, se adicionan las relaciones detectadas entre las diferentes clases. Estas relaciones están almacenadas en el mismo conjunto de las clases, por lo que son recorridas para ser agregadas en el archivo de la siguiente manera, de acuerdo al tipo de relación y el par de clases a los cuales relaciona:

```
NombreClase1 -- NombreClase2 (Asociación)
NombreClase3 o-- NombreClase4 (Agregación)
NombreClase5 *-- NombreClase6 (Composición)
NombreClase7 <|-- NombreClase8 (Herencia)
NombreClase9 ..> NombreClase10 (Dependencia)
```

Una vez que se han agregado todas las clases y relaciones obtenidas en el archivo, este es almacenado y nombrado con el formato `[Nombre del conjunto]-Clases.txt`.

Acto seguido, se ejecuta el siguiente comando para generar el diagrama de clases por medio de PlantUML:

```
python3 -m plantuml [Nombre del archivo creado]
```

Este comando analiza la estructura del archivo y genera el correspondiente diagrama de clases retornando una imagen PNG en la cual se puede visualizar el diagrama obtenido. En la figura **3-7** se puede observar un ejemplo de como se estructuran las clases y sus relaciones dentro del archivo de texto. En la figura **3-8** se observa el diagrama resultado del procesamiento de PlantUML sobre el archivo de texto con la estructura de clases.

3.5.3. Actores y casos de uso

Los actores y casos de uso detectados anteriormente se recorren para construir los correspondientes diagramas de caso de uso. En este caso, se crea un archivo por cada actor, por lo tanto, cada actor estará representado en un diagrama de casos de uso distinto.

Para esto, por cada actor, se construye un archivo de texto con la siguiente estructura:

```
:NombreActor: --> (Caso de uso 1)  
:NombreActor: --> (Caso de uso 2)  
:NombreActor: --> (Caso de uso 3)  
...
```

Al terminar de construir el archivo de texto del actor, este es almacenado y nombrado con el formato [Nombre del conjunto]-CasoUso-[Nombre del actor].txt .

Posteriormente, se ejecuta el comando para la generación de diagramas con PlantUML usado anteriormente con las clases, excepto que en esta ocasión se ejecutará para cada uno de los archivos de texto creados con los diferentes casos de uso.

Al finalizar, se tendrá un conjunto de imágenes PNG correspondientes a los diagramas de casos de uso generados para cada uno de los actores. En la figura **3-9** se puede observar parte de la estructura utilizada para el análisis y construcción de estos diagramas y en la figura **3-10** se puede observar el resultado de dicha estructura. En la estructura de los casos de uso, el comando `left to right direction` indica que el actor estará a la izquierda apuntando hacia los casos de uso a su derecha.

Con esto, se estaría cumpliendo el tercer objetivo específico, el cual consta de generar los diagramas de clase y casos de uso, partiendo de los resultados obtenidos en la extracción de sus componentes en los procedimientos anteriores, utilizando librerías de transformación de archivos a diagramas UML, en este caso se utilizó PlantUML.

```
class ifa {
    empleado
    administrador
    gerente
    presupuesto
    unir_sistema()
    administrar_liga()
    ingresar_conjunto_regla_competición()
    evitar_sesgo()
    evitar_conflicto_horario_juego()
}
class partido {
    oficial
    estadística
    evento
    partido
    acceder_sistema()
    tener_cuenta_franja()
    informar_evento()
}
class juego {
    partido
    horario
    evento
}
...
error -- activo
ifa -- liga
lugar -- dato
administrador -- liga
administrador -- equipo
gerente -- equipo
gerente -- liga
...
```

Figura 3-7.: Ejemplo de estructuración de clases para su análisis con PlantUML [Autor]

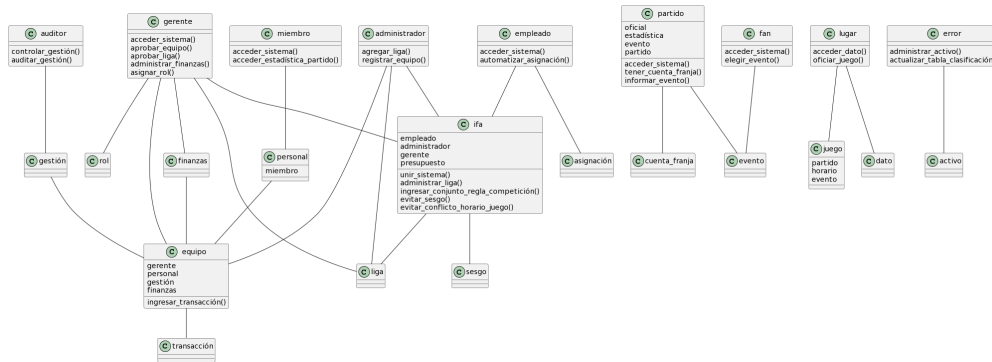


Figura 3-8.: Ejemplo de diagrama de clases creado por medio de PlantUML [Autor]

```

left to right direction
skinparam actorStyle awesome
:repcionista: --> (asignar cama_paciente)
:repcionista: --> (asignar médico_paciente)
:repcionista: --> (reservar cita_paciente_médico)
    
```

Figura 3-9.: Ejemplo de estructuración de casos de uso para su análisis con PlantUML [Autor]

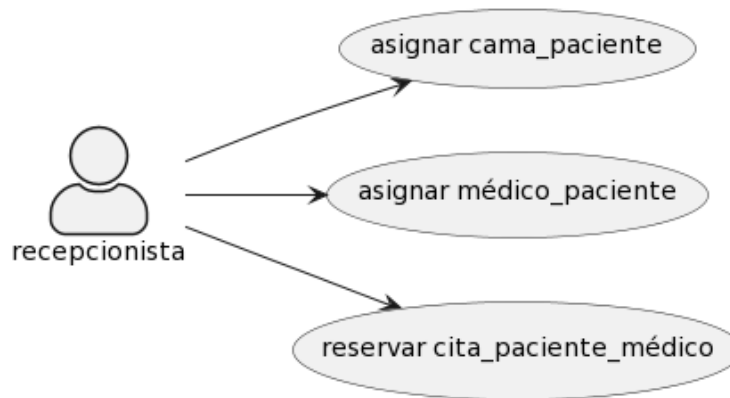


Figura 3-10.: Ejemplo de diagrama de casos de uso creado por medio de PlantUML [Autor]

4. Evaluación de resultados

Dado que el conjunto de datos obtenido para este trabajo consta de múltiples conjuntos de historias de usuario y las imágenes de los diagramas UML construidos a partir de las mismas, se construyó un archivo JSON en el cual se ingrese la información de los componentes de los distintos diagramas del conjunto de datos, con el objetivo de mejorar el proceso de comparación de resultados entre los diagramas construidos manualmente y los diagramas construidos por el sistema.

El archivo JSON construido cuenta con el siguiente formato para los distintos diagramas:

```
{
  "DC": {
    "Clases": {
      "Clase1": {
        "Atributos": ["Atributo1", "Atributo2"],
        "Metodos": ["Metodo1", "Metodo2"]
      },
      "Clase2": {
        "Atributos": ["Atributo1", "Atributo2"],
        "Metodos": ["Metodo1", "Metodo2"]
      }
    },
    "Relaciones": [
      ["Clase1", "Clase2", "ASOC"],
      ["Clase3", "Clase4", "AGR"],
    ]
  }
}
```

```
    ["Clase4", "Clase6", "COMP"],
    ["Clase5", "Clase8", "HER"],
    ["Clase6", "Clase10", "DEP"]
  ]
},
"UC": {
  "Actor1": {
    "Casos": [
      ["1", "CU", "Caso 1..."],
      ["1.1", "CU", "Caso 2..."],
      ["1.1.1", "CU", "Caso 3..."],
      ["1.2", "CU", "Caso 4..."],
      ["2", "CU", "Caso 5..."],
      ["2.1", "CU", "Caso 6..."],
      ["2.2", "CU", "Caso 7..."],
      ["2.2.1", "A", "Actor2"]
    ],
    "Relaciones": [
      ["1", "1.1", "I"],
      ["1", "1.2", "I"],
      ["1.1", "1.1.1", "E"],
      ["2", "2.1", "E"],
      ["2", "2.2", "I"],
      ["2.2", "2.2.1", "I"]
    ]
  }
},
"Entities": ["Entidad1", "Entidad2"]
}
```

Donde **DC** contiene las clases, sus componentes y relaciones entre ellas. **UC** contiene los actores y casos de uso detectados manualmente en el conjunto de datos. Cada caso de uso tiene un identificador que lo posiciona dentro de la jerarquía de casos que constituyen el diagramas de casos de uso. Por esto mismo, **UC** también posee relaciones, las cuales indican como se conectan los distintos casos de uso de cada actor. También se pueden encontrar las entidades extraídas de cada conjunto de historias de usuario, de modo que este archivo contiene toda la información sobre las entidades, clases y casos de uso de cada conjunto de historias analizado, ya que por cada conjunto se cuenta con un formato de este tipo.

Haciendo uso de este archivo y los resultados del sistema, se evalúan estos últimos por medio de las métricas Precisión, *Recall* y F1, las cuales son calculadas por medio de los siguientes elementos:

- **True Positive (TP)**: Los elementos o componentes que fueron predichos por el sistema y son similares o iguales a componentes obtenidos manualmente.
- **False Positive (FP)**: Los elementos o componentes que fueron predichos por el sistema pero no son similares a los componentes obtenidos manualmente.
- **False Negative (FN)**: Los elementos o componentes que fueron obtenidos manualmente, mas no se encuentran o no son similares a los resultados del sistema.

Una vez que se obtienen estos elementos, se procede a calcular cada una de las métricas de la siguiente manera:

- **Precisión**: Indica la cantidad de elementos predichos correctamente entre todos los elementos predichos.

$$Precision = \frac{TP}{TP + FP} \quad (4-1)$$

- **Recall**: Indica el porcentaje de elementos predichos correctamente por la herramienta comparado con los elementos obtenidos manualmente.

$$Recall = \frac{TP}{TP + FN} \quad (4-2)$$

- **F1:** Es la combinación entre Precisión y *Recall*, permitiendo establecer un valor con base en los valores de estas dos métricas, dándole la misma importancia a ambas.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (4-3)$$

Con base en estas ecuaciones, se procede a evaluar los resultados para cada grupo de componentes detectado por el sistema y el rango de umbrales de similaridad `UMBRAL = [0.5,0.55,0.6,0.65,0.7,0.75,0.8,0.85,0.9,0.95,]`, con los cuales se determinará el comportamiento del sistema al establecer diferentes valores mínimos de similaridad entre los términos o componentes comparados.

4.1. Extracción de entidades

En la figura 4-1 se puede observar el flujo del proceso realizado para realizar la comparación entre las entidades obtenidas por el sistema y las entidades obtenidas manualmente en el conjunto de datos.

En este proceso, se recorren las entidades detectadas por el sistema y cada una es comparada con las entidades obtenidas de manera manual. Para realizar la comparación entre las diferentes entidades, ambas entidades son transformadas a minúsculas para tratar de que estén la mayor cantidad de mismas condiciones para la comparación y se emplea el método *Similarity* ofrecido por la librería *Spacy*. Esta ofrece múltiples herramientas y métodos para el procesamiento de lenguaje natural, desde tokenización a extracción de posibles dependencias entre palabras. El método *Similarity* toma dos textos y compara la similitud entre estos por medio de representaciones multidimensionales de palabras o *Word Embeddings* y retornará un número decimal que indica el porcentaje de similitud entre ambos textos, siendo 0 el resultado más bajo cuando no hay similitud entre los textos y 1 el resultado más alto cuando

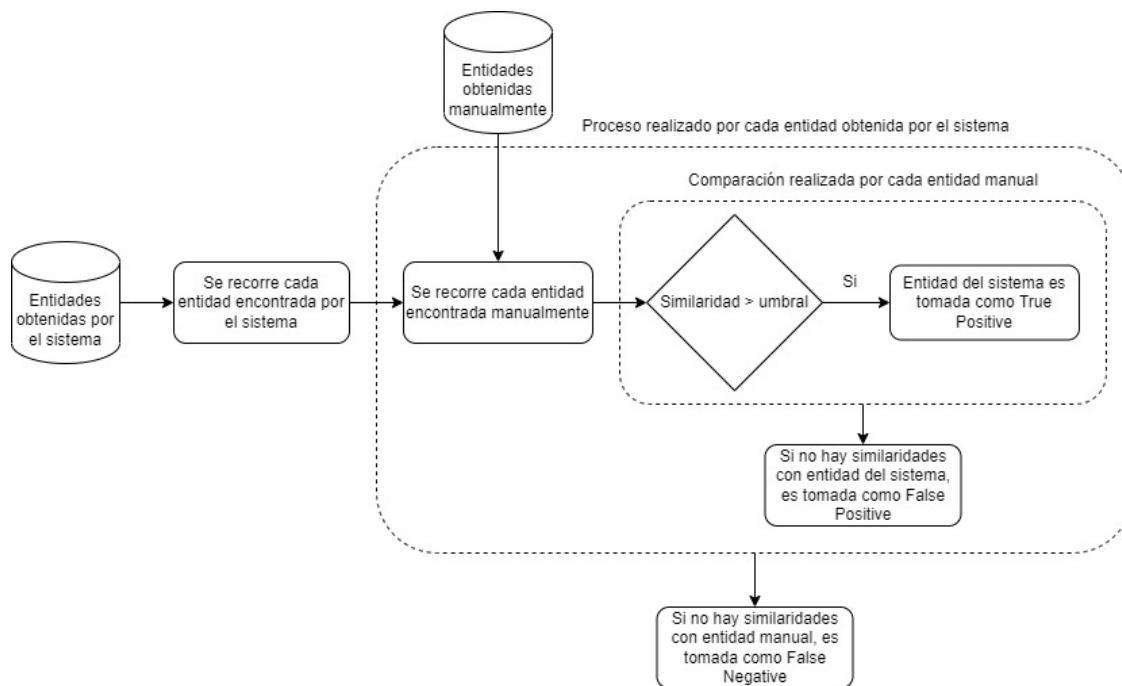


Figura 4-1.: Flujo de comparación de resultados obtenidos en extracción de entidades [Autor]

son iguales. Para la evaluación de las entidades, se establece un umbral de similitud para determinar que los términos son similares. En caso de que si haya una similitud igual o superior a la esperada en el umbral, la entidad del sistema es añadida al conjunto de resultados **True Positive**. Si al terminar de recorrer las entidades manuales no se ha detectado una similitud con las entidades detectadas por el sistema, esta es añadida al conjunto de resultados **False Positive**. Por último, si al finalizar de recorrer las entidades del sistema, no se encontró una similitud con alguna entidad detectada manualmente, esta sera añadida al conjunto de resultados **False Negative**. Este proceso es realizado por cada uno de los conjuntos de historias de usuario.

Para realizar este proceso de evaluación, se realiza la comparación teniendo en cuenta cada uno de los umbrales establecidos en el rango *UMBRAL* y el rango de cantidades *CANTIDAD_ENTIDADES* = [25, 50, 75 , 100] con el que se evaluará como se comporta el software y los resultados obtenidos al comparar las primeras entidades del listado obtenido por el sistema, de acuerdo a la cantidad establecida por el rango, es decir, tomando las

primeras 25 entidades o las primeras 50 entidades retornadas por el sistema.

