UNIVERSIDAD NACIONAL DE COLOMBIA

Un modelo de resolución de ambigüedad de sentidos de palabras para mejorar la calidad de resultados en una arquitectura de educción de requisitos de software

Sebastián Alonso Gómez Arias

Universidad Nacional de Colombia

Facultad de Minas, Departamento de Ciencias de la Computación y de la Desición

Medellín, Colombia

2014

Un modelo de resolución de ambigüedad de sentidos de palabras para mejorar la calidad de resultados en una arquitectura de educción de requisitos de software[1]

Sebastián Alonso Gómez Arias

Tesis o trabajo de investigación presentado como requisito parcial para optar al título de:

Magister en Ingeniería de Sistemas

Director (a):

Ph.D. y Jaime Alberto Guzmán Luna

Línea de Investigación:

Razonamiento y Representación del Conocimiento en la Web

Grupo de Investigación:

Sistemas Inteligentes Web (SintelWeb)

Universidad Nacional de Colombia

Facultad de Minas, Departamento de Ciencias de la Computación y Decisión

Medellín, Colombia

2014

---

[1] El termino arquitectura de educción de requisitos de software es usado en esta tesis como: Cualquier sistema o modelo que permite transformar especificaciones de requisitos de software en lenguaje natural o controlado a modelos conceptuales de manera manual, semi-automática o automática.

(Dedicatoria)

A mi madre que cada mañana y cada noche busco hacer de mí un mejor ser humano. A mis familiares y amigos por su apoyo incondicional. A todas las personas que aportaron en todo este proceso de investigación. A mi amigo James Cummiskey Q.E.P.D.

“Si lo puedes medir lo puedes mejorar.”
-Peter Drucker

Agradecimientos

A mi director de tesis Jaime Alberto Guzmán Luna que ha hecho un trabajo digno de admirar porque siempre busco trascender y mirar “más allá” en cada etapa de esta investigación.

A los compañeros de mil batallas, Carlos Andres Velez Carvajal y Daniel Correa Botero, su ayuda y aporte fueron fundamentales en esta investigación. A mis familiares y amigos que me apoyaron incondicionalmente durante el desarrollo de esta tésis, especialmente a mi tía Magaly Arias por su esfuerzo y dedicación al final de esta investigación.

Al alma mater mi Universidad Nacional permitiendo hacer de mí un mejor profesional y un mejor ser humano y finalmente a dos proyectos de investigación bajo el marco de los cuales se desarrollo la presente investigación:

A Colciencias mediante su convocatoria de jóvenes investigadores 2013 bajo el proyecto: “un modelo de resolución de ambigüedad de sentidos de palabras para mejorar la calidad de resultados en una arquitectura de educción de requisitos de software”.

A la Dirección de Investigaciones de la Universidad Nacional de Colombia Sede Medellín DIME, a través de la Convocatoria del Programa Nacional de Proyectos para el Fortalecimiento de la Investigación, la Creación y la Innovación en Posgrados de la Universidad Nacional de Colombia 2013-2015, bajo el proyecto “modelo de procesamiento terminológico basado en ontologías para la desambiguación verbal en la educción de requisitos de software”

Resumen

El problema de ambigüedad de los sentidos de las palabras (polisémica) es un problema que es transversal a diversas áreas del conocimiento entre ellas a la ingeniería de software y sus sub-ramas como la educción de requisitos. En la fase de educción de requisitos se presentan problemas en la comunicación entre el analista y el interesado debido a factores como poca claridad del interesado, interpretación subjetiva del analista, dominios del discurso diferentes y ambigüedad semántica (polisémica, léxica y lingüística, etc.), estos factores dificultan la comprensión correcta y no ambigua de los requisitos. Las arquitecturas de educción de requisitos de software son el lugar donde se plasman los requisitos obtenidos desde el discurso y eso donde finalmente se evidencian los problemas de ambigüedad de esta fase (inconsistencia, duplicidad y no unificidad). En este trabajo se busca enfrentar la ambigüedad del sentido de las palabras presente en las arquitecturas de educción de requisitos de software y los errores asociados que genera la ambigüedad polísemica en requisitos de software. Para ello se propone un modelo de resolución de ambigüedad de sentidos de palabras para mejorar la calidad de resultados en una arquitectura de educción de requisitos de software.

Palabras clave: Ingeniería de Requisitos, Ingeniería de software, Desambiguación del significado de las palabras, Ambigüedad semántica, Lingüística computacional.

Abstract

The problem of ambiguity of word meanings (polysemy) is a problem that cuts across different knowledge areas including software engineering and its sub-branches such as requirements elicitation. In the phase of requirements elicitation problems arise in communication between the analyst and the person due to factors such as lack of clarity interested, subjective interpretation of the analyst different domains of discourse and semantic ambiguity (polysemy, lexical and linguistic, etc.) these factors make the correct and unambiguous understanding of the requirements. The architectures of software requirements elicitation are where the requirements obtained from the speech are reflected and that where finally the problems of ambiguity of this phase (inconsistency,

duplicity and no unificidad) are evident. This paper seeks to address the ambiguity of the meaning of words present in the architectures of software requirements elicitation and associated errors generated by the polysemous ambiguity in software requirements. For these model ambiguity resolution senses of words is proposed to improve the quality of results architecture software requirements elicitation.

Keywords: Requirements Engineering, Software Engineering, Disambiguation of the meaning of words, semantic ambiguity, Computational Linguistics.

Contenido

Introducción XX
Motivación XX
Preguntas de Investigación XXII
Objetivos XXV
Objetivo General XXV
Objetivos específicos XXV
Metodología utilizada XXVI
Contribuciones principales alcanzadas XXVIII
Organización del documento XXIX
Difusión de resultados XXXI

1. Marco teórico y estado del arte 33
1.1. Ingeniería de requisitos 33
1.1.1. Captura de requisitos 34
1.1.2. Definición de requisitos 35
1.1.3. Validación de requisitos 36
1.1.4. Problemas en la ingeniería de requisitos 36
1.2. Arquitecturas de educción de requisitos 39
1.2.1. Modelo de KAOS 39
1.3. Ambigüedad Semántica 42
1.3.1. Procesamiento del lenguaje natural 45
1.3.2. Sentido de las palabras. 48
1.3.3. Desambiguación del sentido de las palabras 50
1.3.3.1. Supervisados 51
1.3.3.2. No supervisados 52
1.3.3.3. Basados en conocimiento 53
1.4. Métricas de validación 55
1.4.1. Métricas de validación en la desambiguación de sentidos de las palabras 55
1.4.2. Métricas de evaluación en la arquitectura de educción de requisitos. 57
1.5. Comparación con enfoques actuales 58

2. Ambigüedad semántica en requisitos de software 63
2.1. El lenguaje 63
2.2. La Ambigüedad 64
2.2.1. Ambigüedad Morfológica 64
2.2.2. Ambigüedad Sintáctica 65
2.2.3. Ambigüedad Semántica 67
2.3. Transformación de requisitos de software en elementos constitutivos. 69
2.4. Errores generados por la ambigüedad semántica en la educción de requisitos de software. 72
2.4.1. Ambigüedad semántica 72

2.4.2. Inconsistencia 72
2.4.3. Duplicidad 73
2.4.4. No unificidad. 74
2.5. Conclusiones 75

3. Evaluación de técnicas de desambiguación polisémica en ambientes de educción de requisitos de software 77
3.1. Introducción 77
3.2. Técnicas de desambiguación 77
3.2.1. Lesk simplificado 78
3.2.2. Lesk adaptado 80
3.2.3. Algoritmo MFS (Most Frequent Sense) 82
3.2.4. UKB 83
3.3. Evaluación de las técnicas de desambiguación. 85
3.4. Conclusiones 88

4. Modelo de desambiguación de los sentidos de las palabras para minimizar errores en la identificación de requisitos de software 90
4.1. Introducción 90
4.2. Capa de desambiguación 92
4.2.1. Proceso morfosintáctico 92
4.2.2. Filtrado de coeficientes de sentidos de palabras mediante la técnica de mínimos cuadrados para la desambiguación semántica polisémica 94
4.3. Capa de validación de errores en elementos constitutivos 99
4.3.1. Detección de errores en elementos constitutivos. 99
4.3.1.1. Detección de inconsistencias en elementos constitutivos 99
4.3.1.2. Identificación de duplicidad en elementos constitutivos. 100
4.3.1.3. Identificación de no unificidad en elementos constitutivos 100
4.3.2. Corrección de errores en elementos constitutivos 101
4.4. Modelo de integración de propósito general usando wrappers 102
4.5. Conclusiones 105

5. NEONPL: Un desambiguador semántico polisémico para la identificación de requisitos de software. 107
5.1. Arquitectura del sistema 107
5.1.1. Freeling 108
5.1.2. Multilingual Central Repository y Wordnet. 110
5.2. NEONPL, esquema funcional 112
5.2.1. Módulo Adaptador 113
5.2.2. Módulo de desambiguación de elementos constitutivos usando filtrado de coeficientes. 119
5.2.3. Módulo de identificación de errores y corrección interactiva de errores con el usuario. 122
5.3. Diagrama de Clases NEONPL 130
5.4. Esquema de Secuencias NEONPL 133
5.5. Conclusiones 136

6. Pruebas y resultados 138
6.1. Evaluación de la desambiguación de los sentidos de las palabras en los elementos constitutivos. 138
6.1.1. Caso de estudio ambulancia 139

6.1.2. Caso de estudio ascensor 140
6.1.3. Caso de estudio Pizzería 141
6.1.4. Caso de estudio Carrefour 143
6.1.5. Caso de estudio Colcerámicas. 144
6.1.6. Caso de estudio aseguradora 145
6.1.7. Caso de estudio sistema de Juegos. 146
6.2. Resumen de resultados de desambiguación de los sentidos de las palabras sobre los casos de estudio. 147
6.2.1. Análisis de Cobertura 147
6.2.2. Análisis de Recuerdo 149
6.2.3. Análisis de precisión 150
6.2.4. Análisis de la métrica F1. 152
6.3. Comparación de la desambiguación de los sentidos de las palabras. 154
6.4. Rendimiento del módulo de desambiguación. 157
6.5. Evaluación del módulo de detección de errores. 158
6.6. Conclusiones 160

7. Conclusiones y trabajo futuro. 163
7.1. Resumen de Resultados y Contribuciones 163
7.2. Direcciones para trabajo futuro 165

A. Anexo Casos de estudio detallados. 167
A.1. Caso de estudio “Ambulancia de Londres” 167
A.2. Caso de estudio sistema de Ascensor 171
A.3. Caso de estudio PizzerÍa 175
A.4. Caso de estudio Carrefour 178
A.5. Caso de estudio Colcerámicas 181
A.6. Caso de estudio Aseguradora 183
A.7. Caso de estudio Sistema de juegos 185
A.8. Tabla comparativa técnicas de desambiguación vs Filtrado de coeficientes. 187

B. Anexo Algoritmos y Clases implementadas del sistema NEONPL. 191

Abreviaturas

Abreviatura	Término
WSD	Word Sense Disambiguation (Desambiguacion del sentido de las palabras.)
Alesk	Algoritmo Lesk Adaptado
Slesk	Algoritmo Lesk Simplificado
MFS	Algoritmo del sentido mas frecuente (Most Frequent Sense)

Abreviatura	Término
UKB	Algoritmo de desambiguación (Graph based Word Sense Disambiguation and Similarity)
KAOS	Diagrama de objetivos de KAOS (Knowledge Acquisition in Automated Specification), Propuesto por Axel Van Lamsweerde.
EDC	Elemento constitutivo del diagrama o modelo, ejemplo: objetivos, agentes,entidades y operaciones.