En la figura 4-2 se pueden observar la precisión, *Recall* y F1 de los resultados obtenidos al cambiar los umbrales de similitud y la cantidad de entidades tomadas para la comparación.

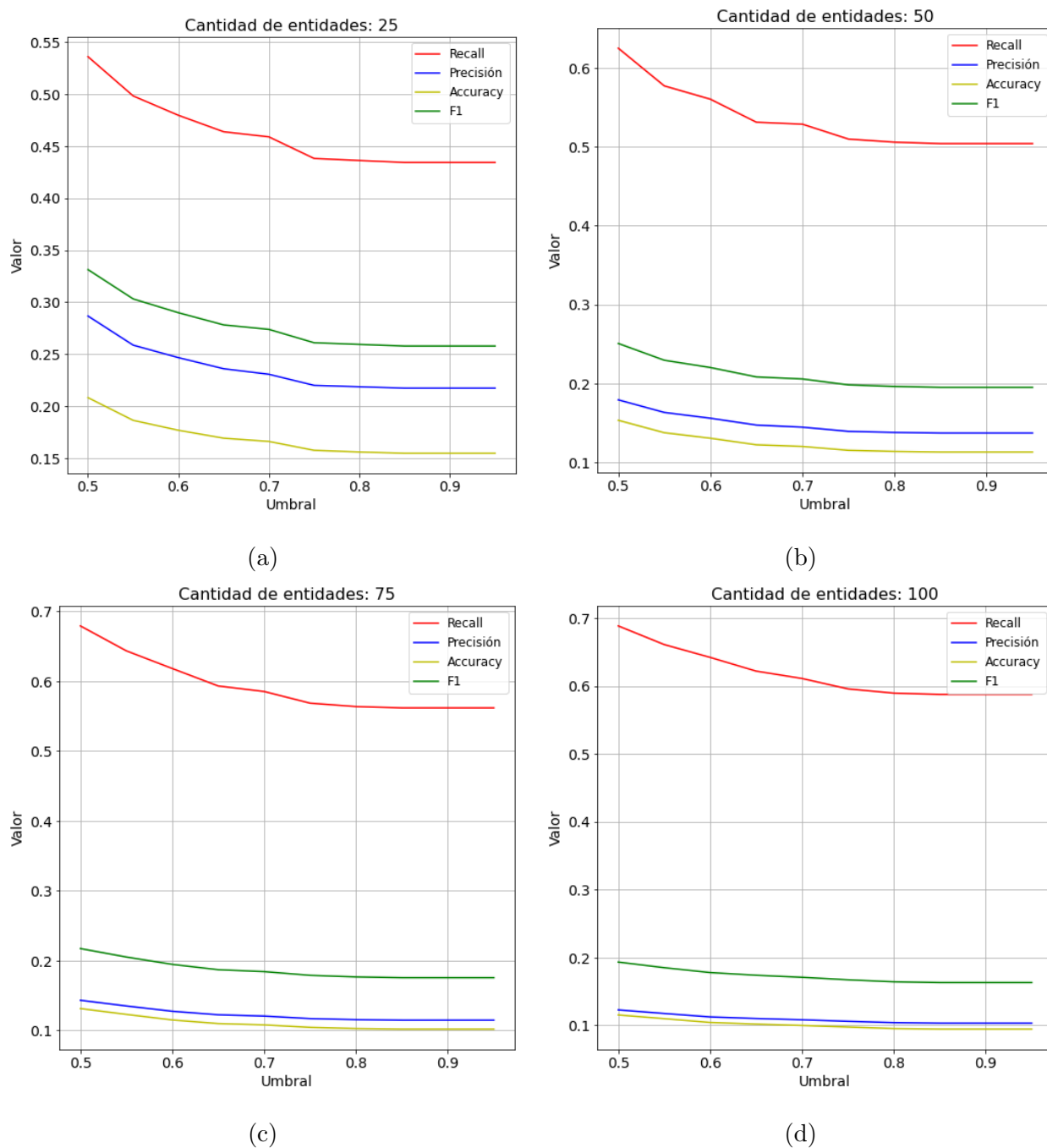


Figura 4-2.: Gráficas de precisión, *Recall*, *Accuracy* y F1 correspondientes a la cantidad de entidades tomadas del resultado del sistema

Como se puede observar, al tomar las 25 entidades más frecuentes detectadas por el sistema (Figura 4-2a) y ser evaluadas con el umbral de similaridad 0.5, el *Recall* promedio tiene un valor cercano al 0.54, mientras que la precisión promedio se encuentra cerca del 0.29. Con base en estos, el valor promedio de F1 se acerca al 0.33. Sin embargo, a medida que aumentamos el umbral de similaridad, los valores decrecen gradualmente llegando a un valor aproximado de 0.44 para el *Recall* cuando el umbral es de 0.95, mientras que el valor de la precisión llega cerca del 0.22 y el valor promedio de F1 se acerca al 0.25.

Tan pronto como aumentamos en la evaluación la cantidad de entidades más frecuentes encontradas por el sistema, el valor del *Recall* tiende a aumentar, mientras la precisión y F1 tienden a disminuir. Esto lo podemos evidenciar al tomar las 100 entidades más frecuentes encontradas por el sistema (Figura 4-2d), denotando que el *Recall* es cercano al 0.7 en el umbral más bajo y desciende hasta llegar aun valor levemente inferior al 0.6 en el umbral más alto. Así mismo, la precisión y el F1 tienen valores aproximados de 0.12 y 0.2 respectivamente cuando el umbral es de 0.5 y valores cercanos a 0.1 y 0.16 respectivamente cuando el umbral es de 0.95.

Podemos notar que los valores de las métricas disminuyen, debido a que al aumentar el umbral de similaridad, disminuye la probabilidad de que dos entidades sean consideradas como similares en la evaluación, por lo cual, la cantidad de entidades detectadas manualmente que se encuentren dentro de las entidades obtenidas por el sistema disminuirá con el aumento del umbral. Sin embargo, el *Recall* nos indica que aun cuando se toma una cantidad reducida de entidades encontradas por el sistema, al menos cerca de la mitad de las entidades encontradas manualmente se encuentran dentro de las clases obtenidas por el sistema. No obstante, la precisión y por consiguiente, el F1 del extractor de entidades se ven afectados por los términos que fueron detectados erróneamente como entidades, a causa de que poseen una estructura gramatical similar a las entidades y no es posible establecer una diferencia clara entre estos términos y las entidades, debido a que no se cuenta con la suficiente información.

Todo esto es aun más notable cuando aumentamos la cantidad de entidades tomadas para la evaluación, pues a mayor cantidad de entidades, el valor del *Recall* aumenta dado que se incrementa la posibilidad de encontrar las entidades detectadas manualmente, dentro del

conjunto de entidades encontradas por el sistema, por lo cual esta métrica tiende a aumentar y estabilizarse después de el umbral de similaridad con valor 0.8. Por otro lado, los valores de la precisión y el F1 se ven aun más afectados por este aumento, pues la cantidad de términos detectados erróneamente que se tienen en cuenta es mayor y no es compensado por el aumento del *Recall*, por lo cual, la precisión disminuye en cuanto aumentan los términos detectados erróneamente y el F1 se ve afectado por dicha disminución.

4.2. Extracción de clases, atributos, métodos y relaciones

4.2.1. Clases

Para la comparación de las clases obtenidas por el sistema frente a las clases obtenidas de manera manual, se siguió el proceso ilustrado en la figura 4-3. En este proceso, las

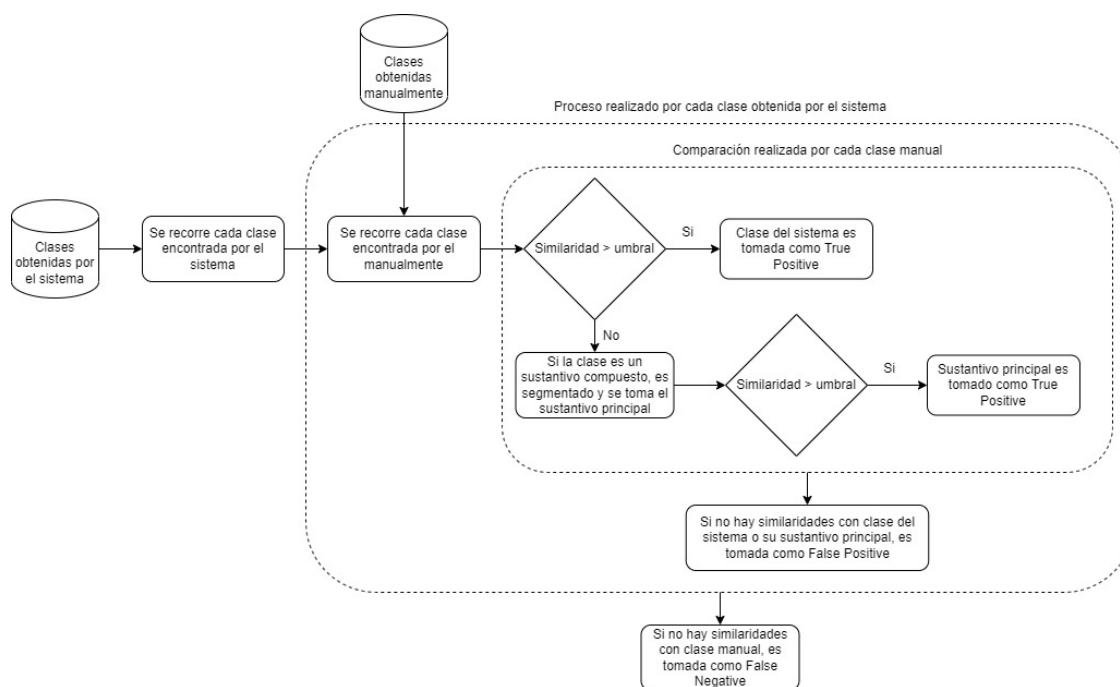


Figura 4-3.: Flujo de comparación de resultados obtenidos en extracción de clases [Autor]

clases obtenidas por medio del sistema son recorridas y cada una es comparada con todas las clases obtenidas manualmente, en busca de similitudes entre los elementos de ambos conjuntos. Al igual que con las entidades, se utiliza la función *Similarity* de *Spacy* para

determinar si la similaridad entre una clase del sistema y una clase manual supera un umbral determinado. Si el resultado de la comparación entre las dos clases es mayor o igual al umbral, se determina que la clase del sistema es un ***True Positive***. Si el resultado es menor al umbral y la clase es un sustantivo compuesto, se toma el primer sustantivo y se realiza la comparación nuevamente con la clase manual. Ej: Si la clase es *Usuario_administrador*, se toma el sustantivo *Usuario* y se realiza la comparación. Si esta segunda comparación supera o es igual al umbral, se determina que la clase del sistema es ***True Positive***.

Si al terminar de recorrer las clases manuales, aun no se ha encontrado una similaridad para la clase del sistema, se determina que esta es un ***False Positive***. Por último, al terminar se recorrer todas las clases del sistema, no se encuentra similaridad para una clase obtenida manualmente, se determina que esta es un ***False Negative***.

Para evaluar los resultados de la extracción de clases, se realiza la comparación de las clases del sistema y las clases manuales con cada uno de los umbrales establecidos en *UMBRAL*.

Adicionalmente, se realizaron tres evaluaciones diferentes para las clases:

- **Evaluación #1:** En esta evaluación se toma una determinada cantidad de clases encontradas por el sistema, aplicando solo los patrones de detección de clases y se realiza la comparación de los resultados con las clases tomadas. En esta no se tienen en cuenta relaciones, atributos o métodos de las clases, sino solamente el conjunto de datos detectado por el sistema por medio de los patrones. Las cantidades que se tomaron son las 25, 50, 75 y 100 clases más frecuentes encontradas por el sistema (Figura 4-4).
- **Evaluación #2:** En esta evaluación se toman todas las clases encontradas por el sistema, siempre y cuando estén relacionadas con otra clase (Figura 4-5).
- **Evaluación #3:** En esta evaluación se toman todas las clases detectadas por el sistema que tengan un atributo o método y estén relacionadas a otra clase (Figura 4-6).

En la figura 4-4, podemos observar que al no tener en cuenta mayor información sobre las clases y evaluar las 25 clases más frecuentes encontradas por el sistema (Figura 4-4a) con el umbral de similaridad en 0.5, el *Recall* del modelo se encuentra sobre 0.8, indicando que hay

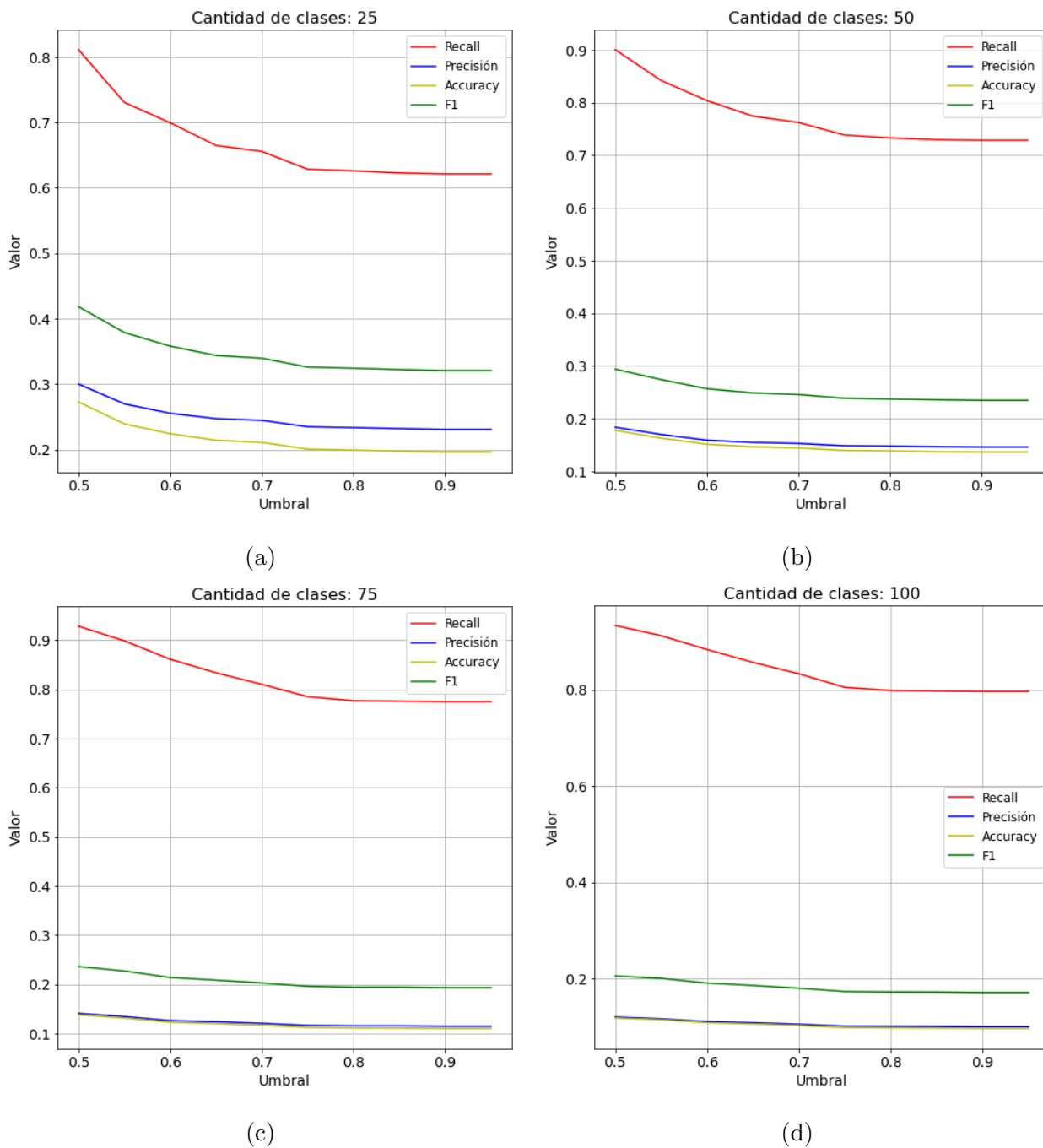


Figura 4-4.: Gráficas de precisión, *Recall*, *Accuracy* y F1 correspondientes a la evaluación #1 de las clases [Autor]

una alta probabilidad de que las clases detectadas manualmente encuentren similitudes dentro de las 25 clases evaluadas. No obstante, la precisión en este caso se encuentra en 0.3,

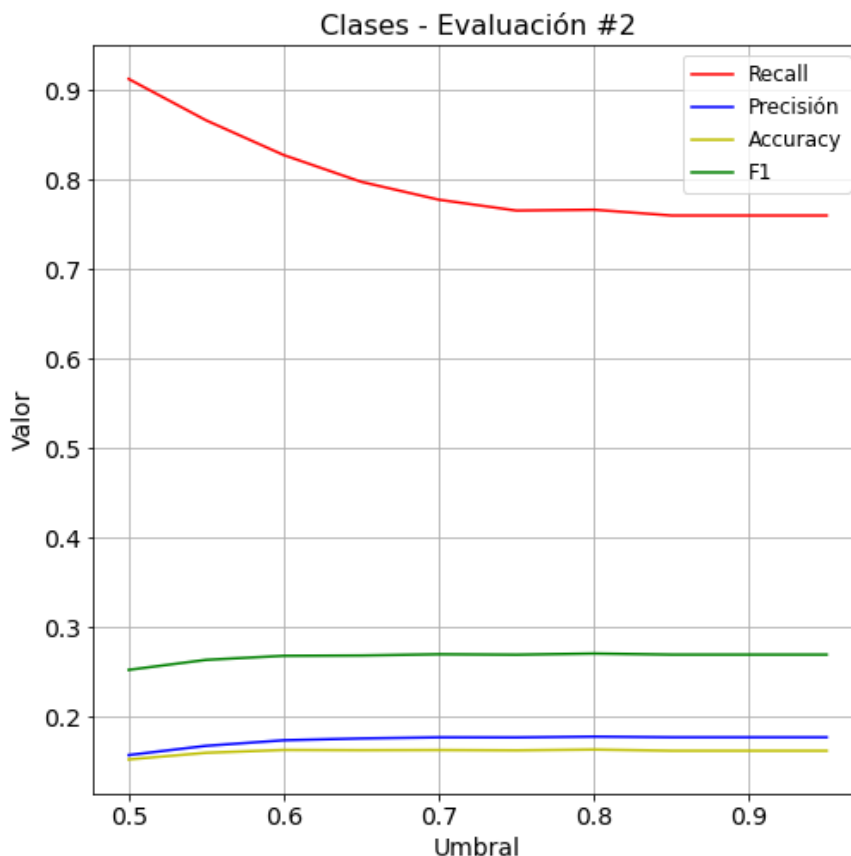


Figura 4-5.: Gráficas de precisión, *Recall*, *Accuracy* y F1 correspondientes a la evaluación #2 de las clases [Autor].

dado que se ve afectadas por aquellas clases detectadas erróneamente por el sistema, a causa de términos o entidades que compartir una similitud gramatical con las clases lo suficiente alta para no poder marcar una diferencia clara entre estas y poderlas excluir de manera adecuada. Sin embargo, al aumentar el umbral de similitud, las métricas se ven afectadas debido a que hay una menor probabilidad de que las clases encontradas manualmente sean similares a las clases encontradas por el sistema y con esto, aumenta la cantidad de términos detectados erróneamente por el sistema, como lo podemos constatar al realizar la evaluación con el umbral en 0.95, donde el *Recall* llega a acercarse al 0.6 y la precisión decrece a poco menos de 0.25.

Esto podemos apreciarlo con mayor detalle en la figura 4-4d, en la cual se evalúan las 100 clases más frecuentes encontradas por el sistema. Aquí podemos apreciar como el *Recall* inicia

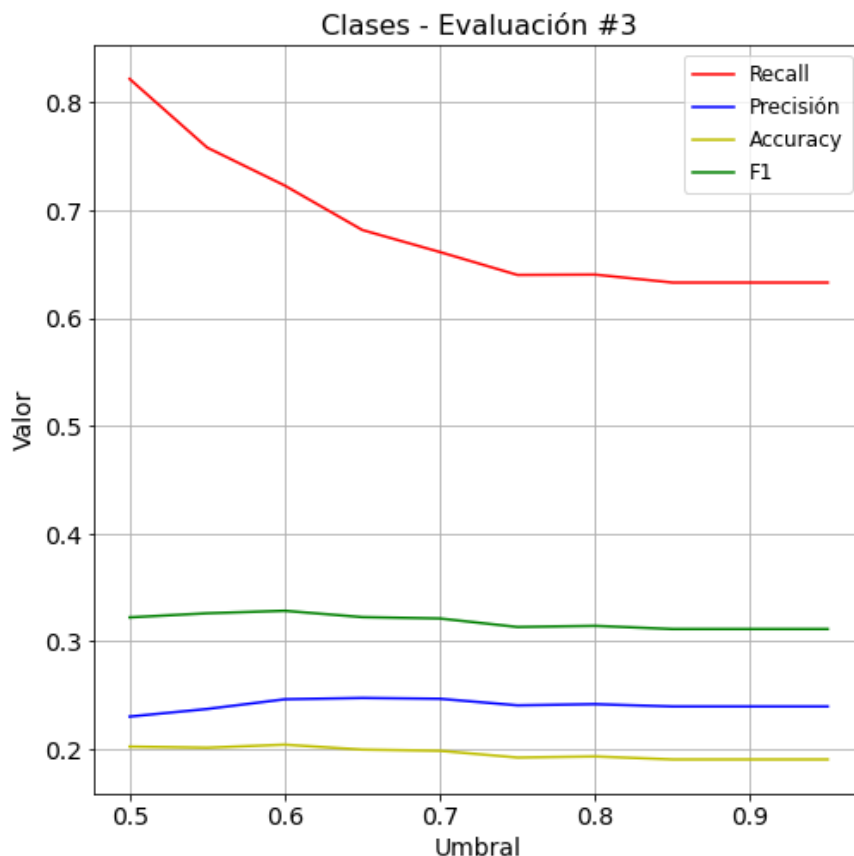


Figura 4-6.: Gráficas de precisión, *Recall*, *Accuracy* y F1 correspondientes a la evaluación #3 de las clases [Autor].

con cerca de 0.9 en el umbral de similitud 0.5 y decrece gradualmente hasta estabilizarse cerca de 0.8 desde el umbral de 0.75, debido a que al contar con una mayor cantidad de clases encontradas por el sistema, aumenta la posibilidad de encontrar las clases manuales dentro del conjunto de clases obtenidas por el sistema y en el caso, en cualquier umbral hay una gran probabilidad de encontrar las clases correctas dentro de las clases detectadas.

No obstante, la precisión se ve drásticamente afectada por el aumento de clases tomadas, pues aumenta en gran medida la cantidad de términos detectados erróneamente y la cantidad de clases detectadas correctamente no aumenta en la misma proporción o en mayor medida, por lo cual la precisión (y por consiguiente, el F1) disminuye drásticamente, iniciando cerca de 0.1 cuando el umbral de similitud es 0.5 y decreciendo hasta llegar cerca de 0.05 con el umbral en 0.95.

En la figura 4-5, podemos observar como se ven afectados los resultados al tener en cuenta las relaciones entre clases, pues en la evaluación #2 se evalúan todas las clases obtenidas por el sistema, omitiendo aquellas clases que no tienen relación con ninguna otra. En este caso, podemos observar que el *Recall* del modelo supera el 0.9 cuando el umbral de similaridad está en 0.5, sin embargo va decreciendo gradualmente a medida que aumenta el umbral hasta acercarse a 0.75 en el umbral de 0.95. Por otro lado, la precisión del modelo en este caso tiende a aumentar levemente, pues en el umbral 0.5 tiene un valor cercano a 0.15 y a medida que aumenta el umbral de similaridad, mantiene un crecimiento leve hasta acercarse a 0.18 al llegar al umbral 0.95. Debido a la disminución en el *Recall* y el aumento leve de la precisión, el modelo presenta un F1 que inicia en 0.25 en el umbral más bajo y crece levemente hasta acercarse al 0.27 cuando se presenta el umbral más alto.

Debido a que en este caso se tienen en cuenta las relaciones entre clases, gran parte de los términos detectados erróneamente son removidos de los resultados al no tener una relación con algún otro término. Esto permite realizar un filtrado más sólido de los resultados obtenidos en la evaluación anterior y se presenta una mejora en las métricas de desempeño. Al tomar todas las clases detectadas por el sistema y mantener solo aquellas que estén relacionadas, aumenta la posibilidad de encontrar entre estas las clases detectadas manualmente, por lo cual, el *Recall* presenta un aumento en el umbral más bajo respecto a los resultados mencionados en la anterior evaluación y decrece gradualmente hasta estabilizarse en 0.75, que es un valor relativamente alto en el cual se puede asegurar que gran parte de las clases manuales fueron detectadas por el sistema. Así mismo, la precisión presenta una leve mejora, pues al disminuir la cantidad de términos detectados erróneamente, mejora la capacidad del modelo para extraer las clases correctas. Con base en esto, el F1 también se ve afectado de manera positiva, pues a diferencia de la mayoría de resultados de la evaluación anterior, se mantiene sobre 0.3 a través del aumento del umbral de similaridad, a causa de la reducción de ruido en los resultados.

Si además de las relaciones, se tienen en cuenta los métodos y atributos de las diferentes clases, en la figura 4-6 podemos observar que el *Recall* disminuye un poco respecto a los resultados anteriores. En este caso, el *Recall* es cercano al 0.82 cuando el umbral está en 0.5

y decrece gradualmente al aumentar el valor del umbral, hasta aproximarse al 0.64 cuando el umbral de similaridad es 0.95. Sin embargo, la precisión tuvo un aumento notable, pues con el umbral en 0.5 tuvo un valor aproximado de 0.24 y tiende a mantenerse estable con el aumento del umbral. Con base en esto, el F1 del modelo presenta una disminución leve durante la evaluación, con un valor cercano a 0.32 en el umbral más bajo y descendiendo levemente hasta un valor cercano a 0.31 en el umbral más alto.

La disminución en el *Recall* se fundamenta en la exclusión de términos o clases detectadas por el sistema al no tener atributos y métodos, pues entre estos términos excluidos se pueden encontrar clases que fueron obtenidas de manera manual en los diagramas de clase, pero a las cuales el sistema no logró detectarles sus respectivos componentes, por lo cual son excluidas de los resultados, afectando el *Recall* y disminuyendo su valor. No obstante, la precisión presentó una mejora, pues al aplicar un filtro más profundo sobre los resultados con base en la información de las clases, aumenta la cantidad de términos detectados erróneamente que son excluidos, mejorando la capacidad de detección de clases del modelo.

Con base en los resultados de las evaluaciones, podríamos inferir que si tomáramos las 25 clases más frecuentes encontradas por el sistema, podríamos encontrarnos con mejores resultados, pues en este conjunto es más probable encontrar las clases obtenidas manualmente, aunque posiblemente 80 % de las clases encontradas podrían ser términos erróneos. Sin embargo, el sistema requiere tener mayor información sobre estas como sus atributos y métodos, con el objetivo de construir los correspondientes diagramas, por lo cual, los resultados presentados en la evaluación #3 (Figura 4-6) representan una mejor alternativa para el modelo para poder mantener las relaciones entre las clases y los componentes asociados a estas de la manera más cercana a los componentes obtenidos de manera manual.

4.2.2. Atributos

Como se puede observar en la figura 4-7, el flujo de evaluación de los atributos es muy similar al proceso de evaluación de las clases, en el cual los atributos obtenidos por el sistema son comparados con los atributos obtenidos manualmente y según la similaridad de estos, se determina si un elemento es *True Positive*, *False Positive* o *False Negative*. No

obstante, al evaluar los atributos no se realiza una doble verificación con los posibles sustantivos compuestos, sino que se hace la determinación con la similitud existente entre los atributos tal como los detecta el sistema y tal como se obtuvieron manualmente.

Sin embargo, los atributos son comparados por cada una de las clases a las cuales pertenecen y la similitud entre estas. Es decir, si al evaluar las clases se determinó que dos clases son similares, los atributos de ambas clases se compararan entre si y no con los atributos de las demás clases que no fuesen similares.

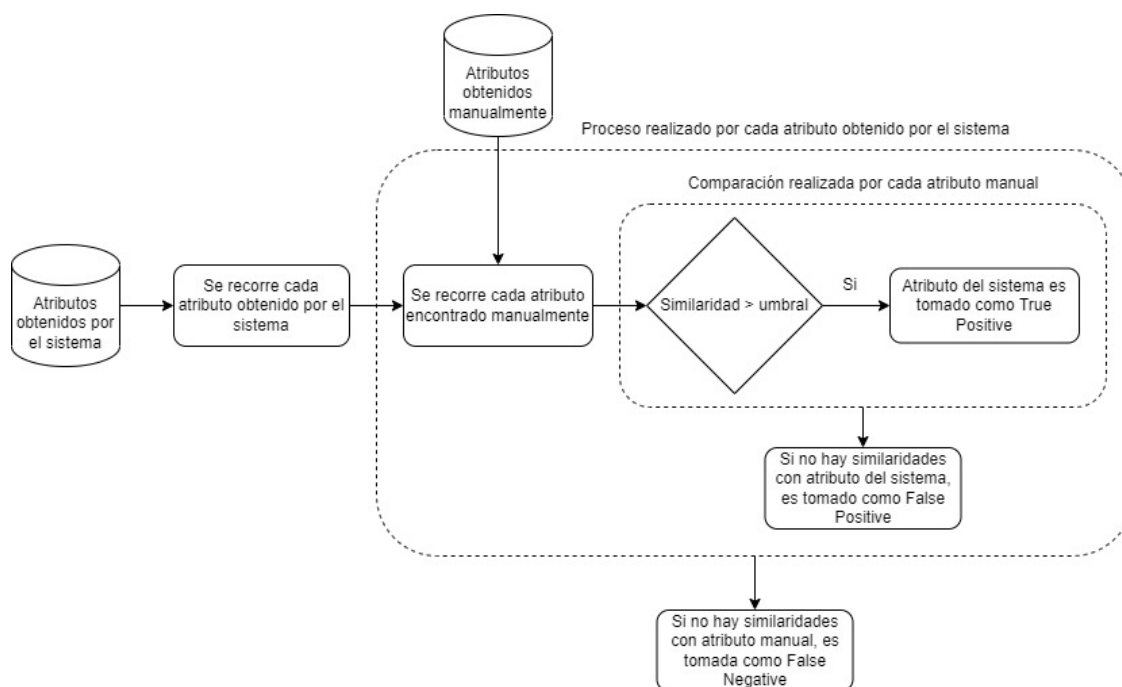


Figura 4-7.: Flujo de comparación de resultados obtenidos en extracción de atributos de la clase [Autor]

Con base en esto, se realizaron dos evaluaciones distintas de acuerdo a la información de las clases:

- **Evaluación #1:** En esta evaluación se toman los atributos de las clases encontradas por el sistema, siempre y cuando estén relacionadas por otra clase (Figura 4-8).
- **Evaluación #2:** En esta evaluación se toman los atributos de las clases detectadas por el sistema que tengan un atributo o método y estén relacionadas a otra clase (Figura 4-9).