Índice de figuras

Figura 0.1 *Problema de ambigüedad en la educción de requisitos de software. Fuente: Elaboración Propia*..... XXI
Figura 0.2 *Factores de fracaso en proectos de software. Fuente: Elaboración propia.*. XXII
Figura 1.1 *Ejemplo modelo de objetivos de KAOS. Fuente: Adaptado de* [16] 40
Figura 1.2: *Porción del Meta-Modelo de KAOS Fuente: Adaptado de* [20] 41
Figura 1.5 *Ejemplo arquitectura de educción de requisitos y sus respectivos elementos constitutivos. Fuente: Elaboración propia*..... 42
Figura 1.6 *Descripción general del procesamiento del lenguaje natural. Fuente: Elaboración Propia.*..... 46
Figura 1.7 *Procesos morfológicos y sintácticos básicos de tratamiento de textos. Fuente: Elaboración Propia.*..... 47
Figura 1.8 *Ejemplo tareas de procesamiento morfosintáctico aplicadas a un requisito de software. Fuente: Elaboración Propia.* 48
Figura 1.9: *Ejemplo estructura de una frase con palabras polisémicas Fuente: Elaboración Propia*..... 49
Figura 2.1 *Primer posible árbol sintáctico para la frase "El sistema debe comprar los libros baratos." Fuente: Elaboración Propia.* 66
Figura 2.2 *Segundo posible árbol sintáctico para la frase "El sistema debe comprar los libros baratos." Fuente: Elaboración Propia.* 67
Figura 2.3 *Ejemplo problema de inconsistencia entre elementos constitutivos. Fuente: Elaboración Propia.*..... 73
Figura 2.4 *Ejemplo elementos constitutivos no unificados Fuente: Elaboración Propia.*.. 74
Figura 3.1 *Algoritmo Simplified Lesk utilizado en la presente tesis. Fuente: Elaboración Propia.* 79
Figura 3.2 *Algoritmo Adapted Lesk utilizado en la presente tesis. Fuente: Elaboración Propia.* 81

Figura 3.3 *Algoritmo Most Frequent Sense utilizado en la presente tesis. Fuente: Elaboración Propia.* 82
Figura 3.4 *Algoritmo UKB utilizado en la presente tesis. Fuente: Elaboración Propia*..... 84
Figura 3.5 Evaluación de las técnicas de desambiguación. *Fuente: Elaboración Propia* 88
Figura 4.1 *Modelo de desambiguación de los sentidos de las palabras para minimizar errores en la identificación de requisitos de software. Fuente: Elaboración Propia*..... 91
Figura 4.2 *Algoritmo de segmentar y etiquetar.* 92
Figura 4.3 *Proceso por etapas del modelo de desambiguación Fuente: Elaboración Propia.*..... 95
Figura 4.4 *Algoritmo de filtrado de coeficientes para encontrar el mejor sentido de una palabra. Fuente: Elaboración Propia.* 97
Figura 4.5 *Detección formal de palabras contradictorias en elementos constitutivos. Fuente: Elaboración propia.*.....100
Figura 4.6 *Regla formal para la detección de elementos constitutivos duplicados. Fuente: Elaboración propia.* 100
Figura 4.7 *Regla formal para la identificación de elementos constitutivos no unificados. Fuente: Elaboración propia.*.....101
Figura 4.8. *Esquema del modelo adaptador (Wrapper) de propósito general. Fuente: Elaboración propia.* 102
Figura 4.9 *Algoritmo de validación de reglas morfosintácticas asociadas a KAOS, en elementos constitutivos. Fuente: Elaboración propia.* 105
Figura 5.1 *Arquitectura general del modelo propuesto. Fuente: Elaboración Propia.*.....108
Figura 5.2 *Conjunto de clases del paquete freeling usado por el sistema NEONPL.* 110
Figura 5.3 *Tablas de multilingual central repository usadas por el sistema NEONPL. Fuente: Elaboración propia.*.....112
Figura 5.4 *Esquema funcional del sistema NEONPL. Fuente: Elaboración Propia.* 113
Figura 5.5 *Ejemplo arquitectura de educción de requisitos presentada en formato XML siguiendo la plantilla de la figura 5.4. Fuente: Elaboración Propia.* 114
Figura 5.6 *XMLSchema de la arquitectura de educción de requisitos de software sobre la cual se valida el documento XML con la arquitectura de educción de requisitos. Fuente: Elaboración Propia.* 115
Figura 5.7 *Interfaz del módulo adaptador para recibir la arquitectura de educción de requisitos de software en formato XML.* 116
Figura 5.8 *Configuración de reglas morfosintácticas para la validación de los elementos constitutivos en la arquitectura de educción de requisitos. Fuente: Elaboración Propia.*117
Figura 5.9 *Algoritmo de Validación de reglas morfosintácticas implementado en JAVA. Fuente: Elaboración Propia.* 118
Figura 5.10 *Algoritmo de filtrado de coeficientes presentado en lenguaje JAVA. Fuente: Elaboración Propia.* 120
Figura 5.11. *Consulta SQL para la extracción de la definición de un sentido asociado a una palabra. Fuente: Elaboración Propia.*.....121
Figura 5.12 *Interfaz gráfica de presentación de los elementos constitutivos desambiguados. Fuente: Elaboración Propia.* 122

Figura 5.13 *Algoritmo de detección de inconsistencias implementado en JAVA. Fuente: Elaboración Propia.* 123
Figura 5.14 *Consulta SQL para la extracción de antónimos de un sentido. Fuente: Elaboración Propia.* 123
Figura 5.15 *Interfaz gráfica presentación de errores de inconsistencia. Fuente: Elaboración Propia.* 124
Figura 5.16 *Algoritmo de detección de duplicidades en elementos constitutivos. Fuente: Elaboración Propia.* 125
Figura 5.17. *Interfaz gráfica de presentación de errores de duplicidad. Fuente: Elaboración propia.* 126
Figura 5.18. *Algoritmo para la detección de no unificidades en elementos constitutivos. Fuente: Elaboración propia.* 127
Figura 5.19 *Interfaz para la presentación de errores de no unificidad al usuario. Fuente: Elaboración propia.* 128
Figura 5.20 *Documento XML de salida del sistema NEONPL con la arquitectura de educción de requisitos desambiguada. Fuente: Elaboración propia.* 129
Figura 5.21 *Diagrama de clases del sistema NEONPL. Fuente: Elaboración propia.* ... 131
Figura 5.22 *Diagrama Entidad-Relación Implementado por el sistema NEONPL. Fuente: Elaboración propia.* 132
Figura 5.23 *Primera parte del esquema de secuencias del sistema NEONPL. Fuente: Elaboración Propia.* 134
Figura 5.24 *Segunda parte del diagrama de secuencias del sistema NEONPL. Fuente: Elaboración Propia.* 135
Figura 5.25 *Tercera parte del esquema de secuencias del sistema NEONPL Fuente: Elaboración Propia.* 136
Figura 6.1 *Cobertura por caso de estudio.* 148
Figura 6.2 *Cobertura versus elementos constitutivos* 148
Figura 6.3 *Recuerdo por caso de estudio* 149
Figura 6.4 *Recuerdo versus elementos constitutivos* 150
Figura 6.5 *Precisión por caso de estudio.* 151
Figura 6.6 *Precisión vs Elementos constitutivos.* 152
Figura 6.7 *F1 por caso de estudio* 153
Figura 6.8 *F1 versus Elementos constitutivos* 153
Figura 6.9 *Comparación técnicas de desambiguación por medio de la métrica F1.* 156
Figura 6.10 *Tiempo de ejecución para realizar las tareas de desambiguación variando el número de elementos constitutivos a procesar.* 157

Índice de tablas.

Tabla 1-1 Autores que exponen los tipos de ambigüedad presente en los requisitos de software y su propuesta para evitarla. 38

Tabla 1-2 Autores que exponen los tipos de ambigüedad presente en el lenguaje..... 44
Tabla 1-3 Resumen de los enfoques y técnicas utilizadas para atacar la ambigüedad del sentid de las palabras..... 54
Tabla 1-4. Métricas utilizadas para la evaluación de los resultados en la desambiguación de los sentidos de las palabras en los elementos constitutivos. 55
Tabla 1-5. Métricas de evaluación en la arquitectura de educción de requisitos..... 57
Tabla 1-6 Resumen comparación de trabajos relacionados. 60
Tabla 2-1. Ejemplos ambigüedades semánticas en requisitos de software en español. . 68
Tabla 2-2. Reglas morfosintácticas para la transformación de requisitos de software en elementos constitutivos bajo la arquitectura de educción de requisitos KAOS. 70
Tabla 2-3. Estructuras gramaticales utilizadas en la definición de las reglas usadas para la transformación de requisitos de software en elementos constitutivos. 71
Tabla 2-4. Ejemplo Duplicidad en elementos constitutivos 73
Tabla 3-1 Evaluación técnicas de desambiguación referenciados en el presente trabajo. 86
Tabla 4-1 Filtrado de coeficientes para un sentido i..... 98
Tabla 6-1 *Tabla general de resultados de desambiguación obtenidos para el caso de estudio de la ambulancia.* 139
Tabla 6-2 Resultado métricas evaluadas para el caso de estudio ambulancia. 140
Tabla 6-3 *Tabla general de resultados de desambiguación obtenidos para el caso de estudio del ascensor.* 140
Tabla 6-4 Resultado métricas evaluadas para el caso de estudio ascensor. 141
Tabla 6-5 *Tabla general de resultados de desambiguación obtenidos para el caso de estudio de la pizzería.* 142
Tabla 6-6 Resultado métricas evaluadas para el caso de estudio pizzería. 142
Tabla 6-7 *Tabla general de resultados de desambiguación obtenidos para el caso de estudio de Carrefour.* 143
Tabla 6-8 Resultado métricas evaluadas para el caso de estudio de Carrefour.....143
Tabla 6-9 *Tabla general de resultados de desambiguación obtenidos para el caso de estudio del sistema de juegos.* 144
Tabla 6-10 Resultado métricas evaluadas para el caso de estudio de Colcerámicas. ...145
Tabla 6-11 *Tabla general de resultados de desambiguación obtenidos para el caso de estudio de la aseguradora* 145
Tabla 6-12 Resultado métricas evaluadas para el caso de estudio de Aseguradora. 145
Tabla 6-13 *Tabla general de resultados de desambiguación obtenidos para el caso de estudio del sistema de juegos.* 146
Tabla 6-14 Resultado métricas evaluadas para el caso de estudio de Aseguradora. 146
Tabla 6-15 Resumen resultados por casos de estudio. 147
Tabla 6-16 Resumen comparación del modelo de desambiguación de filtrado de coeficientes vs técnicas de desambiguación que son usadas por el modelo. 154
Tabla 6-17 Tiempo tomado por el modelo de desambiguación versus los elementos constitutivos a desambiguar. 157
Tabla 6-18 Resumen datos obtenidos sobre los casos de estudio en la detección inconsistencias, duplicidades y no unificidades. 158

Tabla 6-19 Valores obtenidos al aplicar las métricas sobre la tabla anterior 159
Tabla A-1 XML Entrada Ambulancia 167

Introducción

Motivación

El proceso de la educción de requisitos que se realiza en la primera fase del ciclo de vida del software, consiste en descubrir y adquirir el conocimiento necesario de las necesidades del interesado en relación con el problema o necesidad que se desea computarizar, ésta fase permite a su vez identificar, analizar y validar los requisitos que el futuro software deberá satisfacer. En esta fase, se presentan problemas en la comunicación entre el analista y el interesado debido a: La (i) diferencia de dominios que el cliente y el analista manejan, lo que conlleva a diferentes interpretaciones de los requisitos, provocando la entrega de un producto que en la mayoría de los casos, no cumple con las necesidades y expectativas del cliente, Los (ii) tipos de ambigüedades presentes en el discurso tales como ambigüedad léxica o morfológica, ambigüedad sintáctica o estructural y ambigüedad semántica (Polisémica y referencial), que dificulta la comprensión correcta y no ambigua de los requisitos, La (iii) la subjetividad que el analista incorpora al proceso de desarrollo de software debido a las creencias, experiencias y conocimiento del mundo que tiene desviando en muchos casos el real enfoque del cliente y (iv) a la poca claridad por parte del interesado al momento de expresar sus necesidades, dando lugar a mas ambigüedades en las especificaciones de requisitos.