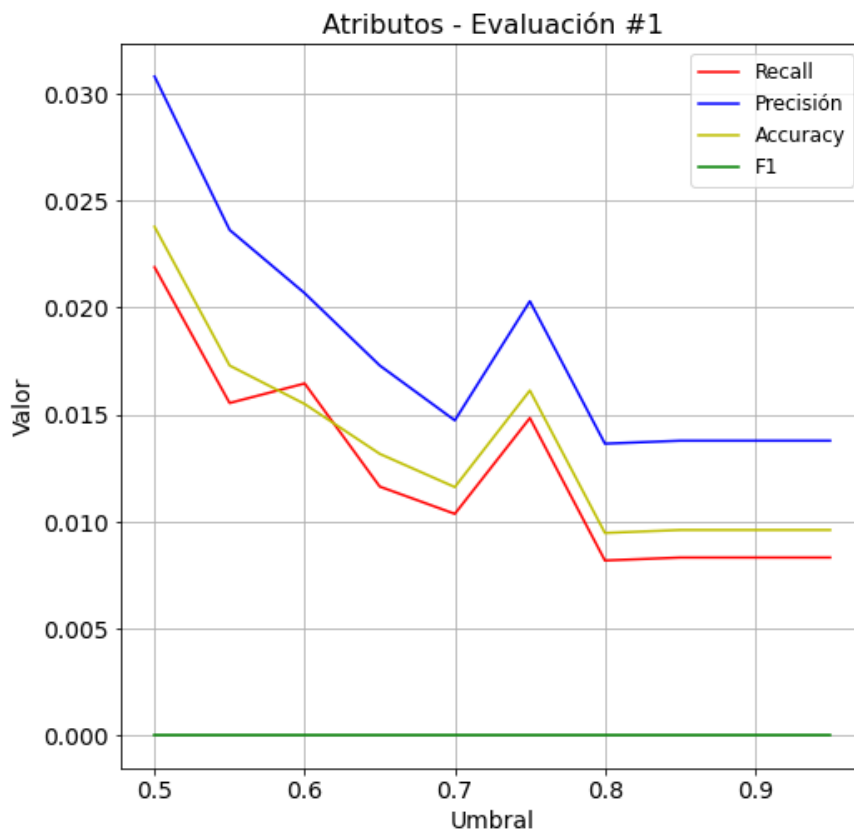


Figura 4-8.: Gráficas de precisión, *Recall*, *Accuracy* y F1 correspondientes a la evaluación #1 de los atributos [Autor].

En la figura 4-8 podemos observar que el desempeño del modelo al momento de extraer los atributos se reduce drásticamente frente a su desempeño extrayendo clases, pues teniendo en cuenta solo los atributos de las clases que este relacionadas con otra clase, la precisión del modelo es cercana al 0.03 cuando el umbral de similaridad se encuentra en 0.5 y decrece con el aumento del umbral, hasta llegar a un valor cercano a 0.014 cuando el umbral se encuentra en 0.95. Así mismo, se puede apreciar que a diferencia de los resultados obtenidos con las clases, el *Recall* presenta valores más bajos que la precisión, iniciando con un valor cercano a 0.22 con el umbral de 0.5 y disminuyéndolo a medida que crece el valor del umbral, acercándose a 0.008 con el umbral más alto.

Estos resultados presentan valores demasiado bajos respecto a los resultados obtenidos con las clases, a causa de que en la estructura de las historias de usuario no se brinda la información suficiente para poder determinar los atributos o características propias de cada una

de las clases, por lo cual, es reducida la cantidad de atributos que pueden ser encontrados directamente por el sistema por medio de patrones gramaticales. Igualmente, los atributos presentes en las clases encontradas manualmente son, en gran parte, inferidos por la experiencia humana mas no están presentes directamente en las historias, pues si la historia de usuario menciona "... ubicaciones cercanas a mi ...", el humano puede deducir que se requiere tener la ubicación o posición del usuario como uno de sus atributos, sin embargo, el sistema no cuenta con la capacidad de realizar estas deducciones y extrae los atributos que pueda encontrar directamente en el texto analizado a través de los patrones, afectando negativamente los resultados obtenidos y con esto, las métricas de evaluación.

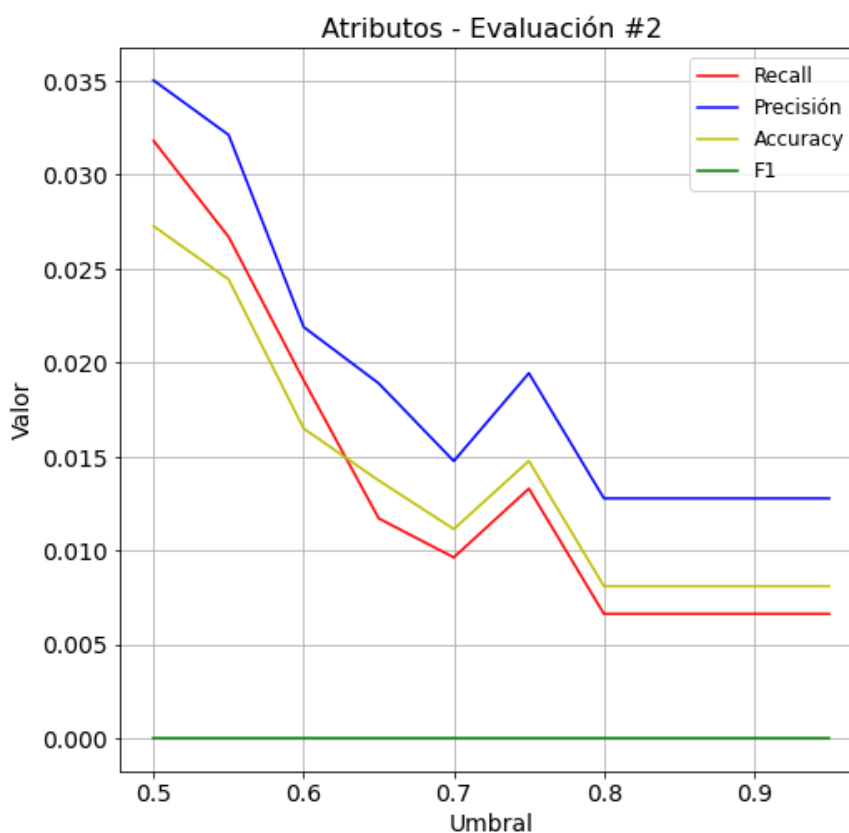


Figura 4-9.: Gráficas de precisión, *Recall*, *Accuracy* y F1 correspondientes a la evaluación #2 de los atributos [Autor].

En la figura 4-9, podemos ver los resultados al tener en cuenta los atributos de aquellas clases que no solo estén relacionadas con otras, sino que cuenten con atributos o métodos,

reduciendo aún más la cantidad de clases evaluadas y con esto, la cantidad de atributos a evaluar. Al realizar estos cambios en la evaluación, se puede apreciar una ligera mejora en los resultados en los umbrales de similaridad más bajos, pues la precisión inicia en 0.035 en el umbral más bajo y decrece hasta acercarse al 0.014 en el umbral más alto. Sin embargo, se mantiene superior al *Recall*, pues este inicia cerca de 0.032 y decrece a medida que se aumenta el umbral de similaridad hasta llegar a un valor cercano a 0.006 al llegar al umbral 0.95 .

Aquí podemos observar que al reducir la cantidad de clases, se presenta una mejora en los resultados a causa de que disminuye la cantidad de clases a las cuales no se les detectaron atributos y con esto, se reduce el ruido generado por la comparación entre atributos de las clases detectadas manualmente y clases detectadas por el sistema que sean similares a estas pero no cuenten con atributos, pues esto conlleva a que todos los atributos de las clases encontradas manualmente sean determinados como *False Negative*, afectando gravemente los resultados. Otro aspecto por mencionar es la diferencia entre la precisión y el *Recall*, pues al evaluar las clases, esta métrica se mantenía superior a la precisión del modelo. Sin embargo, este caso la precisión se mantiene sobre el *Recall* en todos los umbrales, indicando que la probabilidad o desempeño del modelo para detectar correctamente los atributos de las clases es superior a la probabilidad de encontrar los atributos extraídos manualmente dentro de los atributos evaluados, entonces, podemos inferir que el desempeño se mantiene se ve beneficiado por la exclusión realizada en las clases. Aun así, los valores se mantienen bastante bajos en todas las métricas debido a la ausencia de información que permita establecer criterios más claros para la detección de atributos y la incapacidad del sistema de poder deducir atributos que no se encuentren directamente mencionados en las historias de usuario del sistema.

4.2.3. Métodos

En un proceso muy similar al ejecutado para la evaluación de los atributos, los métodos son evaluados entre las clases similares encontradas en la evaluación de clases y tal como se obtuvieron del sistema o manualmente, como se puede observar en la figura 4-10.

No obstante, se realizan algunas validaciones adicionales además de la similaridad entre los métodos obtenidos:

- Se valida que el método contenga un verbo, dado que representa un acción.
- Se pueden presentar casos en los cuales los métodos no sean similares según el umbral, pero aún así, representen la misma acción y los términos del método manual se encuentren en el método obtenido por el sistema, por lo cual no son similares en escritura pero si en contexto de la acción. Ej: Si el método manual es *loguear usuario* y el método obtenido por el sistema es *loguear usuario administrador*.

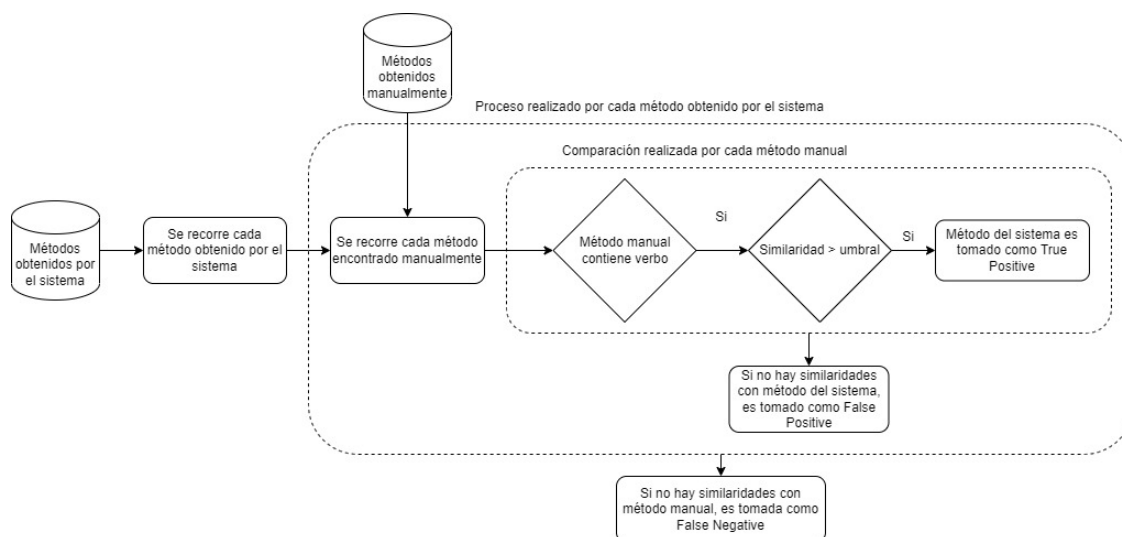


Figura 4-10.: Flujo de comparación de resultados obtenidos en extracción de métodos de la clase [Autor]

Si al comparar los métodos del sistema con los métodos manuales, superan el umbral de similaridad y/o aprueban estas validaciones, se toma como el método obtenido por el sistema es un **True Positive**. Si no se encuentran similitudes para el método del sistema, es tomado como un **False Positive**. Si no se encuentran similitudes para el método obtenido manualmente, se toma como **False Negative**.

En este caso también se realizaron dos evaluaciones distintas de acuerdo a la información de las clases a las cuales pertenecen los métodos y la similaridad entre ellas:

- **Evaluación #1:** En esta evaluación se toman los métodos de las clases encontradas por el sistema, siempre y cuando estén relacionadas por otra clase (Figura 4-11).
- **Evaluación #2:** En esta evaluación se toman los métodos de las clases detectadas por el sistema que tengan un atributo o método y estén relacionadas a otra clase (Figura 4-12).

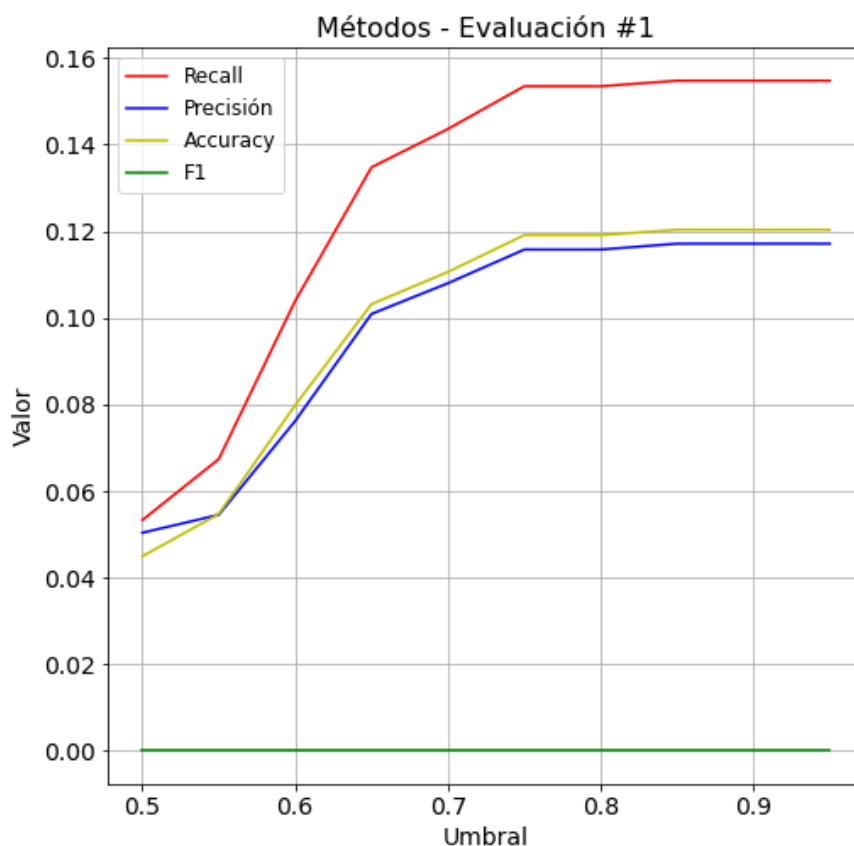


Figura 4-11.: Gráficas de precisión, *Recall*, *Accuracy* y F1 correspondientes a la evaluación #1 de los métodos [Autor].