La actividad de la educción de requisitos, se basa generalmente en un diálogo realizado entre analista e interesado. En este diálogo el analista es quien aporta su conocimiento para la elaboración de software y, el interesado, aporta su conocimiento sobre el área del dominio. Una vez finaliza el diálogo, queda como elemento de salida en algunos casos un texto en lenguaje natural que consigna el conocimiento de los interesados, y/o un conjunto de diagramas propios de las arquitecturas de educción de requisitos, que representan la interpretación que realizó el analista de dicho conocimiento.

Los requisitos de software generalmente son plasmados en diagramas o modelos conceptuales que hacen parte de arquitecturas o modelos de educción de requisitos (tales como KAOS, casos de uso, diagramas de actividades y modelos entidad relación entre otros) y es en estas arquitecturas o modelos donde se evidencian las fallas en el proceso de educción de requisitos debido a que se encuentran requisitos ambiguos o mal estructurados.

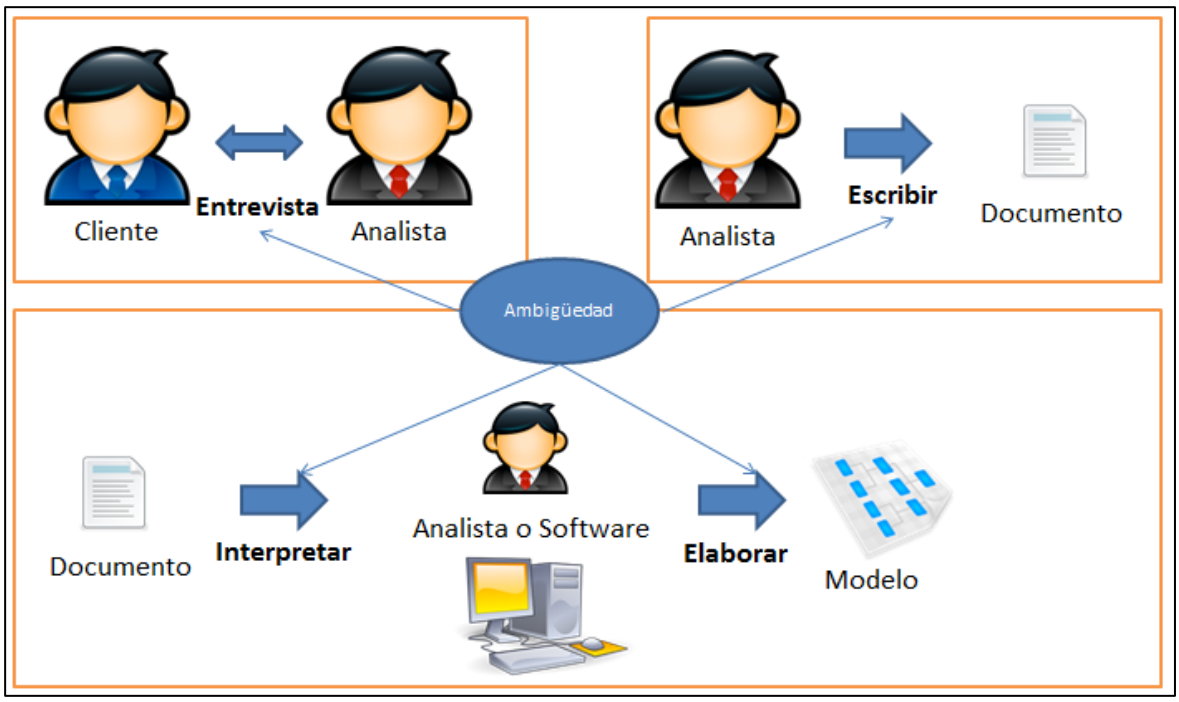


Figura 0.1 *Problema de ambigüedad en la educción de requisitos de software. Fuente: Elaboración Propia*

Es por esto que aproximadamente el 53% de los proyectos de software que fracasan, fracasan debido a errores e inconsistencias en la fase de educción de requisitos como se expresa en [1].

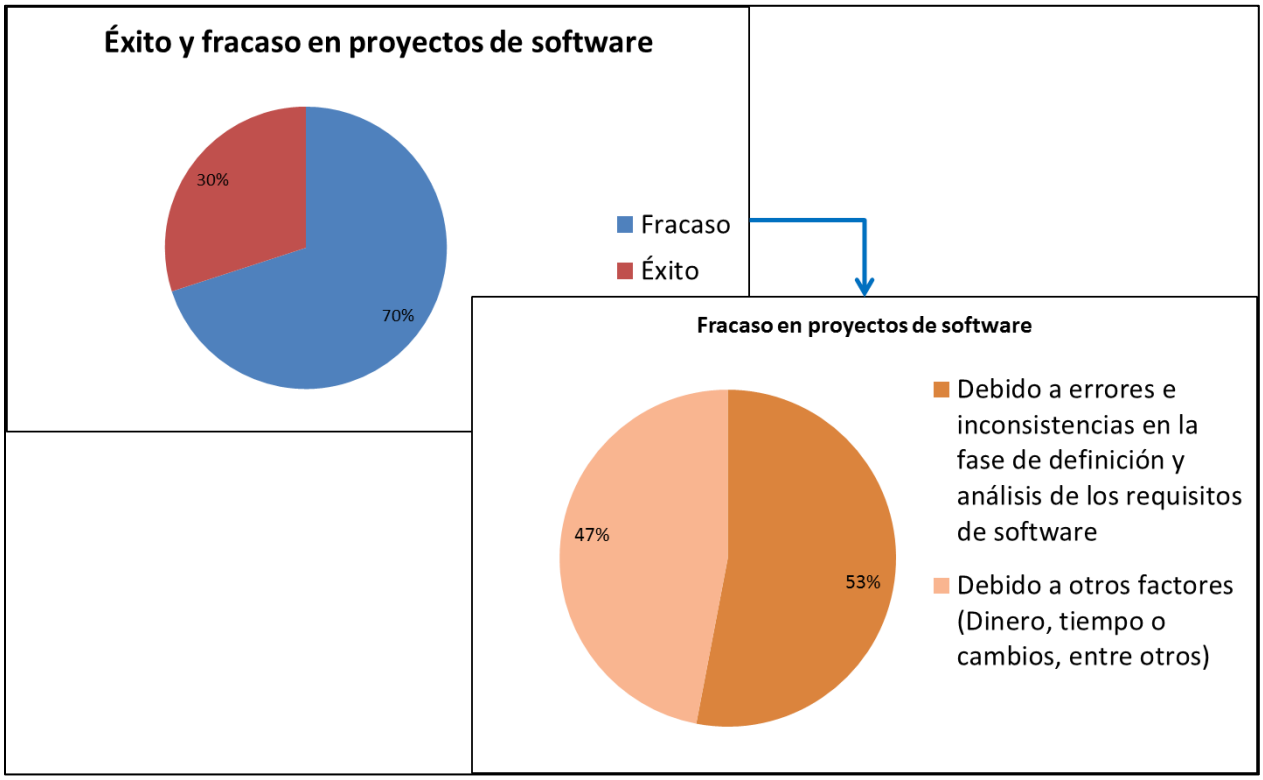


Figura 0.2 *Factores de fracaso en proectos de software.*
Fuente: Elaboración propia.

Para enfrentar el problema de la ambigüedad del significado de las palabras en una especificación de requisitos de software en lenguaje natural, en esta tesis se propone un modelo de resolución de la ambigüedad de sentidos de palabras, que permita mejorar la calidad de resultados en las arquitecturas de educción de requisitos, mejorando algunas de las propuestas en el estado del arte, que hacen uso de vocabularios controlados en la especificación de tales requisitos para evitar este problema. Con este modelo se busca acercarse más a la realidad del usuario, quien en general usa su propio vocabulario para expresar dichos requisitos y no vocabularios restringidos ajenos a su realidad.

Preguntas de Investigación

La pregunta general de investigación que esta tesis trata de responder es la siguiente:

¿Cómo se puede implementar un modelo de resolución de ambigüedad de sentidos de palabras que permita mejorar la calidad de resultados en una arquitectura de educción de requisitos de software?

Con el fin de poder dar una respuesta a la anterior pregunta, se define un conjunto de preguntas de investigación que abordan el problema en detalle.

Pregunta 1: ¿Cómo identificar y caracterizar formalmente y de manera sistémica las ambigüedades semánticas que generan errores durante el proceso de educción de requisitos de software?

De esta pregunta surgen los siguientes interrogantes.

- ¿Cuáles son las ambigüedades semánticas que generan errores durante el proceso de educción de requisitos de software?
- ¿Cómo las ambigüedades semánticas generan errores en el proceso de educción de requisitos de software?
- ¿Cómo se pueden caracterizar formalmente y de manera sistémica las ambigüedades semánticas que generan errores durante el proceso de educción de requisitos de software?

Pregunta 2: ¿Cómo evaluar las diferentes técnicas de resolución de ambigüedad de sentidos de palabras para seleccionar la más adecuada con el fin de minimizar errores en la identificación de requisitos causados por la ambigüedad de los sentidos de las palabras durante la educción de requisitos de software?

De esta pregunta surgen los siguientes interrogantes:

- ¿Cuáles son las técnicas de resolución de ambigüedad de sentidos de palabras?
- ¿Cómo se minimizan los errores en la identificación de requisitos al utilizar una técnica de resolución de ambigüedad de los sentidos de las palabras?
- ¿Cómo determinar las técnicas de resolución de la ambigüedad de sentidos de palabras más adecuadas para minimizar errores en la identificación de requisitos de software?

Pregunta 3: ¿Cómo definir un modelo de resolución de ambigüedad polisémica con base en las técnicas seleccionadas de resolución de la ambigüedad polisémica que permita minimizar los errores ocasionados por la ambigüedad polisémica en un proceso de educción de requisitos de software?

De esta pregunta se desprenden los siguientes interrogantes:

- ¿Cómo definir un modelo de resolución de la ambigüedad de sentidos de palabras?
- ¿Cómo unir las técnicas de resolución de la ambigüedad polisémica seleccionadas en un modelo que permita minimizar los errores ocasionados por la ambigüedad polisémica?

Pregunta 4: ¿Cómo definir un modelo de integración de propósito general que permita la adición del modelo de resolución de ambigüedad polisémica propuesto a una arquitectura de educción de requisitos de software?

De esta pregunta se desprenden los siguientes interrogantes:

- ¿Cómo integrar en un modelo de propósito general un modelo de resolución de la ambigüedad polisémica y una arquitectura de educción de requisitos de software?

Pregunta 5: ¿Cómo validar el modelo de resolución de ambigüedad de sentidos de palabras y el modelo de integración de propósito general propuestos mediante la implementación de un ambiente de prueba que permita evaluar el desempeño de una arquitectura existente de educción de requisitos de software con y sin el uso de los modelos propuestos con el fin de validar sus fortalezas y debilidades?

De esta pregunta se desprenden los siguientes interrogantes:

- ¿Cuáles pruebas de validación del modelo son adecuadas para identificar las fortalezas y debilidades del modelo propuesto?
- ¿Cómo realizar las pruebas de validación del modelo?
- ¿Es posible realizar un análisis de las fortalezas y debilidades del modelo a partir de las pruebas de validación realizadas?

Objetivos

Objetivo General

Proponer un modelo de resolución de ambigüedad de sentidos de palabras para mejorar la calidad de resultados en una arquitectura de educción de requisitos de software

Objetivos específicos

I. Identificar las ambigüedades semánticas que generan errores durante el proceso de educción de requisitos de software, con el fin de caracterizar estas ambigüedades formalmente y de manera sistémica.

II. Evaluar diferentes técnicas de resolución de ambigüedad de sentidos de palabras, con el fin de seleccionar la más adecuada para minimizar errores en la identificación de requisitos, causados por la ambigüedad de los sentidos de las palabras durante la educción de requisitos de software.

III. Definir un modelo de resolución de ambigüedad de sentidos de palabras, con base a las técnicas seleccionadas en el objetivo anterior que permita minimizar los errores ocasionados por la ambigüedad de los sentidos de las palabras en un proceso de educción de requisitos de software.

IV. Definir un modelo de integración de propósito general, que permita la adición del modelo de resolución de ambigüedad de sentidos de palabras propuesto, a una arquitectura de educción de requisitos de software.

V. Validar el modelo de resolución de ambigüedad de sentidos de palabras y el modelo de integración de propósito general, propuestos mediante la implementación de un ambiente de prueba que permita evaluar el desempeño de una arquitectura existente de educción de requisitos de software con y sin el uso de los modelos propuestos, con el fin de validar sus fortalezas y debilidades.

Metodología utilizada

Esta tesis se guió con una metodología compuesta por cinco fases, cada una de estas fases llevaron consigo una serie de etapas con el fin de cumplir el objetivo asociado a cada fase de esta tesis, a continuación se describen cada una de las fases, sus etapas y los logros asociados a cada una.

Fase 1. Identificación de las ambigüedades semánticas, que generan errores durante el proceso de educción de requisitos de software, con el fin de caracterizar estas ambigüedades formalmente y de manera sistémica.

Se definieron y presentaron las ambigüedades que afectan el lenguaje natural escrito, como lo son las ambigüedades de tipo morfológico o léxico, sintáctica o estructural y semánticas de tipo polisémica y de tipo referencial.

Se formalizaron y especificaron de manera sistémica las ambigüedades que afectan el lenguaje natural escrito, al realizar una descripción de cada ambigüedad mediante una definición matemática y lógica. También se estableció la relación con los requisitos de software, demostrando como cada una de estas ambigüedades genera errores en los requisitos de software

Fase 2. Evaluación de diferentes técnicas de resolución de ambigüedad de sentidos de palabras con el fin de seleccionar la más adecuada para minimizar errores en la identificación de requisitos causados por la ambigüedad de los sentidos de las palabras durante la educción de requisitos de software.

Se escogió como ambigüedad a tratar, la ambigüedad de los sentidos de palabras, es decir la ambigüedad semántica de tipo polisémica, con base a esto se presentaron diversas técnicas de resolución de la ambigüedad polisémica con el fin de evaluarlas de acuerdo a los resultados históricos del estado del arte y con base a esta evaluación se eligió un subconjunto de técnicas para unificarlas y realizar una mejor desambiguación polisémica.

Fase 3. Definición de un modelo de resolución de ambigüedad de sentidos de palabras con base a las técnicas seleccionadas en la fase anterior que permita minimizar los

errores ocasionados por la ambigüedad de los sentidos de las palabras en un proceso de educción de requisitos de software.

Se creó un modelo de resolución de ambigüedad de sentidos de las palabras que agrupa algunas de las técnicas expuestas en la fase anterior, dada su evaluación para generar una mejor desambiguación asociada a un proceso de educción de requisitos de software.

Fase 4. Definición de un modelo de integración de propósito general, que permita la adición del modelo de resolución de ambigüedad de sentidos de palabras propuesto, a una arquitectura de educción de requisitos de software.

Se definió un modelo que permite procesar una arquitectura de educción de requisitos instanciada. Este modelo adaptador, permite validar las entradas que el sistema acepta mediante una plantilla XMLSchema, con el fin de que él se pueda integrar con cualquier sistema de educción de requisitos de software. Este modelo se encarga de realizar la validación de las reglas morfosintácticas asociadas a KAOS para los elementos constitutivos introducidos.

Fase 5. Validación del modelo de resolución de ambigüedad de sentidos de palabras y el modelo de integración de propósito general, propuesto mediante la implementación de un ambiente de prueba que permita evaluar el desempeño de una arquitectura existente de educción de requisitos de software con y sin el uso de los modelos propuestos, con el fin de validar sus fortalezas y debilidades.

Se creó un prototipo que permitió ejecutar el modelo de resolución de la ambigüedad de sentidos de palabras y el modelo de integración de propósito general.

Se diseñaron un conjunto de pruebas que permitieron evaluar el desempeño del modelo de resolución de ambigüedad de sentidos de palabras y el modelo de integración de propósito general mediante el prototipo implementado.

Se validaron las fortalezas y debilidades del modelo al realizar las pruebas y evaluar el desempeño.

Contribuciones principales alcanzadas

La **primera** y **principal contribución** de esta tesis, es el desarrollo de un modelo de resolución de la ambigüedad del sentido de las palabras, que mejora la calidad de los resultados en una arquitectura de educción de requisito de software, el cual maneja de manera conjunta los siguiente aspectos:

- No está ligado a ninguna arquitectura educción de requisitos en particular, por lo que este modelo puede ser usado de propósito general con cualquier arquitectura de educción de requisitos de software.
- Realiza un proceso de desambiguación de los sentidos de las palabras, basado en la unión de varias técnicas de resolución de la ambigüedad de los sentidos de las palabras.
- Permite detectar errores e inconsistencias entre los requisitos de software, al tener reglas formales que permiten detectar los errores.

Adicionalmente como valor agregado, este modelo permite también el manejo de los siguientes aspectos no contemplados en los objetivos de esta tesis:

- Caracterización morfosintáctica de los requisitos de software, introducidos bajo el modelo de KAOS.
- Implementación de procesos morfosintácticos, necesarios para realizar una desambiguación de los sentidos de las palabras en los requisitos de software.

Otras contribuciones específicas de esta tesis se listan a continuación:

Segunda contribución: Se creó un sistema prototipo llamado NEONPL el cual permite procesar una arquitectura de educción de requisitos introducida en el formato XML definido para esta tesis.

Las características básicas de este sistema-prototipo se listan a continuación:

1. Este sistema prototipo extrae las características morfosintácticas de los requisitos de software que componen los modelos de educción de requisitos y que son procesados por las arquitecturas de educción de requisitos, es decir los

elementos constitutivos del modelo de KAOS como objetivos, agentes, operaciones y entidades..

2. Valida los elementos constitutivos mediante reglas morfosintácticas que permiten determinar si el requisito se encuentra bien definido.
3. Realiza una desambiguación semántica polisémica mediante un filtrado de coeficientes usando mínimos cuadrados, sobre cuatro técnicas de resolución de la ambigüedad de los sentidos de las palabras basadas en conocimiento.
4. Presenta los resultados en una interfaz gráfica para que el usuario valide la desambiguación obtenida para los elementos constitutivos y le presenta al usuario los errores identificados.

Tercera contribución: Diseño y desarrollo de un mecanismo interactivo de corrección de errores ocasionados por la ambigüedad semántica polisémica.

- Permite identificar inconsistencias, duplicidades y no unificidades en los requisitos de software.
- Presenta los errores en una interfaz gráfica tipo web para que el usuario elija la mejor corrección posible de estos errores.
- Exporta los resultados en formato XML con el fin de que puedan ser procesados por otros sistemas.

Organización del documento

Este documento de tesis se encuentra organizado de la siguiente manera:

El capítulo 1, proporciona una visión general de literatura asociado en primer lugar al proceso de educción de requisitos, posteriormente a los problemas que se asocian los procesos de educción de requisitos y se mencionan entre ellos los problemas de ambigüedad presentes. También se presentan los mecanismos de desambiguación del sentido de las palabras así como las métricas utilizadas para evaluar estos mecanismos de desambiguación, finalmente se presentan algunos trabajos relacionados, que permiten contextualizar la presente tesis.

El capítulo 2, presenta el conjunto de las ambigüedades que afectan el lenguaje natural escrito y se explica por qué, afectan también a los requisitos de software. Estas

ambigüedades se presentan de manera formal y sistémica haciendo especial énfasis en la ambigüedad de los sentidos de las palabras, la cual será la ambigüedad a tratar en la presente tesis. En este capítulo se presentan los errores de inconsistencia, duplicidad y no unificidad ocasionados por la ambigüedad semántica polisémica en requisitos de software.

El capítulo 3, presenta la evaluación de diversas técnicas de resolución de ambigüedad de los sentidos de las palabras, las cuales se comparan basándose en las métricas expuestas en el estado del arte, en este capítulo principalmente se expone una tabla en donde se realiza la comparación y se propone el enfoque a seguir de acuerdo a estas técnicas.

El capítulo 4, presenta el modelo de resolución de la ambigüedad de sentidos de palabras con base a las técnicas seleccionadas en la fase anterior. Lo anterior, con el fin de unir algunas de las técnicas expuestas en el capítulo 3, de manera que se realice una mejor desambiguación que si se usara una sola técnica de resolución. Se presenta en detalle el modelo que permite identificar y corregir los errores de inconsistencia, duplicidad y no unificidad.

El capítulo 5, presenta el modelo de integración de propósito general que, permite unir el modelo de resolución de la ambigüedad del sentido de las palabras, con el módulo de entrada del sistema que recibe la arquitectura de educción de requisitos instanciada.

El capítulo 6, presenta el sistema prototipo que implementa los modelos expuestos en los capítulos 4 y 5, con el fin de realizar las pruebas de validación sobre los modelos propuestos para así evaluar las fortalezas y debilidades del presente modelo.

Finalmente el capítulo 7, resume y discute el modelo propuesto, presenta las conclusiones generales de esta tesis y los trabajos futuros que se derivan de ella.

El anexo A, presenta los textos y las tablas extendidas con los resultados para los casos de prueba implementados en el presente sistema.

El anexo B, presenta los algoritmos extendidos del sistema NEONPL implementados en el lenguaje JAVA.

Difusión de resultados

Los resultados de esta tesis se presentaron en diferentes revistas nacionales e internacionales, capítulos de libro y conferencias. Así mismo de este trabajo de tesis se desprendieron otros trabajos relacionados con trabajos de grado (TDG). En esta sección se resume los diferentes resultados en publicaciones y proyectos asociados que soportan esta tesis.

Artículos Publicados en revistas Indexadas internacionales.

- Filtering Word Senses Using the Least Squares Technique for Word Sense Disambiguation, Jaime Alberto Guzmán-Luna, Sebastián Alonso Gómez Arias. International Journal of Engineering and Technology, e-ISSN : 0975-4024 p-ISSN ISSN: 2319-8613 , Publicado Diciembre de 2014.
- Extraction of goals and their classification in the KAOS model using natural language processing, Guzmán-Luna Jaime Alberto, Lezcano Rodriguez Luis, Gomez Arias Sebastian Gómez. Ingeniare. Revista chilena de Ingeniería, 2014. ISSN 0718-3305.

Artículos publicados en revistas internacionales.

- Desarrollo y construcción de una Ontología basada en el Metamodelo del Diagrama de Objetivos KAOS, Carlos Vélez Carvajal, Sebastián Gómez Arias y Jaime Guzmán-Luna. Revista coloquio de investigación multidisciplinaria, Volumen 1, Número 1, Octubre 2013 issn 2007-8102

Capítulos de libro Internacionales.