En la figura 4-11, se puede observar un comportamiento distinto a las demás evaluaciones anteriores, pues en el umbral más bajo, el *Recall* inicia con un valor cercano al 0.055 y aumenta constantemente junto con el umbral de similaridad hasta estabilizarse cerca de 0.16 cuando el umbral supera el 0.75. Así mismo, la precisión tiene un valor aproximado a 0.05 cuando el umbral de similaridad está en 0.5 y aumenta gradualmente hasta estabilizarse cerca de 0.12 desde el umbral en 0.75.

El modelo presenta un bajo desempeño al momento de extraer los métodos de las clases a causa de que se ve afectada por los resultados obtenidos en la evaluación #2 de extracción de clases, pues al no encontrar una clase similar para la clase que contiene los métodos o que las clases sean similares en cuanto a su nombre más no en cuanto a su contenido, ocasionara que los métodos extraídos por el sistema no encuentren sus correspondientes similitudes al momento de realizar la evaluación. Así mismo, también se ve afectado por la interpretación dada por el usuario a los posibles métodos presentes en las historias de usuario, pues mientras el modelo extrae los métodos tal cual como se encuentran en el texto, el humano puede ajustar el nombre de los mismos para adaptarse al contexto de la historia o su interpretación del posible método. Por ejemplo:

Si la historia menciona «... quiero ver las versiones disponibles de un objeto ...», el sistema extrae el método *ver_version_objeto*, mientras el método identificado por un humano es *ver_versiones_disponibles*, los cuales hacen referencia a la misma acción, sin embargo, no son similares de acuerdo a sus representaciones vectoriales. Otro ejemplo se presenta cuando la historia menciona «... quiero otorgar privilegios de embargo a otros administradores de repositorios ...», pues el sistema extrae el método *otorgar_privilegio_embargo_administrador_repositorio*, mientras el método extraído manualmente es *otorgar_permiso_admin*, ocasionando que el modelo no los detecte como métodos similares a pesar de hacer referencia a la misma acción.

Sin embargo, también es apreciable que a medida que aumenta el umbral de similaridad para la evaluación, se presenta una creciente mejora en las métricas debido a que este aumento en el umbral conlleva a que los métodos comparados sean cada vez más similares, obteniendo una mayor precisión al momento de determinar los métodos correctos del sistema y disminuir los métodos detectados como similares de manera errónea, reduciendo el posible ruido en los resultados y mejorando el desempeño del modelo.

En la figura 4-12 podemos ver que hay una leve mejora al tener en cuenta los métodos de las clases que no están vacías (tienen atributos y/o métodos), pues el *Recall* tiene un valor de 0.6 cuando el umbral de similaridad es 0.5 y aumenta a un gran ritmo hasta estabilizarse cerca del umbral 0.75, donde disminuye su velocidad de aumento y continua aumentando a menor ritmo hasta aproximarse a 0.17 cuando el umbral es 0.95 . Por otro lado, la precisión

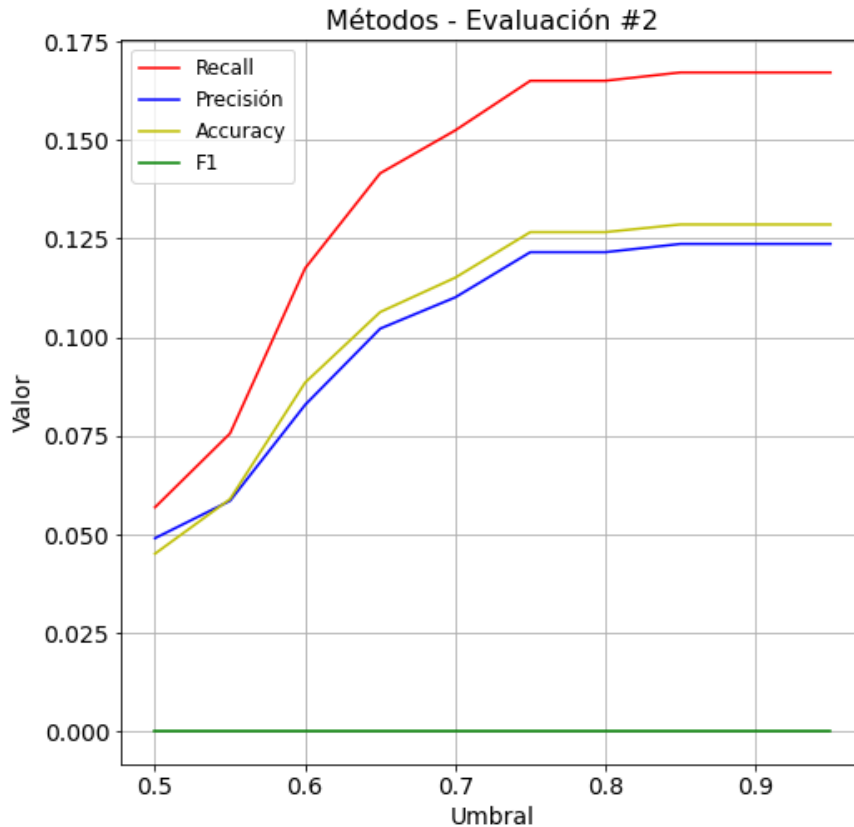


Figura 4-12.: Gráficas de precisión, *Recall*, *Accuracy* y F1 correspondientes a la evaluación #2 de los métodos [Autor].

es de 0.05 cuando el umbral es de 0.5 y aumenta gradualmente hasta llegar a 0.125 en el umbral 0.95 .

Con base en esto, podemos observar que hay una leve mejora en los resultados obtenidos durante la evaluación, pues al tener en cuenta solo las clases que tengan métodos o atributos, disminuye la cantidad de clases que no tienen métodos detectados por el sistema y con esto, disminuyen los *False Negative* encontrados durante la evaluación #1, conllevando a una mejora en la precisión del modelo y del desempeño general de este para la extracción de métodos. No obstante, persiste la tendencia a valores bajos (A pesar de su incremento al aumentar el umbral de similaridad) debido al efecto que tiene la posible extracción errónea de clases sobre estos y las diferencias de interpretación por parte del usuario al momento de extraer los métodos manualmente y la incapacidad del modelo de poder realizar estas

mismas interpretaciones que no están presentes directamente en el texto.

4.2.4. Relaciones

Para la evaluación de las relaciones de las clases extraídas por el sistema, se hace un procedimiento similar a los realizados anteriormente (Figura 4-13). No obstante, en este caso no se tiene en cuenta un umbral de similaridad entre las relaciones, sino que se realiza una comparación de las clases relacionadas por las mismas.

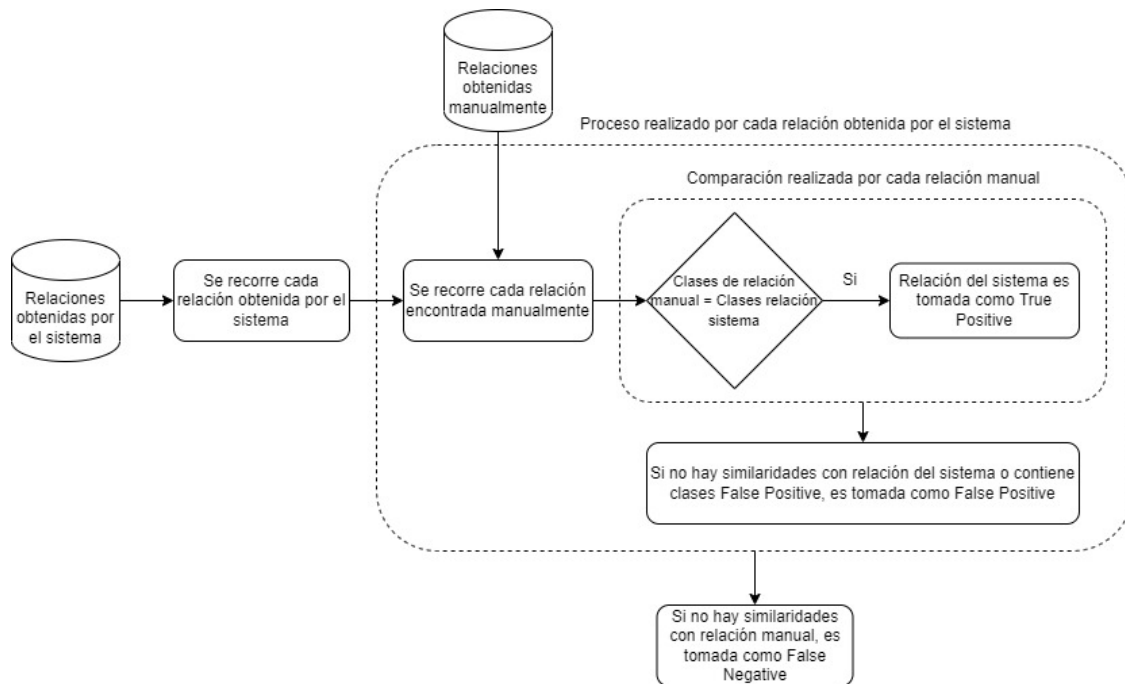


Figura 4-13.: Flujo de comparación de resultados obtenidos en extracción de relaciones entre las clases [Autor]

En esencia, el procedimiento tiene en cuenta las clases similares detectadas en la extracción de clases y analiza si las dos clases de una relación cuentan con clases similares. De ser así, si dichas clases similares también están relacionadas, se determina que la relación encontrada por el sistema es un *True Positive*.

A continuación, se detallan los posibles casos que se podrían presentar:

- Si las clases de una relación encontrada por el sistema, tienen clases similares encontradas manualmente y dichas clases están relacionadas, se considera *True Positive*.

- Si las clases de una relación encontrada por el sistema tienen clases similares encontradas manualmente, pero dichas clases no están relacionadas, se considera *False Positive*.
- Si solo una de las clases de una relación encontrada por el sistema tiene clase similar encontrada manualmente, se considera *False Positive*.
- Si ninguna de las clases de una relación encontrada manualmente tiene clases similares encontradas por el sistema, se considera *False Negative*.
- Si las clases de una relación encontrada manualmente tienen clases similares encontradas por el sistema pero dichas clases no están relacionadas se considera *False Negative*.

Con base en estos casos y dado que las relaciones encontradas se ven influenciadas por la extracción de clases, se realizaron dos evaluaciones distintas para las relaciones, de acuerdo a las clases obtenidas al evaluarlas con cada umbral:

- **Evaluación #1:** En esta evaluación se toman todas las relaciones de las clases encontradas por el sistema (Figura 4-14).
- **Evaluación #2:** En esta evaluación se toman las relaciones de las clases detectadas por el sistema que tengan un atributo o método y relacionadas con otras clases con la misma condición (Figura 4-15).

En la figura 4-14, podemos observar que la precisión es de 0.075 cuando el umbral de similaridad es de 0.5 y tiende a aumentar hasta acercarse a 0.12 en el umbral 0.7 y disminuye hasta estabilizarse cerca de 0.09 cuando se aproxima al umbral 0.9. Por su parte, el *Recall* inicia con un valor cercano a 0.23 en el umbral más bajo, creciendo rápidamente hasta 0.31 cuando el umbral es 0.6, donde empieza a disminuir hasta estabilizarse cerca de 0.24 al aproximarse al umbral más alto. A causa de esta variación en las métricas, el F1 es de 0.1 al evaluar con el umbral más bajo y aumenta levemente hasta llegar a 0.15 en el umbral de similaridad 0.7, donde comienza a decrecer gradualmente hasta acercarse a 0.12 en el umbral 0.95.

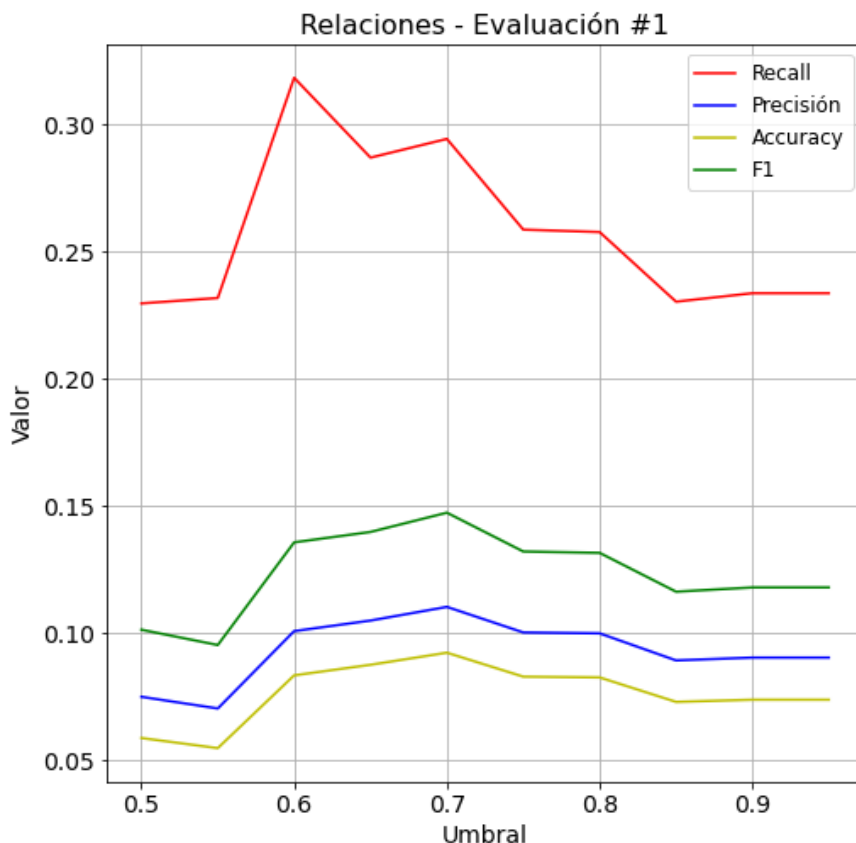


Figura 4-14.: Gráficas de precisión, *Recall*, *Accuracy* y F1 correspondientes a la evaluación # 1 de las relaciones entre clases [Autor].