- Obtaining agents and entities from natural language, Guzmán-Luna Jaime Alberto, Lezcano Rodriguez Luis, Gomez Arias Sebastian Gómez. Lecture Notes in Electrical Engineering, 2014. ISSN: 1876-1100.

Revistas Nacionales.

- Caracterización de los elementos del diagrama de objetivos de kaos a partir de lenguaje natural, Jaime A. Guzmán, Luis Alfonso Lezcano Rodríguez, Sebastián Alonso Gómez Arias, Revista colombiana de tecnologías de avanzada (RCTA), ISSN: 1692-7257 - Volumen 1 – Número 21 – 2013.
- Uso de ontologías para la desambiguación de elementos constitutivos en la educción de requisitos de software. Jaime A. Guzmán-Luna, Sebastian Alonso Gómez Arias Revista QUID, (22). Octubre de 2014

Congresos y coloquios internacionales.

- Semantic characterization of goals in the KAOS model using word sense disambiguation Jaime A. Guzmán, Sebastian Alonso Gómez Arias, International Conference on Systems, Computing Sciences and Software Engineering (SCSS 14), Diciembre de 2014.
- Caracterización de los elementos del diagrama de objetivos de kaos a partir de lenguaje natural, Jaime A. Guzmán, Luis Alfonso Lezcano Rodríguez, Sebastián Alonso Gómez Arias, CIATE VIII, Congreso Internacional Electrónica y Tecnologías de Avanzada. Noviembre de 2012.
- Desarrollo y construcción de una Ontología basada en el Metamodelo del Diagrama de Objetivos KAOS, Carlos Vélez Carvajal, Sebastián Gómez Arias y Jaime Guzmán-Luna. Coloquio de investigación multidisciplinaria, Octubre de 2013.

Articulo sometidos a evaluación y pendientes por revisión.

- Un modelo de procesamiento de lenguaje natural para la detección de errores en requisitos de software. Sebastián Alonso Gómez Arias, Jaime Alberto Guzmán-Luna y Carlos Andrés Vélez Carvajal, Revista virtual Universidad Catolica del norte, Sometido Enero de 2015.
- Ontología para la descripción y desambiguación de los elementos del modelo de KAOS. Sebastián Alonso Gómez Arias, Jaime Alberto Guzmán-Luna y Carlos Andrés Vélez Carvajal, congreso CICOM Cartagena 2015, Sometido Febrero de 2015.

1. Marco teórico y estado del arte

En este capítulo se suministra el marco teórico y el estado del arte donde se fundamenta el presente trabajo. En la primera sección se presentan los principios básicos de la ingeniería de requisitos y los problemas asociados a ésta primera fase del ciclo de construcción de un software. En la segunda sección se presentan las arquitecturas de educción de requisitos. En la tercera sección se analiza el problema de la ambigüedad semántica. En la cuarta sección se presentan algunas métricas y mecanismos de validación y evaluación. Finalmente en la quinta sección se presentan algunos trabajos relacionados.

1.1. Ingeniería de requisitos

La ingeniería de requisitos, según Pressman en [2], es un conjunto de procesos, tareas y técnicas que permiten la definición y gestión de los requisitos de un producto de un modo sistemático. En definitiva, facilita los mecanismos adecuados para comprender las necesidades del cliente, analizando sus necesidades, confirmando su viabilidad, negociando una solución razonable, especificando la solución sin ambigüedad, validando la especificación y gestionando los requisitos para que se transformen en un sistema operacional.

La ingeniería de requisitos permite la gestión adecuada de los requisitos de un proyecto de desarrollo software. Además, mejora la capacidad para realizar planificaciones de los procesos de proyectos de desarrollo software puesto que el conocer qué se tiene que desarrollar permite una efectiva proyección de las actividades, recursos, costos y tiempos del proyecto. Según Sommerville en [3], se puede considerar como el proceso de comunicación entre los clientes, los usuarios del software y los desarrolladores del mismo.

Llevar a cabo de manera adecuada el proceso de ingeniería de requisitos disminuye la probabilidad de fracaso de un proyecto. Esto es porque los requisitos bien definidos permiten conocer de un modo conciso que debe realizar el software a desarrollar y hace orientar las actividades, recursos y esfuerzos de manera eficiente permitiendo la disminución de costos y retrasos. Todo ello provoca una mejora en la calidad del software puesto que los requisitos bien especificados podrán probarse de manera efectiva y, por tanto, cumplirse.

Dentro de las posibles estructuras que se pueden definir en la fase de ingeniería de requisitos, la propuesta con mayor seguimiento es la establecida por Lowe y Hall en [4]. En ella, el proceso de tratamiento de requisitos está compuesto por tres actividades:

- Captura de requisitos.
- Definición de requisitos.
- Validación de requisitos.

1.1.1. Captura de requisitos

La captura de requisitos es la actividad en la que un grupo especializado extrae de cualquier fuente de información disponible (documentos, aplicaciones existentes, entrevistas, etc.), las necesidades que debe cubrir dicho sistema. El proceso de captura de requisitos puede resultar complejo, debido a esto existen un conjunto de técnicas que permiten hacer este proceso de una forma más eficiente y precisa, obteniéndose necesidades y modelos del sistema.

A continuación se relacionan algunas técnicas que son utilizadas para esta actividad [5]:

- Entrevistas: Permiten tomar conocimiento del problema y comprender los objetivos de la solución buscada.
- Desarrollo de conjunto de aplicaciones: Es una práctica de grupo donde participan usuarios, analistas, administradores del sistema, y clientes y en la que se concretan las necesidades del sistema.
- Tormenta de ideas: Consiste en la acumulación de ideas sin evaluar las mismas. Ofrece una visión general de las necesidades del sistema pero sin ofrecer detalles concretos.

- Mapa conceptual: Grafos en los que los vértices representan conceptos y las aristas representan posibles relaciones entre dichos conceptos. Estos grafos sirven para aclarar los conceptos relacionados con el sistema a desarrollar, ofreciendo una visión general de las necesidades del sistema.
- Casos de uso: Muestran el contorno (actores) y el alcance del sistema (requisitos expresados como casos de uso). Un caso de uso describe la secuencia de interacciones que se producen entre el sistema y los actores del mismo para realizar una determinada función. La ventaja principal de los casos de uso es que resultan muy fáciles de entender para el cliente, sin embargo pueden carecer de la precisión necesaria, es por ello que pueden acompañarse de una información textual.
- Cuestionarios: Recoge información del sistema de forma independiente de la entrevista, mediante preguntas y respuestas concretas sobre el software.

Estas técnicas manejan la información recopilada en forma de documentos escritos. Los documentos escritos se asocian al lenguaje natural debido a que el lenguaje natural es la lengua o idioma hablado o escrito por humanos para propósitos generales de comunicación [6]. Dado lo anterior esta fase lleva consigo los problemas de ambigüedad asociados al lenguaje natural escrito.

1.1.2. Definición de requisitos

Para la definición de los requisitos, se usan lenguajes naturales que a pesar de ser ambiguos y/o extensos, su costo reducido hace que sean la alternativa común a los lenguajes formales. En esta fase es necesario elaborar el documento de requisitos, en el cual, se definen los objetivos y necesidades del sistema. Existe un conjunto de técnicas propuestas para esta actividad:

- Lenguajes naturales y formales: Los lenguajes naturales son una técnica muy ambigua en contraposición a los lenguajes formales, permite escribir y definir los requisitos de manera directa en documentos escritos.
- Ontologías: Es donde se definen los conceptos que intervienen en la definición del sistema, así como las relaciones que existen entre ellos.

- Patrones: Define los requisitos mediante el lenguaje natural pero de una manera estructurada. Esta técnica usa plantillas. Una plantilla es una tabla con una serie de campos y una estructura predefinida.

1.1.3. Validación de requisitos

En la validación de requisitos se comprueba la veracidad, consistencia y completitud de requisitos. Las técnicas existentes para desarrollar en esta fase son:

- Revisiones de verificabilidad: Consiste en la lectura y corrección de la documentación o modelado de la definición de requisitos.
- Auditorías. Consiste en un chequeo de los resultados.
- Matrices de trazabilidad: Consiste en marcar los objetivos del sistema y chequearlos contra los requisitos del mismo. Es necesario ir viendo que objetivos cubre cada requisito para detectar inconsistencias u objetivos no cubiertos.
- Prototipos: Permite que el usuario se haga una idea del sistema.

1.1.4. Problemas en la ingeniería de requisitos

La educción de requisitos está incluida en la primera fase del ciclo de vida del desarrollo de software, ésta fase permite identificar, analizar y validar los requisitos que el software futuro debe satisfacer [7]. En esta fase, se presentan problemas en la comunicación entre el analista y el interesado debido a:

- La diferencia de dominios que cada uno maneja, lo que conlleva a diferentes interpretaciones de los requisitos provocando la entrega de un producto que, en la mayoría de los casos, no cumple con las necesidades y expectativas del interesado.
- La ambigüedad semántica presente en el discurso, que dificulta la comprensión clara y correcta de los requisitos, generando inconsistencias y duplicidades entre los requisitos de software.
- La subjetividad que el analista incorpora al proceso de desarrollo de software debido a las creencias, experiencias y conocimiento que tiene del mundo, añadiendo más ambigüedades a los requisitos de software.

- La poca claridad por parte del interesado, al momento de expresar sus necesidades, planteando un documento incompleto o con más ambigüedad.

La ambigüedad presente en el discurso hace que los requisitos de software educidos, sean a su vez ambiguos. Diversos autores han tratado de contextualizar este problema y se presentan a continuación:

En [8] se presenta el lenguaje natural como el insumo más importante en el proceso de educción de requisitos, sin embargo hacen especial énfasis en que los requisitos de software son extraídos desde el lenguaje natural y por tanto tiene inmersa la ambigüedad misma del lenguaje.

En [9] se presenta el lenguaje natural como la herramienta más utilizada para la educción de requisitos, su mayor atractivo radica en su vocabulario, su informalidad y su sintaxis. Es un lenguaje familiar para el interesado y el analista, además no requiere tiempo de aprendizaje. Sin embargo se encuentra plagado de ambigüedades sintácticas, semánticas y pragmáticas.

En [10] afirman que los entregables que contienen los requisitos de software, son escritos en Lenguaje Natural (NL) y por ende sufren el problema de ambigüedad que afecta el lenguaje natural. Esto es causado por la falta de entendimiento entre las partes, analista e interesado, que genera incompletitud e inconsistencia en los requisitos de software. Sin embargo el lenguaje natural es el más entendible para los clientes ya que los requisitos de software en su mayoría son educidos desde el lenguaje natural. Finalmente afirman que los clientes están más familiarizados con el lenguaje natural para producir sus documentos de requerimientos de software.

En [11] se presenta un método para obtener requisitos desde el lenguaje natural usando la herramienta CIRCE. Los autores hacen especial énfasis, en el problema de ambigüedad semántica, presente en el lenguaje natural, heredado por ende a los requisitos de software que son educidos desde el lenguaje natural.

En contraposición al lenguaje natural, en [12] se presenta los lenguajes formales. La utilización de lenguajes formales para la educción de requisitos de una pieza de software

futura, permite eliminar por completo el problema de ambigüedad presente en el lenguaje natural, sin embargo el uso actual de los lenguajes formales para la educción de requisitos de software, no facilita la validación de los requisitos entre analista e interesado debido a que su alto grado de complejidad evita una correcta validación por parte del cliente.

Como mediación entre el lenguaje natural y los lenguajes formales, en [13] se expone un posible Lenguaje Natural Controlado (UN-Lencep), como intermediario de la comunicación entre los interesados y el analista de software para identificar los requisitos del dominio del problema. A través de este lenguaje buscan describir el discurso de un interesado que desea el desarrollo de una pieza de software que satisfaga sus problemas y necesidades. Las dificultades que presenta este lenguaje son: (i) para su utilización, se requiere entrenar previamente al interesado; y (ii) al ser un lenguaje restringido o controlado que contiene una lista de expresiones predefinidas a las cuales se debe adaptar el usuario, minimiza ostensiblemente la expresividad del interesado, situación que podría impedir que éste pueda formular todas sus necesidades y requisitos relacionados con el área del dominio del problema. Adicionalmente aunque el lenguaje natural controlado elimina las ambigüedades sintácticas (ambigüedades de forma de la frase), no elimina la ambigüedad semántica del discurso, debido a que solo restringe la (Sintaxis) pero no su significado (Semántica).

En laTabla 1-1 se resumen los enfoques de los tipos de ambigüedades que afectan los requisitos de software y el método que se propone para atacar la ambigüedad.

Tabla 1-1 Autores que exponen los tipos de ambigüedad presente en los requisitos de software y su propuesta para evitarla.

Autor	Tipo de Ambigüedad	Propuesta
[8]	Semántica, Sintáctica	Ninguna
[9]	Semántica	Lineamientos manuales seguidos por el analista con el fin de evitar al máximo ambigüedades.

[10]	Semántica	Técnicas de desambiguación del sentido de las palabras, lexicones y diccionarios
[11]	Semántica	Lineamiento para evitar la ambigüedad semántica.
[12]	Semántica, Sintáctica	Uso de lenguajes formales
[13]	Sintáctica	Lenguajes controlados.

1.2. Arquitecturas de educción de requisitos

Las arquitecturas de educción de requisitos son modelos que permiten transformar especificaciones de requisitos de software en lenguaje natural o controlado a modelos conceptuales de manera manual, semi-automática o automática conceptuales. Con el fin de realizar un proceso completo y eficaz de levantamiento de requisitos, por ejemplo un sistema que realice la transformación de requisitos de software desde el lenguaje natural hasta el modelo de KAOS [14], el esquema preconceptual [13], el modelo de casos de usos o cualquier diagrama UML en general [15],es considerado una arquitectura de educción de requisitos de software.

Como marco de la presente tesis a continuación se presentan el modelo de KAOS, ya que es el modelo de referencia usado en esta tesis para realizar los procesos semánticos de desambiguación y detección de errores.

1.2.1. Modelo de KAOS

KAOS es una metodología para la ingeniería de requisitos, que permite al analista construir modelos de requisitos. Además, facilita la identificación de propiedades de interés y diseños alternativos inesperados. Este enfoque puede ser usado para cualquier sistema de información según [16].

La ventaja principal de esta metodología es la capacidad de alinear los requisitos a objetivos y metas de la organización. Esta alineación aumenta las posibilidades de que nuevos desarrollos le agreguen valor al negocio [17].

En esencia, la metodología KAOS, es un enfoque de la ingeniería de requisitos, orientada a objetivos, que tiene un conjunto de técnicas de análisis formal. En [18] se muestra que este enfoque presenta varios niveles de expresividad y razonamiento:

- Semi-formal para modelar y estructurar objetivos,
- Cualitativo para la selección entre las alternativas,
- Formal, si es necesario, para un razonamiento más preciso.

Cada elemento en el lenguaje de modelado de KAOS, tiene una estructura de dos niveles: la capa exterior semántica y gráfica, donde el concepto se declara junto con sus atributos y relaciones, con otros conceptos y la capa interna, para la definición formal del concepto como se analiza en [19]. En la figura siguiente se muestra un ejemplo del modelo de KAOS.

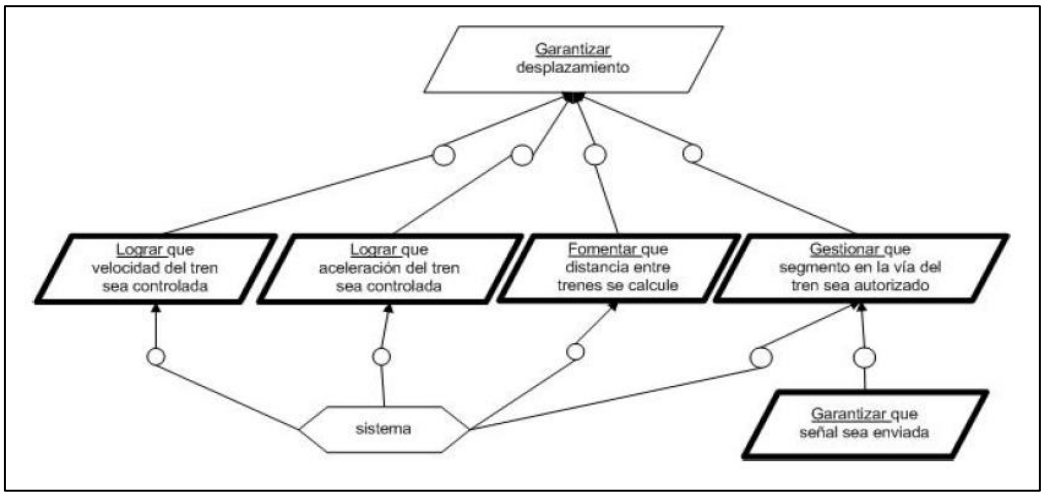


Figura 1.1 *Ejemplo modelo de objetivos de KAOS.*
Fuente: Adaptado de **[16]**

En [20] se propone una metodología para representar los elementos del modelo de KAOS, mediante un diagrama de clases que representa el modelo de KAOS. Una porción de este diagrama de clases se muestra en la Figura 1.2.

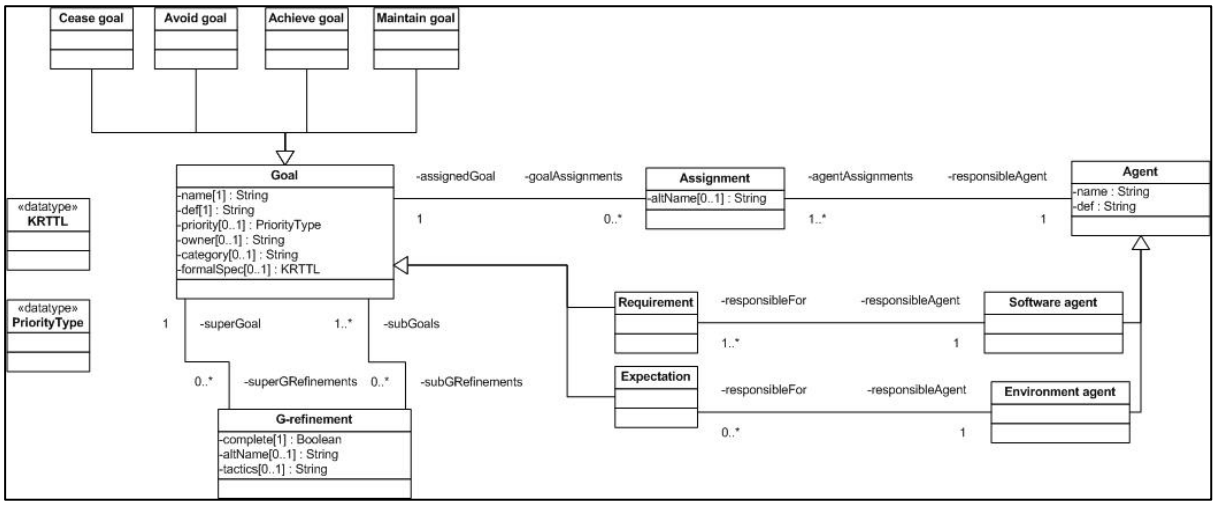


Figura 1.2: *Porción del Meta-Modelo de KAOS*
Fuente: Adaptado de **[20]**

Definiciones asociadas:

- **Definición 1.1**: Una arquitectura de educción de requisitos es un modelo que permite transformar especificaciones de requisitos de software en lenguaje natural o controlado a modelos conceptuales de manera manual, semi-automática o automática conceptuales.
- **Definición 1.2.** Un elemento constitutivo del modelo de requisitos (*en adelante elemento constitutivo*) es un elemento que contiene información acerca de las propiedades o requisitos del dominio del software, estos elementos se encuentran definidos en lenguaje natural a modo de frases o palabras y son propensos a los mismos tipos de ambigüedades que sufre el lenguaje natural. Por ejemplo agentes, objetivos, entidades, operaciones, etc.

En la siguiente figura se ilustran las definiciones anteriores.

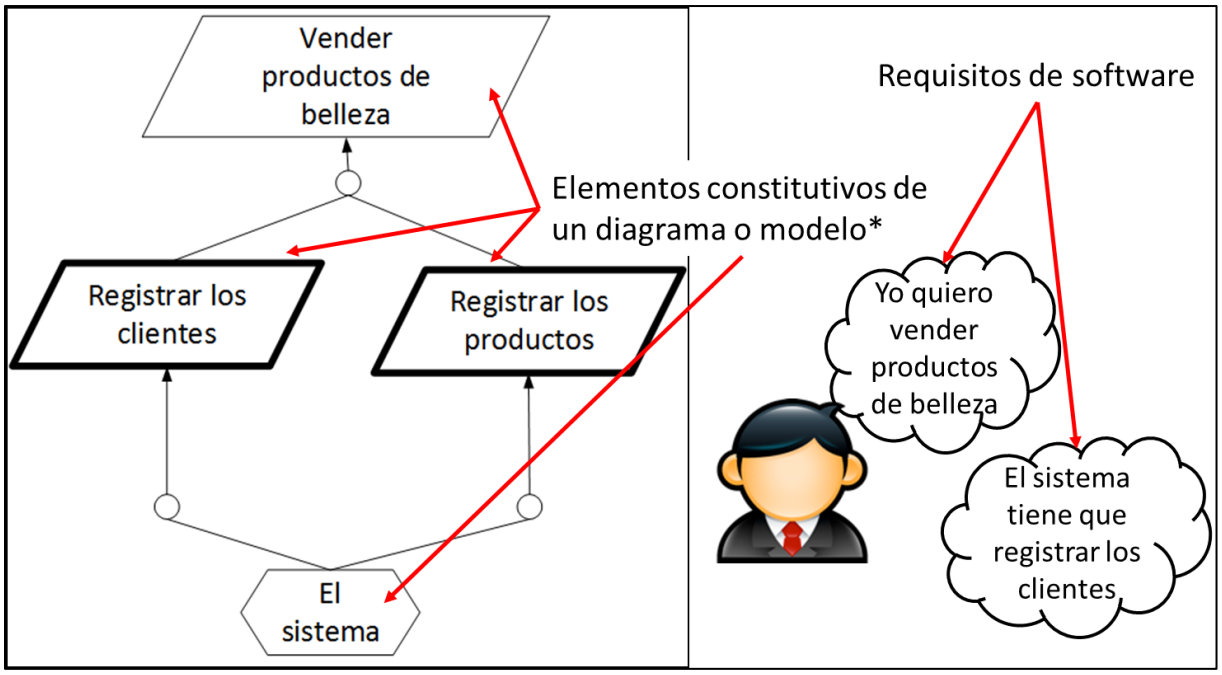


Figura 1.3 *Ejemplo arquitectura de educción de requisitos y sus respectivos elementos constitutivos.*
Fuente: Elaboración propia

1.3. Ambigüedad Semántica

El procesamiento del lenguaje natural hace parte de la rama de la inteligencia artificial y de la lingüística computacional, las cuales son ramas de la ciencia encargadas de estudiar y entender el lenguaje humano mismo, orientado a las máquinas. Particularmente son las ramas de la ciencia, encargadas de representar para las máquinas, el lenguaje humano.