Este comportamiento en los resultados de las relaciones es en gran parte, influenciado por los resultados obtenidos en la extracción de clases dado que cuando estas son evaluadas con los umbrales intermedios (0.6 - 0.7), hay mayores probabilidades de mantener las relaciones correctas, pues es mayor la cantidad de similitudes encontradas entre las clases obtenidas manualmente y las clases obtenidas por el modelo, al mismo tiempo, las clases extraídas erróneamente y detectadas como similares disminuye para este punto, beneficiando al desempeño del modelo para extraer las relaciones correctas dado que la probabilidad de que las similitudes para las clases que las conforman sean encontradas. Sin embargo, al continuar aumentando el umbral de similitud, disminuye la cantidad de clases manuales similares encontradas para las clases del modelo o viceversa, por lo cual, las relaciones se ven afectadas al no encontrar su correspondiente relación entre las obtenidas manualmente, conllevando a

que el desempeño del modelo se vea afectado negativamente al no lograr extraer las relaciones adecuadamente en los umbrales más altos, lo cual se ve reflejado en el descenso de las métricas hasta llegar a un valor muy cercano al valor inicial de la evaluación.

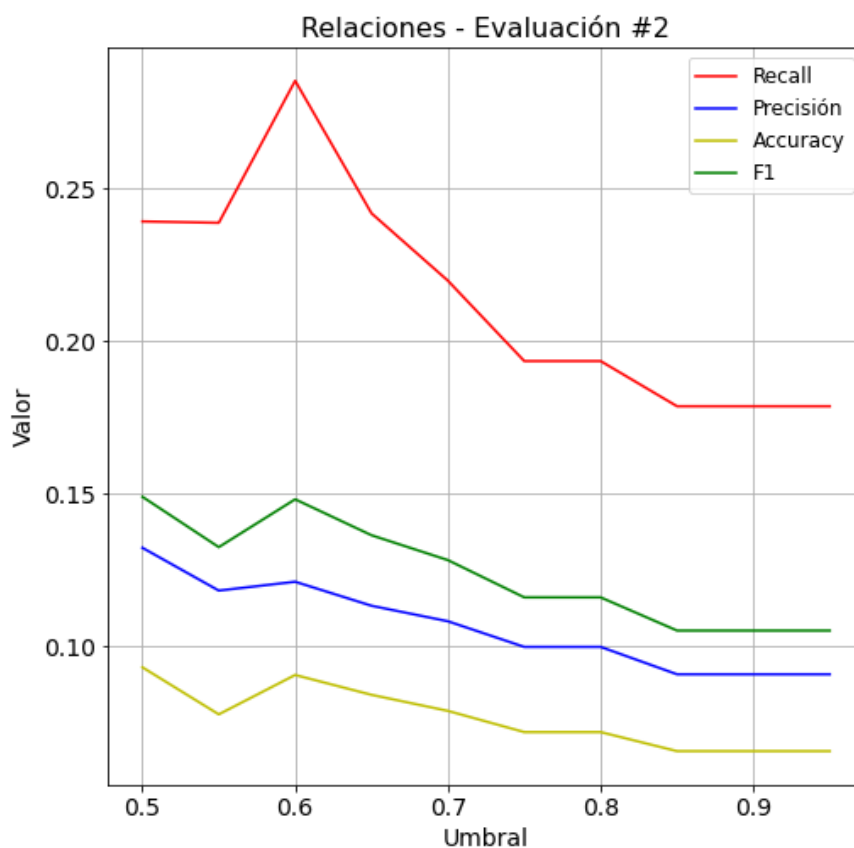


Figura 4-15.: Gráficas de precisión, *Recall*, *Accuracy* y F1 correspondientes a la evaluación # 2 de las relaciones entre clases [Autor].

En la figura 4-15 se puede percibir un cambio en el comportamiento de los resultados, pues el *Recall* tiene un valor aproximado a 0.24 en el umbral más bajo y alcanza su valor más alto en el umbral 0.6 cuando llega a 0.28, decrece hasta llegar a 0.18 en el umbral 0.95. Por otro lado, la precisión es de 0.13 en el umbral más bajo y decrece a un ritmo bajo a medida que aumenta el umbral de similaridad, hasta acercarse a 0.09 en el valor máximo del umbral. Debido a estos comportamientos en las métricas, el F1 inicia con el valor igual a 0.15 y decrece paulatinamente hasta acercarse a 0.1 en los umbrales más altos.

Esta variación en el comportamiento de los resultados se presenta por la reducción de cla-

ses extraídas, pues al tener en cuenta aquellas que no sean vacías (Tienen atributos y/o métodos), se retiran múltiples clases que en la extracción manual hacen parte de las relaciones detectadas, pero que al no lograr extraer adecuadamente su información, el sistema las remueve, provocando que el desempeño del modelo se vea afectado drásticamente al no poseer la capacidad de encontrar relaciones similares o correspondientes a gran parte de las relaciones extraídas manualmente. Adicionalmente, al aumentar el umbral de similaridad, las clases similares encontradas para las clase obtenidas manualmente disminuyen y con esto, la capacidad del sistema para extraer las relaciones que correspondan a dichas clases similares, afectando su precisión. Cabe mencionar que dado que las relaciones se ven directamente afectadas por la extracción de las clases, el desempeño del modelo presenta valores bajos al no poseer la capacidad de filtrar adecuadamente las clases correctas y con esto, evitar la generación o extracción de múltiples relaciones que no fueron detectadas manualmente pero que unen a dichas clases extraídas erróneamente.

Según los resultados de la extracción de las clases y sus componentes, podemos inferir que la implementación de metodologías de extracción que tengan en cuenta la información completa de las clases brinda un mejor desempeño, dado que se pueden estructurar mecanismos de filtrado más sólidos que permitan remover los términos que causan ruido o no brindan información relevante para los resultados. Sin embargo, el modelo requiere de una mayor capacidad de determinación de componentes basado en el contexto del texto, pues al no lograr extraer esta información directamente del texto se están perdiendo datos de gran importancia para la solidez de los resultados, conllevando a que los mismos mecanismos de filtrado implementados remuevan términos que pueden brindar información de calidad para los resultados.

4.3. Extracción de actores y casos de uso

4.3.1. Actores

Para evaluar la similitud de los actores encontrados por el sistema, frente a los casos encontrados manualmente se utiliza nuevamente el umbral de similitud entre estos.

Si dos actores son comparados y su similitud supera el umbral, el actor encontrado por el sistema sera considerado como **True Positive**. Sin embargo, también se tienen en cuenta los casos en los cuales los términos o sustantivos del actor manual podrían estar contenidos dentro del actor encontrado por el sistema con otros términos. Ej: *Usuario_administrador* está contenido dentro de *Usuario_administrador_sistema*.

El flujo de evaluación de los actores se puede observar en la figura 4-16.

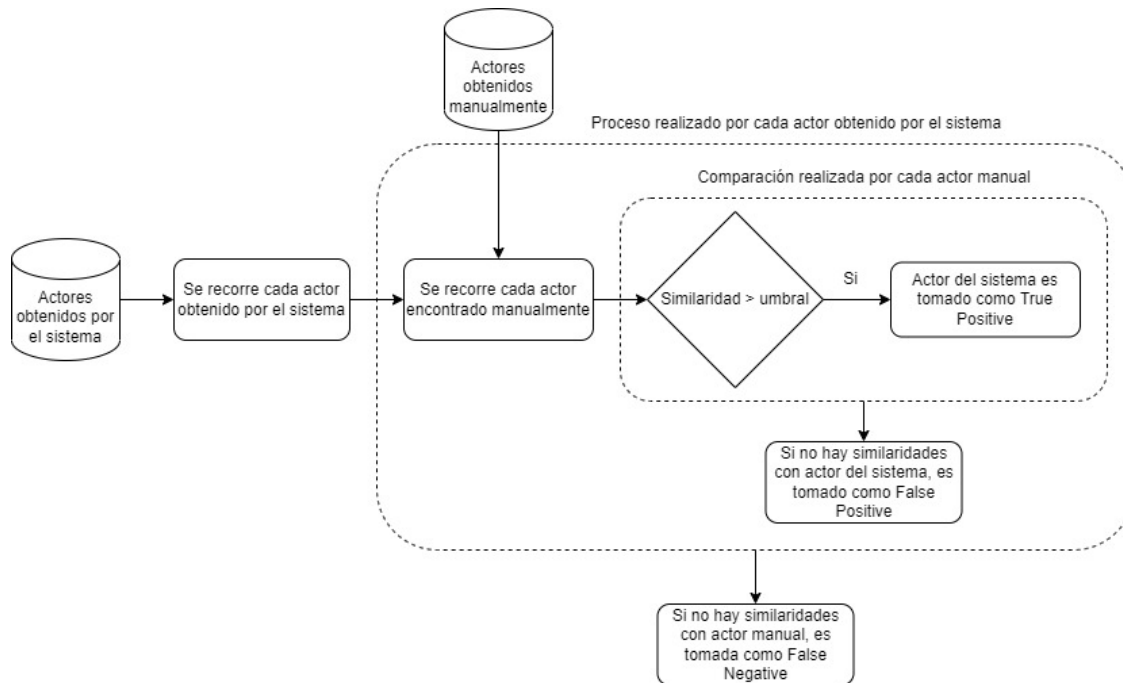


Figura 4-16.: Flujo de comparación de resultados obtenidos en extracción de actores [Autor]

Si un actor del sistema al ser comparado con un actor manual, supera el umbral de similitud o contiene los términos de dicho actor manual, se considera **True Positive**. Si el actor del sistema no cumple estas condiciones con ningún actor manual, es considerado **False Positive** y si el actor manual no cumple dichas condiciones con ningún actor del sistema,

es considerado *False Negative*.

Los resultados de este proceso de evaluación pueden ser observados en las figura 4-17.

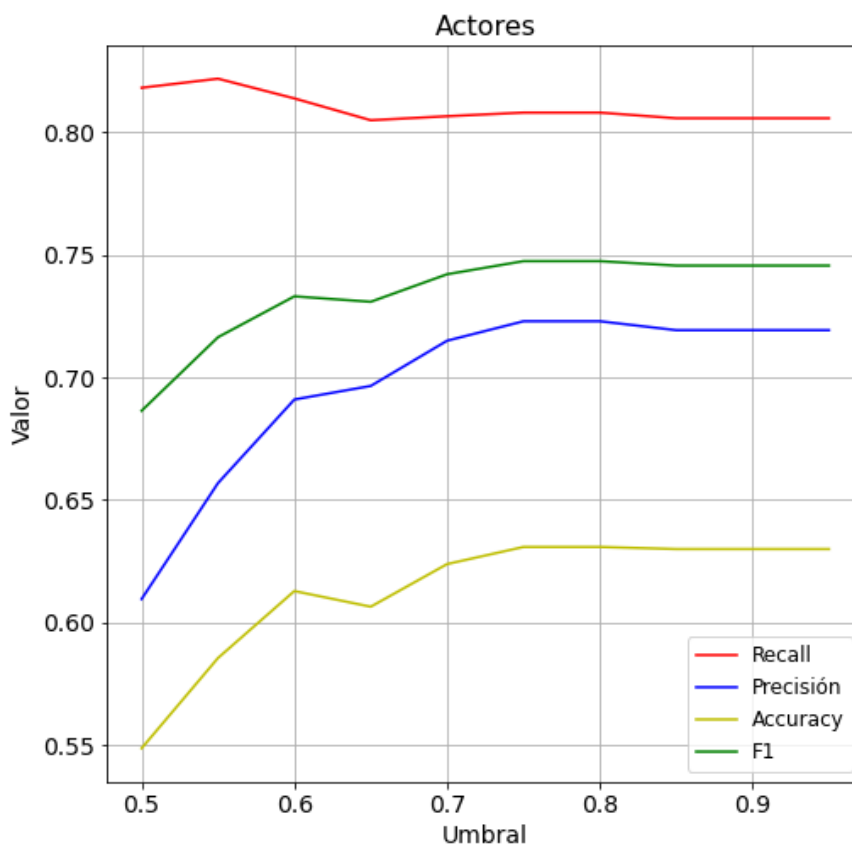


Figura 4-17.: Gráficas de precisión, *Recall*, *Accuracy* y F1 correspondientes a los actores obtenidos por el sistema [Autor].

Como podemos observar en las gráficas de la figura 4-17, la precisión del modelo para predecir los actores es relativamente alta, iniciando con un valor aproximado de 0.61 en el menor umbral y aumentando gradualmente hasta acercarse a 0.72 al llegar al umbral más alto. Por otro lado, el *Recall* inicia con un valor cercano a 0.82 en el umbral de similitud 0.5 y a medida que aumenta el umbral, desciende levemente hasta acercarse a 0.8 en el umbral 0.95 .

A diferencia de la extracción de clases, el modelo presenta un mejor desempeño al momento de extraer los actores presentes en los conjuntos de historias de usuario, dado que la estructura de las mismas permite un reconocimiento más práctico de los mismos al ser el

primer sustantivo presente en estas. Esto lo podemos notar con el constante crecimiento de las métricas al aumentar el umbral, pues al realizar una comparación más estricta entre los actores manuales y detectados por el sistema, se retiran los actores detectados erróneamente, disminuyendo la sobre-generación de términos y dejando en su lugar los actores reales o que también fueron detectados manualmente de acuerdo al *Recall*, en todos los umbrales hay una alta probabilidad de encontrarlos. No obstante, se ven afectados por posibles casos donde la aplicación de patrones no logra detectar los posibles casos de manera adecuada, pues a causa de que al aplicar los patrones gramaticales, se requiere recorrer un par de veces la misma historia para determinar primero, si se cumple alguno de los patrones y segundo, si las partes de la historia que cumplen con un patrón en conjunto, cumplen con un patrón mayor. Por ejemplo:

El actor «administrador/NOUN de/ADP el/DET sistema/NOUN» cumple con el patrón <NOUN><EXC_ACT1><NOUN>. Sin embargo, este mismo, cumple con el patrón ACT_1, por lo cual se requiere reanalizar la historia para poder establecer como el actor cumple con este patrón.

A causa de esto, hay algunos casos en los cuales el sistema no logra determinar adecuadamente los patrones que se cumplen, dado que no se recorre la historia de usuario la cantidad de veces necesaria para poder realizar esta detección.

4.3.2. Casos de uso

Para la comparación de los casos de uso, se tienen en cuenta los casos de uso de los actores que hayan sido encontrados como similares en el proceso anterior. Es decir, se realiza la comparación entre casos de aquellos actores que sean similares, por medio del procedimiento representado en la figura 4-18.

Para determinar si dos casos de uso son similares, se tiene en cuenta el umbral de similaridad utilizado con los demás componentes. Sin embargo, también se tiene en cuenta si los términos del caso de uso obtenido manualmente se encuentran dentro del caso de uso obtenido por el sistema y que los casos de uso manuales cumplan con el formato Verbo + sustantivo.