El lenguaje natural está plagado de ambigüedades (léxica, sintáctica, semántica, fonética), muchas de estas ocasionadas por la vaguedad misma del lenguaje y la poca precisión con que es usado cotidianamente , esto es agravado con la gran cantidad de sentidos posibles asociados a una sola palabra en un idioma específico [21]. A continuación se describen algunos de los trabajos que hacen referencia a la ambigüedad en el lenguaje natural:

En [22] se definen tres tipos diferentes de ambigüedades en algún idioma:

- Ambigüedad léxica: Una misma palabra puede pertenecer a diferentes categorías gramaticales.
- Ambigüedad sintáctica: Aparece cuando debido a la forma en que se asocian los distintos constituyentes de una frase, puede ser interpretada de varias formas.
- Ambigüedad semántica: Dentro de este tipo de ambigüedad, se pueden diferenciar tres clases:
    - Ambigüedad debida a las palabras polisémicas, en este caso, una misma palabra puede tener distintos significados dependiendo del uso que se le esté dando en cada momento.
    - Ambigüedad debida al encontrar una misma estructura sintáctica con diferentes significados.
    - Ambigüedad referencial. En éste caso, es necesario el análisis del texto, más allá de los límites de la frase, determinando los antecedentes referenciales de los pronombres.

En [23] se presentan tres tipos de ambigüedad para el lenguaje natural escrito:

- Ambigüedad Léxica.
- Ambigüedad Semántica.
- Ambigüedad Sintáctica.

Definiendo y tratando esta última como la ambigüedad que ocurre cuando la información sintáctica no es suficiente para hacer una decisión de asignación de estructura, es decir cuando una frase puede tener más de una estructura (árbol) sintáctica. La resolución de la ambigüedad sintáctica debe basarse en modelos de conocimiento diverso, como patrones de manejo, reglas ponderadas, proximidad semántica, entre otras.

En [24] se presenta tres tipos de ambigüedades en el lenguaje:

- Ambigüedad semántica, la cual se refiere al significado de los predicados
- Ambigüedad Sintáctica: Se refiere a la estructura lógica de los predicados.
- Ambigüedad Pragmática: Se refiere a la forma en que es utilizado el lenguaje en un contexto dado.

En [25] se presentan solo dos tipos de ambigüedad léxica:

- Ambigüedad Sintáctica, la cual se refiere a la posible asignación de diferentes categorías sintácticas a cada palabra.
- Ambigüedad Semántica, la cual se refiere a la posible asignación de diferentes significados a cada palabra.

En [26] se presenta dos clases de ambigüedades:

- Ambigüedad léxica categorial, en la cual la palabra tiene diversos significados aunque puede cumplir varias clases de categorías gramaticales (puede ser sujeto, verbo, adjetivo, etc.).
- Ambigüedad léxica semántica, en la que la palabra puede tomar diferentes sentidos dentro de un mismo contexto

Siendo esta última, una de las principales falencias en la educción de requisitos. Así mismo, indica que existen dos métodos para la desambiguación de las palabras, los métodos basados en conocimiento y los métodos basados en Corpus. Los primeros fundamentados en grandes diccionarios o lexicones y los segundos en Corpus, generados mediante modelos estadísticos. El autor propone un método supervisado y a su vez reitera que siendo bueno, aún se puede mejorar. Señala que no existe ninguna metodología para desambiguación de los sentidos de las palabras que logre resultados exitosos al 100%.

En la Tabla 1-2 se resumen los enfoques de los autores y los diferentes tipos de ambigüedad, en el lenguaje que ellos identificaron.

Tabla 1-2 Autores que exponen los tipos de ambigüedad presente en el lenguaje.

Autor	Ambigüedad Léxica	Ambigüedad Sintáctica / Estructural	Ambigüedad Semántica	Ambigüedad Pragmática
[22]	Si	Si	Énfasis	No
[23]	Si	Énfasis	Si	No
[24]	No	Si	Si	Si
[25]	Si	Si	Si	No
[26]	Si	Si	Énfasis	No

El lenguaje natural escrito es menos susceptible a los problemas de ambigüedad, debido a que posee reglas que lo hacen más estricto que el lenguaje natural hablado. Sin embargo el lenguaje natural escrito es también rico en ambigüedades y una de ellas es la ambigüedad polisémica.

Una palabra que posee más de un sentido posible se le denomina palabra polisémica, es decir es una palabra que tiene muchos(poli) sentidos(sémica) pero que bajo un contexto determinado es posible intuir el sentido más correcto, aunque en muchos casos es incluso difícil para el humano mismo descubrir el sentido más correcto [27].

La ambigüedad polisémica es un tipo de ambigüedad semántica que consiste en la incertidumbre del sentido más correcto de una palabra en una frase o en un documento. Los tipos de palabras que son más susceptibles a este tipo de ambigüedad son verbos, adjetivos y sustantivos.

1.3.1. Procesamiento del lenguaje natural

El procesamiento del lenguaje natural es la tarea de procesar el lenguaje para obtener información valiosa a partir de agrupaciones de términos (generalmente textos). El procesamiento del lenguaje natural puede incluir diversas fases del tratamiento de la información tales como filtrado, cuantificación, cualificación e incluso el uso de métodos estadísticos que permitan procesar mejor un texto [28], [29].

En el marco de la presente tesis, se considerará procesamiento del lenguaje natural, como la tarea de procesar el lenguaje natural escrito considerando como entrada, un texto en lenguaje natural y como salida, una estructura que contiene las características morfológicas, sintácticas y semánticas, asociadas a cada palabra que compone el texto. En la siguiente figura 1.6, se ilustra este proceso.

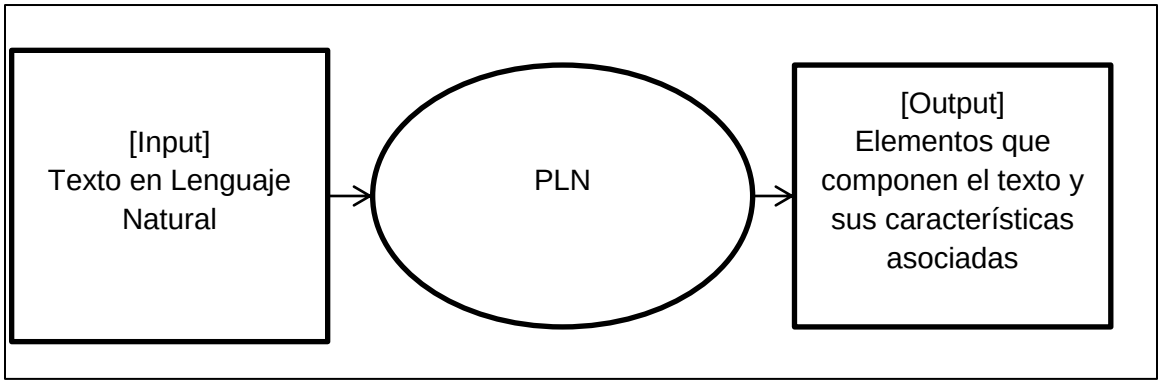


Figura 1.4 *Descripción general del procesamiento del lenguaje natural. Fuente: Elaboración Propia.*

Toda tarea de procesamiento del lenguaje natural, tiene unos procesos morfosintácticos iniciales de tratamiento de textos, que deben ser realizados antes de realizar un proceso semántico. Esto se debe a que computacionalmente un texto es una fuente de información no estructurada y los procesos morfosintácticos permiten estructurar el texto para que sea una entrada computacionalmente válida. Adicionalmente estos procesos morfosintácticos permiten identificar las palabras vacías (Palabras que carecen de significado como determinantes y artículos) y por tanto palabras que no son importantes en un proceso semántico. Así mismo determinan la categoría gramatical de cada palabra, que es un insumo necesario para la validación de los elementos constitutivos mediante reglas y finalmente estos procesos morfosintácticos eliminan también en gran medida, las ambigüedades de tipo morfológico y sintáctico con una precisión de más del 70%, usando bases estructuradas de conocimiento léxico [29].

Para adecuar un texto como entrada de cualquier método o proceso de desambiguación semántica, se debe transformar en un recurso estructurado [29]. Usualmente el proceso de transformación de un texto en una fuente estructurada consiste en los siguientes pasos:

- Segmentación por frases (Tokenization): Este proceso consiste en dividir el texto en un conjunto de frases de acuerdo a su puntuación.
- Segmentación por palabras: Este proceso consiste en dividir cada frase en las palabras que la componen.

- Etiquetado (Tagging): Consiste en la asignación de una categoría gramatical a cada palabra. (Ej.: El/Det Carro/NN esta/VR Rojo/AD). Este proceso elimina la ambigüedad morfológica de la frase. (Las categorías gramaticales son clases de palabras, según la función que desempeñan en la frase. Concretamente las categorías gramaticales para español son: Sustantivos, adjetivos, pronombres, verbos, adverbios, preposiciones, conjunción e interjección.)
- Lematización (Lemmatization): Donde las palabras son reducidas a su forma básica terminológica, (Ej: Corrio→Correr, Fue→Ser).
- Fragmentación (Chunking): Consiste en dividir el texto en dependencias sintácticas, una dependencia sintáctica es la agrupación de una o más categorías gramaticales de acuerdo a su función en la frase, como sintagmas nominales o perífrasis verbales.