Si un caso de uso supera el umbral de similaridad o contiene los términos del caso de uso

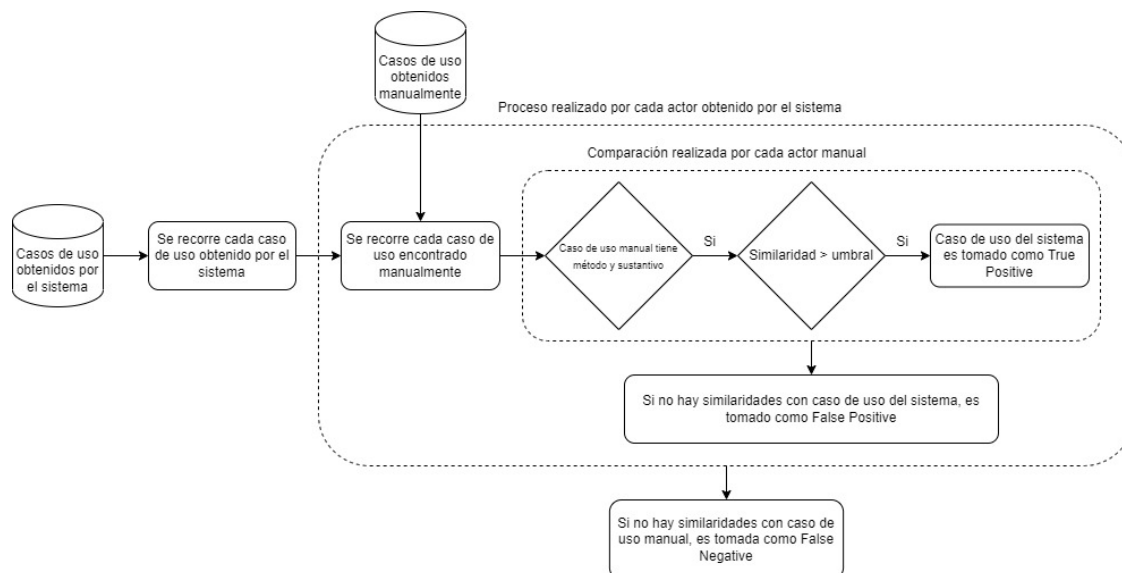


Figura 4-18.: Flujo de comparación de resultados obtenidos en extracción de casos de uso de los actores [Autor]

encontrado manualmente, se determina que el caso de uso es **True Positive**. Si el caso de uso del sistema no cumple con estas condiciones con ninguna clase manual, se determina que es un **False Positive**. Si no se encuentran similitudes para una clase manual, se determina que es **False Negative**.

Los resultados de precisión, *Recall* y F1 por cada umbral, son representados en la figura 4-19.

En la figura 4-19, podemos apreciar como la precisión del modelo tiene una tendencia creciente a medida que aumenta el umbral de similitud, pues su valor más bajo se presenta en el umbral 0.5 con un valor cercano a 0.25 y aumenta gradualmente hasta acercarse a 0.33 al llegar al umbral 0.95. Con un comportamiento similar, el *Recall* tiene un valor cercano a 0.25 en el umbral más bajo y crece gradualmente hasta acercarse a 0.35 con el umbral más alto.

Para la extracción de caso de uso el modelo presenta un desempeño más bajo, pues a pesar de que los actores son extraídos adecuadamente en su gran mayoría, los casos de uso de estos pueden presentar diferencias con respecto a los casos obtenidos manualmente dado que el usuario o humano puede analizar la historia e inferir las acciones de cada actor, aunque estas

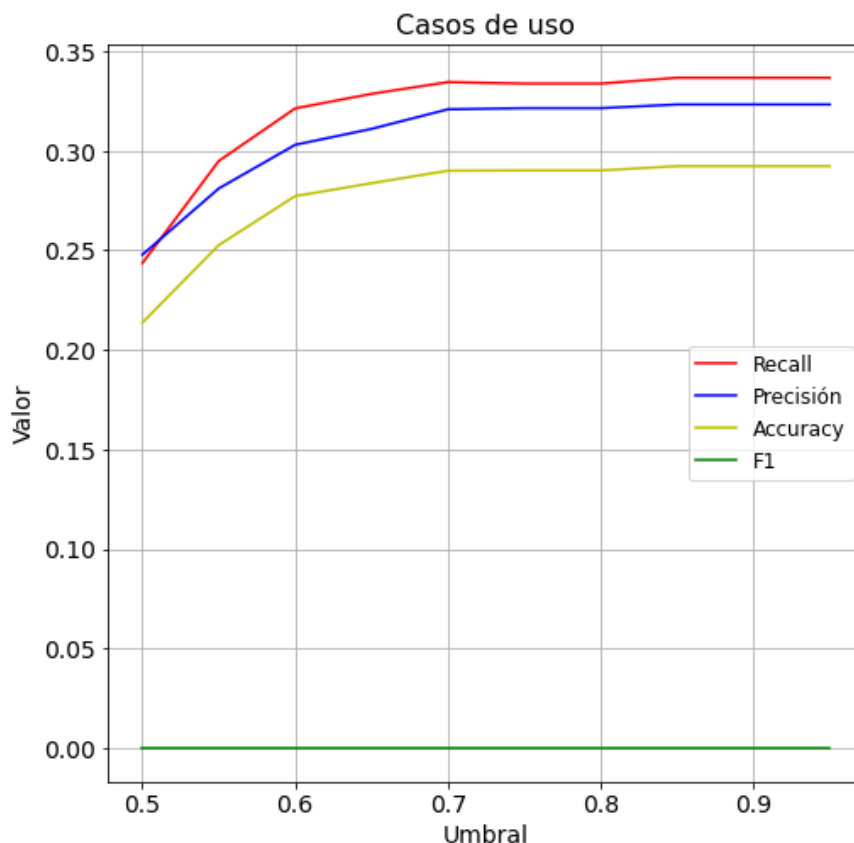


Figura 4-19.: Gráficas de precisión, *Recall*, *Accuracy* y F1 correspondientes a los casos de uso obtenidos por el sistema [Autor].

no estén presentes en el texto o en la estructura analizada de la historia de usuario.

Un ejemplo de esto se presenta en la siguiente historia de usuario: «Como investigador, quiero poder ver una ayuda de búsqueda para una colección / recurso en particular», donde el sistema detecta al actor *Investigador* y el caso de uso *ver_ayuda_búsqueda_coleccion*, sin embargo, su correspondiente caso de uso obtenido manualmente sería *realizar_búsqueda* pero dado que no se encuentra directamente en la historia de usuario, el sistema no logra extraerlo ni encontrar una similitud para este.

Sin embargo, a medida que aumenta el umbral de similitud, las métricas de extracción de casos de uso mejoran gradualmente, dado que al determinar la similitud de manera más estricta, es menor la cantidad de casos de uso encontrados similares erróneamente, por lo cual se reduce la posibilidad de encontrar múltiples términos o frases como similares a los

casos de uso y con esto, aumenta la precisión del modelo al no verse afectado por posibles excesos en la cantidad de términos extraídos.

5. Prototipo de software

El software construido para la extracción de componentes y generación de diagramas UML fue desarrollado en lenguaje *Python 3.10*, haciendo uso principalmente de las librerías:

- **NLTK**: Herramienta para PLN utilizada para generar los analizadores sintácticos para cada uno de los conjuntos de patrones utilizados y aplicarlos sobre las historias de usuario analizadas.
- **Stanza**: Herramienta para PLN utilizada para extraer las principales características de las historias de usuario y sus términos, por medio de etiquetas *Part-of-Speech* y extracción de características.
- **Sklearn**: Utilizada para determinar y extraer las entidades más frecuentes de acuerdo a su aparición en el texto.
- **Spacy**: Herramienta para PLN utilizada para determinar la similaridad de los términos al momento de realizar la evaluación de resultados.

El software desarrollado se ejecuta por medio de consola, haciendo un llamado al archivo principal (*generator.py*) para iniciar el proceso de análisis de las historias de usuario y obtener los diagramas objetivo. La ejecución total del software se tarda entre 1 o 2 minutos desde el momento en que se realiza el llamado al generador, hasta el momento en que se obtienen las imágenes de los respectivos diagramas. En la figura **5-1** se evidencia el proceso ejecutado por el software y la interacción con los diferentes extractores en cada etapa del proceso.

A continuación se explica a mayor detalle el funcionamiento de cada una de las partes expresadas en el diagrama:

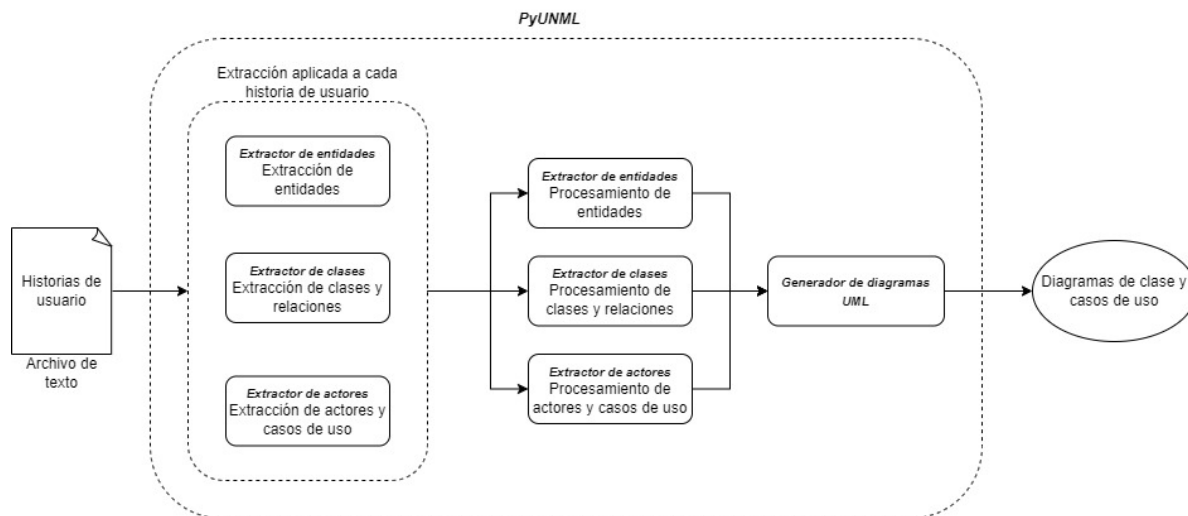


Figura 5-1.: Diagrama esquemático de software construido para la generación de diagramas UML a partir de historias de usuario [Autor].

5.1. PyUNML

El prototipo es ejecutado por el usuario por medio del comando:

```
python generator.py
```

Luego, se le pregunta al usuario el nombre o ruta del archivo de texto con las historias de usuario que serán analizadas para generar los respectivos diagramas. Una vez que el usuario ingresa dicha información, el software abre el respectivo archivo y recorre una a una las historias de usuario contenidas en el mismo.

Para cada historia de usuario se realiza el siguiente proceso:

- La historia es analizada por medio de *Stanza* para extraer sus características y etiquetas *Part-of-Speech*.
- La historia etiquetada es enviada al extractor de entidades, el cual retorna las entidades presentes en la historia de usuario, luego son almacenadas para ser tenidas en cuenta en cada historia y evitar duplicación de entidades.

- La historia etiquetada es enviada al extractor de clases, este retorna las clases encontradas en la historia de usuario, que son almacenadas junto con sus respectivas relaciones, para ser tenidas en cuenta al analizar cada historia y de este modo, estructurar las clases y sus relaciones a medida que se realiza el análisis.
- La historia etiquetada es enviada al extractor de actores y casos de uso, el cual retorna el actor de la historia de usuario y su respectivo caso de uso, los cuales son almacenados para evitar tener casos de uso o actores duplicados y de este modo, obtener cada uno de los casos de uso para cada actor.

Acto seguido, al completar el análisis para todas las historias de usuario, la información recolectada es enviada a cada uno de los respectivos extractores (Extractor de entidades, extractor de clases y extractor de actores) para ser depurada y reorganizada de acuerdo a las reglas y validaciones establecidas para cada tipo de componente. La información organizada es retornada por cada extractor para ser enviada al generador de diagramas UML, que toma dicha información y genera los respectivos archivos de texto e imágenes de los diagramas UML de acuerdo a los datos suministrados.

5.2. Extractor de entidades

Este extractor (`EntitiesExtractor.py`) primero recibe las entidades encontradas para las historias de usuario anteriores y la historia de usuario actual con sus respectivas etiquetas PoS, la cual es procesada por un analizador sintáctico con los patrones construidos para la extracción de entidades y evalúa que las entidades obtenidas cumplan con las validaciones establecidas para extracción de entidades (Ver sección 3.2). De ser así, agrega las entidades encontradas para la historia actual al conjunto de entidades recibido y lo retorna.

Una vez que todas las historias de usuario han sido analizadas, el extractor recibe el conjunto de todas las entidades encontradas en total durante el análisis y por medio de *Sklearn* determina la frecuencia TF-IDF de las entidades encontradas y las retorna en un conjunto organizado de la más frecuente a la menos frecuente según los resultados de la librería.

5.3. Extractor de clases

En el extractor de clases (`ClassExtractor.py`) se recibe primero la historia de usuario con sus respectivas etiquetas PoS, a la cual se le realiza el respectivo análisis con los patrones contruidos para la extracción de clases por medio de un analizador sintáctico. De acuerdo a los resultados del analizador, se realizan las validaciones respectivas para determinar si todos los términos extraídos pueden ser considerados clases (Ver sección 3.3) y de ser así, retorna el conjunto de clases encontradas en la historia para ser agregadas al conjunto de clases encontradas en historias anteriores. Acto seguido, recibe también el conjunto de relaciones entre clases que se han encontrado en las historias de usuario que ya han sido analizadas. El software realiza el procesamiento de la historia de usuario con los patrones contruidos para la extracción de relaciones y clasifica las posibles relaciones obtenidas de acuerdo a su forma y los tipos de términos que la componen. Una vez que se han obtenido las posibles relaciones entre clases presentes en la historia de usuario, son agregadas al conjunto de relaciones encontrada anteriormente y lo retorna.

Una vez que se ha completado el análisis para todas las historias de usuario, el extractor recibe el conjunto total de clases encontradas y el conjunto total de relaciones obtenidas de las historias, para ser procesadas con el objetivo de extraer los atributos y métodos de las clases de acuerdo a sus relaciones con otras clases y depurar las relaciones finales de las clases recibidas. Una vez que se ha realizado este proceso, el software reorganiza las clases con su respectiva información y las relaciones finales encontradas entre estas para retornar dicho conjunto al proceso principal del prototipo.

5.4. Extractor de actores y casos de uso

Este extractor (`CaseUseExtractor.py`) recibe la historia de usuario con sus respectivas etiquetas PoS, el listado de actores y los casos de uso encontrados en las historias de usuario anteriores. El extractor toma la historia de usuario etiquetada y extrae el actor de la misma aplicando los patrones de extracción de actores sobre la primera parte de la misma (*Como*

[*Actor*],...) por medio de *NLTK*. Una vez que se obtiene el actor de la historia de usuario, se procede a extraer el caso de uso de este por medio de la aplicación de los patrones construidos para los casos de uso y la validación de los mismos de acuerdo a las reglas establecidas (Ver sección 3.4).

Al terminar de analizar todas las historias de usuario, el extractor recibe todos los actores encontrados y los posibles casos de uso, por lo cual se realiza la depuración de los mismos, con el objetivo de evitar casos de uso repetidos y poder organizarlos de acuerdo al actor al cual pertenecen. Al final del proceso, cada actor contiene un conjunto de casos de uso y todos los actores se encuentran almacenados en un mismo conjunto, el cual es retornado por el extractor al proceso principal del prototipo.

5.5. Generador de diagramas UML

Este generador (*DiagramFileGenerator.py*) recibe todos los conjuntos retornados por los extractores y los procesa con el objetivo de construir un archivo de texto con la estructura necesaria para la generación de los correspondientes diagramas UML. Los formatos requeridos en los archivos de texto será diferente de acuerdo al tipo de diagrama que se desee construir, por lo cual el generador toma cada uno de las salidas de los respectivos extractores y las organiza de modo que cumplan con lo exigido por la librería *PlantUML* para la generación de diagramas UML. Los formatos especificados para cada tipo de diagramas pueden ser observados en la sección 3.5.

Una vez que los archivos de texto respectivos han sido generados, son procesados por medio de *PlantUML*, la cual toma cada archivo de texto, analiza su estructura y retorna la imagen del diagrama UML respectivo. De este modo, se obtienen los diagramas requeridos por el software y en el caso de las entidades, se obtiene un archivo de texto con un listado de las entidades ordenadas de acuerdo a su frecuencia.

6. Conclusiones y trabajo futuro

6.1. Conclusiones

El análisis de requerimientos es un tema que día tras día toma mayor trascendencia en el mundo de la ingeniería de software, siendo la etapa en la que se toman decisiones primordiales para la continuación del proyecto y como se desarrollarán las siguientes etapas del mismo. No obstante, a causa de la globalización y la importancia que ha adquirido el idioma inglés durante las últimas décadas, gran parte de los esfuerzos dedicados al avance de las tecnologías y crecimiento de las diferentes áreas de desarrollo se enfocan en este idioma o se centralizan en los lugares de habla inglesa, dejando rezagados aquellos países o oportunidades que puedan presentarse en un idioma diferente a este, como es el caso del lenguaje español y el análisis de requerimientos. Durante la última década se han desarrollado múltiples trabajos en aras de automatizar y brindar mejores soluciones para el proceso de análisis de requerimientos, sin embargo, estos esfuerzos se ven dedicados al idioma inglés, construyendo modelos que pueden analizar la gramática de este idioma o las ocurrencias en su estructura y de este modo, extraer las principales características del software. Sin embargo, la ausencia de un enfoque de análisis en idioma español conlleva a que los procesos y desarrollos de software ejecutados en países de habla hispana o cuya base recaída en el idioma español, se vean retrasados o afectados por los posibles riesgos de la falta de automatización del proceso para apoyar el análisis, pues no hay alternativas aplicables que puedan ser de ayuda en esta situación.

Durante el desarrollo de este trabajo se buscó el desarrollo de un modelo computacional que

podiera mitigar dicha situación o ser de ayuda para los proyectos de software en el lenguaje español por medio de un enfoque cercano a los trabajos desarrollados en el idioma español: El uso de patrones gramaticales aplicados a historias de usuario. Para esto, se tomaron como base reglas establecidas para la extracción de componentes UML a partir de historias de usuario en inglés, las cuales fueron adaptadas al español para extraer dichos componentes a partir de historias en este lenguaje. Sin embargo, hay múltiples factores que deben ser tenidos en cuenta al momento de adaptar y desarrollar el modelo, los cuales pueden afectar el desempeño del modelo.

Para iniciar, durante la identificación de entidades presentes en las historias, el modelo tuvo un buen desempeño detectando las entidades obtenidas manualmente, a causa de la frecuencia de las mismas a lo largo del texto y la importancia dada a dicha frecuencia. Sin embargo, no es capaz de definir con claridad cuales son las verdaderas entidades presentes y cuales son relativamente objetos o sustantivos que no brindan información de valor para el proyecto de software, conllevando a que se presente una sobre-generación o excesos de extracción de términos que gramaticalmente son similares a las entidades, mas no representan a estas, por lo cual el modelo disminuye su desempeño respecto a la precisión del mismo por las entidades encontradas erróneamente.

Adicionalmente, para el diseño y reconocimiento de patrones dentro de las historias de usuario en español, se deben tener en cuenta las diferencias gramaticales entre el idioma español y el idioma inglés, pues mientras en este último los verbos tienen 4 conjugaciones distintas en promedio, mientras en el español se pueden llegar a tener 12 o más conjugaciones para un mismo verbo. Así mismo, en el inglés los adjetivos anteceden al sustantivo o los sustantivos son convertidos en posesivos (Genitivo sajón), a diferencia del español, donde los adjetivos suelen presentarse después de los sustantivos y suelen usarse preposiciones como « de » para indicar los posesivos dentro del lenguaje. Estas diferencias gramaticales conllevan a que se deba realizar un análisis más detallado de la estructura de las historias de usuario durante el procesamiento de lenguaje natural a causa de las múltiples formas gramaticales que pueden presentarse para expresar ideas similares y así mismo, conducen a la posible extracción de

términos que poseen una gran similaridad gramatical con los componentes objetivo pero no brindan la información o cumplen con los criterios que se tendrían en cuenta al momento de realizar la extracción manual, ya sea por su poca importancia para el funcionamiento del software o porqué no brindan una utilidad a las funcionalidades que se podrían implementar para el software. Por otro lado, por razones muy similares se podrían estar perdiendo términos que lograrían brindar una mejor estructura al software o relacionar diferentes funcionalidades que serian vitales para el sistema, pues dada a su estructura gramatical puede que no sean detectados adecuadamente o el modelo los descarte al no poder extraer mayor información sobre ellos, debido a su incapacidad de poder inferir algunos componentes o términos a pesar de que no estén propiamente en el texto.

Este comportamiento en el modelo durante la aplicación de los patrones construidos incide directamente en la generación de diagramas UML a partir de los componentes extraídos, pues al detectar algunos componentes erróneamente o excluirlos a pesar de que podrían tener una gran importancia para el software, se generan diagramas que podrían presentar inconsistencias o exceso de información que no deje muy claro al usuario los componentes primordiales del sistema. Sin embargo, el modelo logra extraer parte de los componentes que estructurarán el sistema adecuadamente, lo cual establece un base sobre la que el usuario puede guiarse o apoyarse para determinar que términos son realmente necesarios al diseñar el software con base en la información suministrada o bien, tener una perspectiva distinta del software que le permita realizar un análisis más claro respecto a que componentes brindan la información o solidez requerida para el desarrollo del software.

En cuanto al desempeño del modelo, este presenta mejores resultados al excluir aquellos componentes que sean vacíos o que no brinden suficiente información al software, pues el recall alcanza valores cercanos a 0.65 al extraer las clases y 0.81 al extraer los actores del software, donde también alcanza valores cercanos a 0.72 en la precisión, incluso en la comparación más estricta entre los componentes extraídos por el sistema y los extraídos manualmente. Sin embargo, debido a la incapacidad del modelo para inferir términos o componentes y la similitud gramatical de algunos otros, la extracción de componentes como atributos, métodos y relaciones de las clases se ve afectada negativamente, pues a pesar de que el desempeño del

modelo tiende a mejorar al realizar comparaciones más estrictas de estos, su valor máximo de precisión esta cerca de 0.125 al extraer métodos y 0.14 para la extracción de relaciones, lo cuales no se alejan demasiado del recall obtenido, pues alcanza valores cercanos a 0.27 para las relaciones y 0.17 para la extracción métodos. La extracción de atributos es donde se presenta un menor desempeño, pues mientras su precisión es de 0.014 en los umbrales de similaridad más altos, el recall alcanza valores aun más bajos cercanos a 0.006, indicando un gran déficit del modelo al momento de detectar correctamente los atributos al no contar con la suficiente información para su extracción y no contar con criterios más sólidos para su detección. Por su parte, la extracción de casos de uso presenta mejores resultados, pues su precisión supera el 30 % de probabilidad de ser detectados correctamente, mientras que el recall no se aleja mucho con un 35 % de probabilidad de que los casos detectados manualmente sean detectados por el modelo.

Con base en esta evaluación del desempeño, se hace evidente la necesidad de establecer criterios más solidos para la obtención de información de cada uno de los componentes, criterios que permitan marcar una diferencia más clara entre los términos detectados erróneamente y los términos que puedan ser importantes para la estructuración del software, de modo que se pueda mitigar la exclusión de términos que contengan información primordial para la construcción de funcionalidades y reducir los términos que puedan causar ruido y confusión al usuario. Sin embargo, este modelo tiene la capacidad de establecer una base preliminar del software a construir, siendo un apoyo durante el análisis de requerimientos o diseño de software al guiar a los diseñadores con una nueva perspectiva del sistema y cual podría ser el camino a seguir al momento de establecer la estructura del software y las etapas del proyecto, siendo un primer paso o un acercamiento al avance del diseño de software en español.

6.2. Trabajo futuro

Para futuros trabajos, se podrían construir u obtener múltiples conjuntos de historias de usuario propiamente en español, con el fin de mitigar o evitar los posibles cambios de términos, contexto o interpretación que podrían ser ocasionados al utilizar herramientas de traducción automática y de este modo, mantener la integridad de los requisitos expresados en

las historias de usuario. Un aspecto sobre el cual se podría trabajar o tener un mayor enfoque es la obtención de información más detallada de las entidades y/o componentes para brindar mecanismos de filtrado o extracción más sólidos que permitan definir criterios más claros para la diferenciación de los términos obtenidos erróneamente, respecto a los términos esenciales del software, bien sea por medio de reestructuración de los patrones u uso combinado con modelos de inteligencia artificial enfocados en este análisis. Así mismo, podría replantearse la idea de usar NER para la obtención de las principales entidades, por medio de un modelo preentrenado más completo que permita detectar de manera más precisa las entidades que se encuentran contenidas en el texto. Del mismo modo, podría evaluarse el desempeño del modelo al utilizar modelos de Deep Learning como *Transformers* que aprendan a partir del conjunto de datos obtenido y puedan brindar un conjunto de elementos preliminar, ya sean extraídos directamente o deducidos a partir del contexto del texto analizado, que puedan describir el sistema y su estructura, de modo que los componentes extraídos se acerquen más a la interpretación humana de las historias de usuario.

A. Anexo: Prototipo

El dataset utilizado en este trabajo puede ser accedido por medio del siguiente link en GitHub:

<https://github.com/matovaro/PyUNML-DataSet>

El prototipo desarrollado durante este trabajo puede encontrarse en GitHub, en el siguiente repositorio:

<https://github.com/matovaro/PyUNML>

Bibliografía

- [1] A. Batool, Y. Motla, B. Hamid, S. Asghar, M. Mukhtar, and M. Ahmed, “Comparative study of traditional requirement engineering and agile requirement engineering,” pp. 1006–1014, 01 2013.
- [2] H. Gomaa, “Software modeling and design: Uml, use cases, patterns, and software architectures,” *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures*, pp. 1–550, 01 2011.
- [3] J. Rumbaugh, G. Booch, and I. Jacobson, *The Unified Modeling Language Reference Manual*. Addison-Wesley, 2010.
- [4] R. Lee, *Artificial Intelligence in Daily Life*. 01 2020.
- [5] C. Narawita and K. Vidanage, “Uml generator – use case and class diagram generation from text requirements,” *International Journal on Advances in ICT for Emerging Regions (ICTer)*, vol. 10, p. 1, 01 2018.
- [6] S. Vemuri, S. Chala, and M. Fathi, “Automated use case diagram generation from textual user requirement documents,” pp. 1–4, 04 2017.
- [7] F. Gilson and C. Irwin, “From user stories to use case scenarios - towards a generative approach,” 12 2018.
- [8] S. Nasiri, Y. Rhazali, M. Lahmer, and N. Chenfour, “Towards a generation of class diagram from user stories in agile methods,” *Procedia Computer Science*, vol. 170, pp. 831–837, 01 2020.

-
- [9] S. Nasiri, Y. Rhazali, M. Lahmer, and A. Adadi, “From user stories to uml diagrams driven by ontological and production model,” *International Journal of Advanced Computer Science and Applications*, vol. 12, 01 2021.
- [10] A. M. Maatuk and E. A. Abdelnabi, “Generating uml use case and activity diagrams using nlp techniques and heuristics rules,” in *International Conference on Data Science, E-Learning and Information Systems 2021*, DATA’21, (New York, NY, USA), p. 271–277, Association for Computing Machinery, 2021.
- [11] S. Ahmed, A. Ahmed, and N. U. Eisty, “Automatic transformation of natural to unified modeling language: A systematic review,” in *2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA)*, pp. 112–119, 2022.
- [12] E. A. Abdelnabi, A. M. Maatuk, and M. Hagal, “Generating uml class diagram from natural language requirements: A survey of approaches and techniques,” in *2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering MI-STA*, pp. 288–293, 2021.
- [13] M. H. Kassab, “The changing landscape of requirements engineering practices over the past decade,” *2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE)*, pp. 1–8, 2015.
- [14] I. K. Raharjana, D. Siahaan, and C. Fatichah, “User stories and natural language processing: A systematic literature review,” *IEEE Access*, vol. 9, pp. 53811–53826, 2021.
- [15] G. Lucassen, F. Dalpiaz, J. M. Werf, and S. Brinkkemper, “The use and effectiveness of user stories in practice,” in *Proceedings of the 22nd International Working Conference on Requirements Engineering: Foundation for Software Quality - Volume 9619*, REFSQ 2016, (Berlin, Heidelberg), p. 205–222, Springer-Verlag, 2016.
- [16] E. Btoush and M. Hammad, “Generating er diagrams from requirement specifications based on natural language processing,” *International Journal of Database Theory and Application*, vol. 8, pp. 61–70, 04 2015.

-
- [17] E. Meryem, K. Nafil, and R. Touahni, “Automatic transformation of user stories into uml use case diagrams using nlp techniques,” *Procedia Computer Science*, vol. 130, pp. 42–49, 01 2018.
- [18] A. Gupta, “Generation of multiple conceptual models from user stories in agile,” in *REFSQ Workshops*, 2019.
- [19] Y. Rigou, D. Lamontagne, and I. Khriiss, “A sketch of a deep learning approach for discovering uml class diagrams from system’s textual specification,” *2020 1st International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET)*, pp. 1–6, 2020.
- [20] T. Kochbati., S. Li., S. Gérard., and C. Mraidha., “From user stories to models: A machine learning empowered automation,” in *Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development - MODELSWARD*, pp. 28–40, INSTICC, SciTePress, 2021.
- [21] F. Dalpiaz, “Requirements data sets (user stories),” 2018.
- [22] F. Dalpiaz, A. Sturm, and P. Gieske, “Extraction of conceptual models: User stories vs. use cases,” 2020.
- [23] A. Névéol, H. Dalianis, S. Velupillai, G. Savova, and P. Zweigenbaum, “Clinical natural language processing in languages other than english: Opportunities and challenges,” *Journal of biomedical semantics*, vol. 9, p. 12, 03 2018.
- [24] M. S. M. Suhaimin, M. H. A. Hijazi, R. Alfred, and F. Coenen, “Natural language processing based features for sarcasm detection: An investigation using bilingual social media texts,” in *2017 8th International Conference on Information Technology (ICIT)*, pp. 703–709, 2017.
- [25] S. Elbasha, A. Elhawil, and N. Drawil, “Multilingual sentiment analysis to support business decision-making via machine learning models,” 12 2021.

- [26] S. N. Group., “Stanza – a python nlp package for many human languages.” Accedido en 24-09-2022 a <https://stanfordnlp.github.io/stanza/>, 2020.