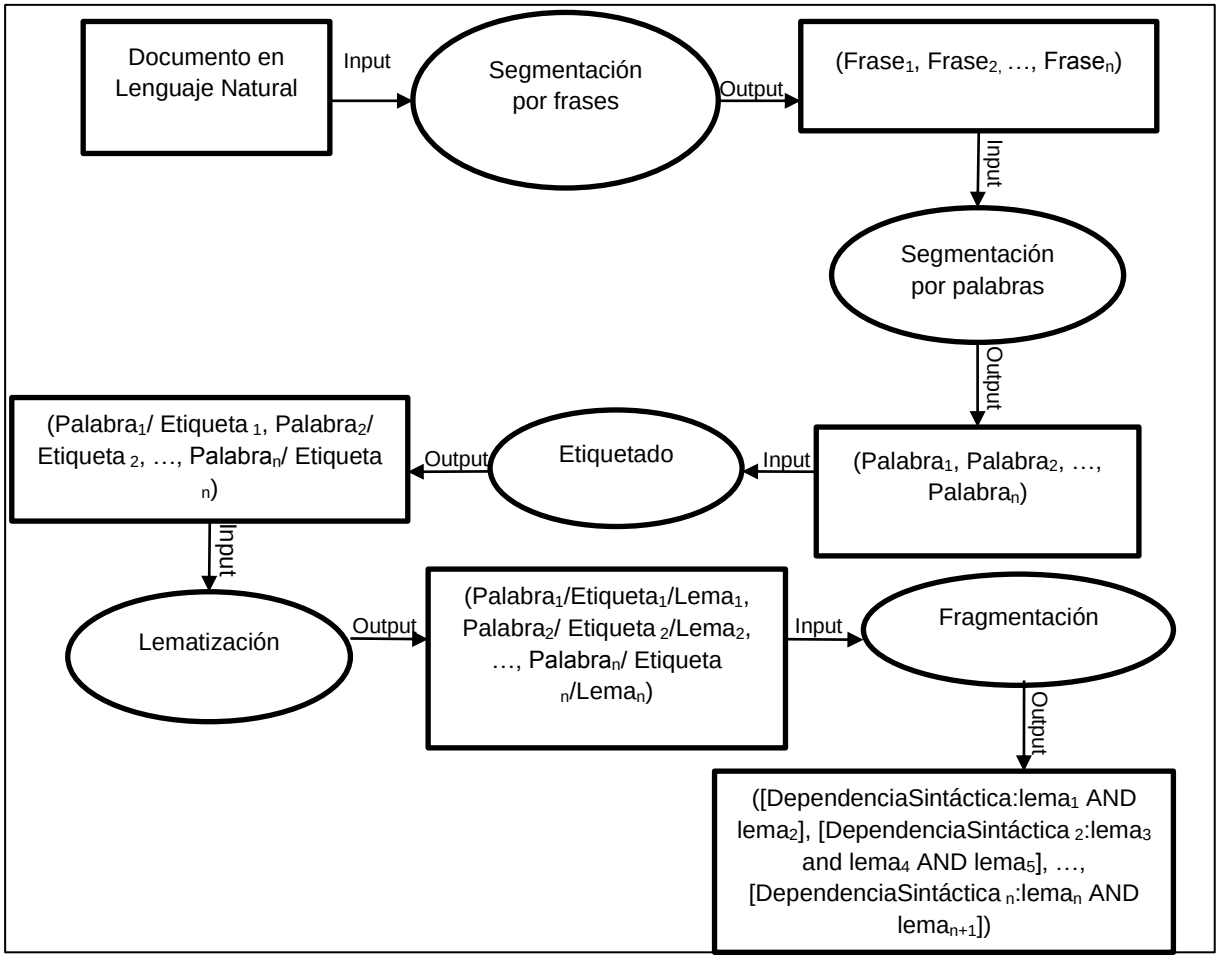
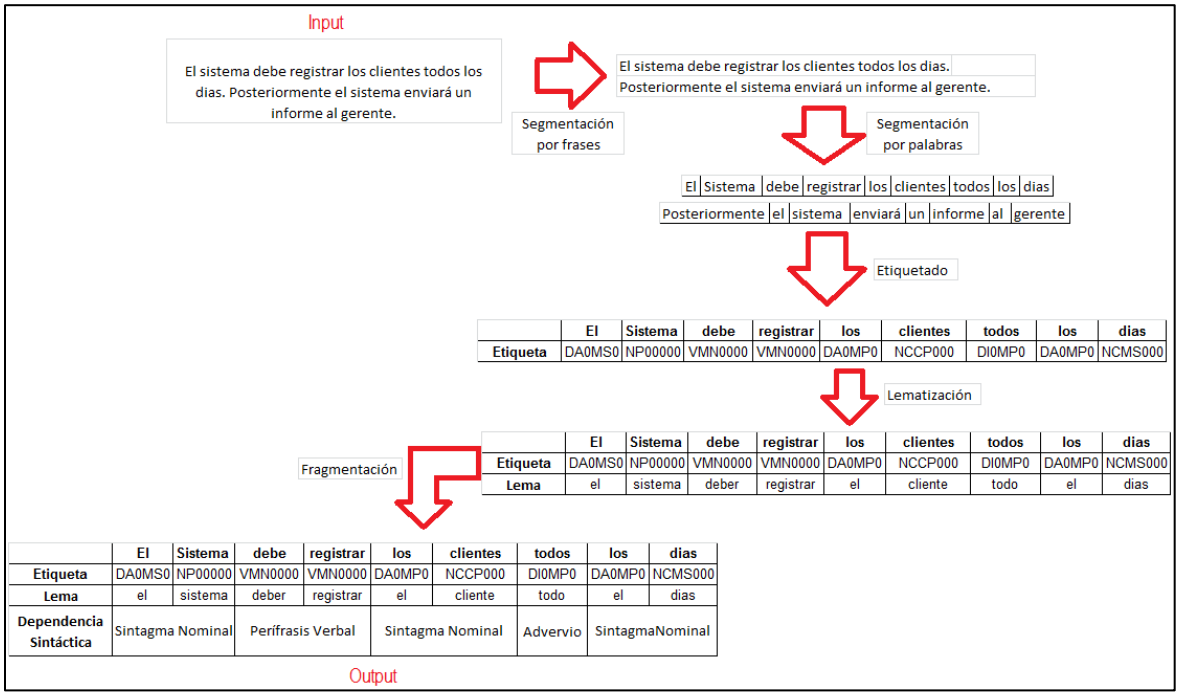


Figura 1.5 *Procesos morfológicos y sintácticos básicos de tratamiento de textos. Fuente: Elaboración Propia.*

Como resultado de éstos pasos, se obtiene un vector de características para cada palabra presente en el documento, luego de éste proceso, cada palabra se encuentra lista para ser desambiguada por algún método de desambiguación propuesto en las siguientes secciones. A continuación se presenta un ejemplo del procesamiento del lenguaje natural, aplicado a un texto que contiene requisitos de software.



	El	Sistema	debe	registrar	los	clientes	todos	los	dias
Etiqueta	DA0MS0	NP00000	VMN0000	VMN0000	DA0MP0	NCCP000	DI0MP0	DA0MP0	NCMS000

	El	Sistema	debe	registrar	los	clientes	todos	los	dias
Etiqueta	DA0MS0	NP00000	VMN0000	VMN0000	DA0MP0	NCCP000	DI0MP0	DA0MP0	NCMS000
Lema	el	sistema	deber	registrar	el	cliente	todo	el	dias

	El	Sistema	debe	registrar	los	clientes	todos	los	dias
Etiqueta	DA0MS0	NP00000	VMN0000	VMN0000	DA0MP0	NCCP000	DI0MP0	DA0MP0	NCMS000
Lema	el	sistema	deber	registrar	el	cliente	todo	el	dias
Dependencia Sintáctica	Sintagma Nominal		Perífrasis Verbal		Sintagma Nominal		Advervio	SintagmaNominal	

Figura 1.6 *Ejemplo tareas de procesamiento morfosintáctico aplicadas a un requisito de software.*
Fuente: Elaboración Propia.

1.3.2. Sentido de las palabras.

Una palabra polisémica cumple una simple estructura computacional básica que puede ser vista como una palabra <palabra> que apunta a 1… <n> sentidos (<sentidos>), pero a su vez un sentido (<sentido>) puede ser apuntado por 1…. <n> palabras (<palabra>), por tanto el problema de desambiguación de este tipo de ambigüedad, se remite a seleccionar el mejor <sentido> correspondiente a una <palabra> en un contexto dado [30].

Un sentido es una definición no ambigua de una abstracción en el mundo real y normalmente como puede ser identificada por muchas etiquetas (palabras), tiene

asociado un único número conocido y aceptado mundialmente por las agrupaciones académicas que trabajan en el área [31].

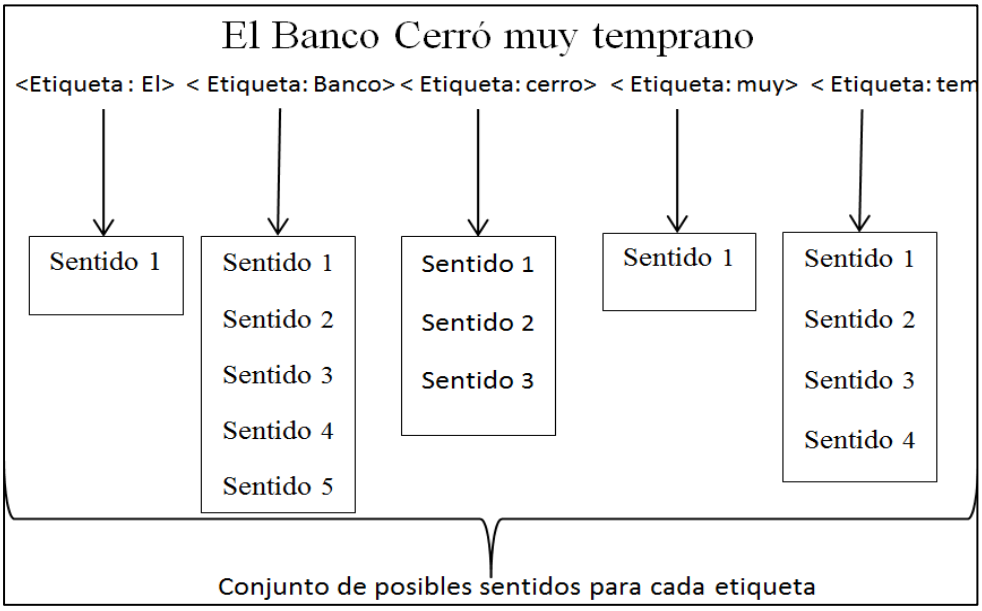


Figura 1.7: *Ejemplo estructura de una frase con palabras polisémicas*
Fuente: Elaboración Propia

Los sentidos son representados de acuerdo al estándar ISO para la formación de lexicones y diccionarios computacionales [32]. La representación de un sentido está dada por tres letras mayúsculas al inicio, que representa el idioma SPA, ENG, POR etc., sin embargo se usa más las letras ILI que significa que el concepto es el mismo, independiente el idioma, posteriormente a las tres (3) letras iniciales, el sentido se compone un numero entero, entre 4 a 11 dígitos y finalmente el sentido posee la inicial del tipo de palabra, Verbo V, Nombre N, Adjetivo A,… etc. Un ejemplo de un sentido es: ILI-8881283-V que representa que el sentido hace referencia a un verbo y que es transversal a todos los idiomas.

- **Definición 1.3.** Los sentidos de las palabras se asocian solo a palabras funcionales. Formalmente se defina una palabra funcional, como aquella que aporta contenido semántico a la frase a la que pertenece. Concretamente bajo el marco de esta tesis, las palabras funcionales son Verbos, Sustantivos, Adjetivos y Adverbios. No se consideran palabras funcionales, preposiciones ni determinantes.

1.3.3. Desambiguación del sentido de las palabras

Desambiguación del sentido de las palabras (WSD, por sus siglas en inglés) es la habilidad de determinar computacionalmente, cuál significado de una palabra debe ser usado en un contexto particular dado. Es posible describir formalmente WSD, como la tarea de asignar los sentidos apropiados a un conjunto de palabras (etiquetas) W, en un texto T. Es decir un mapeo A de las palabras W, a los sentidos S, tal que A(i) ⊆ SentidosD (Wi), donde SentidosD(Wi), es el conjunto de sentidos contenidos en un diccionario D para una palabra Wi y A(i), es el subconjunto de los sentidos de Wi, que son apropiados en el contexto T, aunque típicamente el mejor sentido posible es aquel en el que |A(i)|=1.

WSD también puede ser visto como una tarea de clasificación en el que los sentidos de las palabras son clases y un método de clasificación automático seria signar para cada palabras W,|, una o más clases sentidos basados en la evidencia del contexto T y fuentes externas de conocimiento [33].

La tarea de desambiguación del sentido de las palabras WSD tiene dos variantes.

- Por ejemplo léxico: Donde el sistema debe desambiguar un conjunto restringido de palabras, usualmente uno por frase, esta variante se utiliza principalmente para métodos supervisados, en el cual se utiliza todo el contexto como corpus entrenado y se prueba con una palabra por frase.
- Todas las palabras: Donde el sistema debe desambiguar todas las clases de palabras en el texto (sustantivos, adjetivos, verbos y adverbios). Esta variante utiliza métodos basados en conocimiento.

El conocimiento es un componente fundamental de WSD, las bases de conocimiento proveen información que es importante para asociar un sentido con las palabras que lo describen, estas fuentes de conocimiento pueden variar desde corpus o corporas de texto, (corpus/corpora representa un conjunto amplio y estructurado de textos) hasta diccionarios, tesauros, glosarios y ontologías. Estas bases de conocimiento se utilizan para hacer el análisis estadístico y la prueba de hipótesis, (comprobación de ocurrencias

o validación de reglas lingüísticas dentro de un territorio específico del lenguaje). Estas fuentes se pueden dividir entre fuentes estructuradas y no estructuradas.

- Algunas fuentes estructuradas: Tesauros, Diccionarios leíbles por máquinas y ontologías.
- Algunas fuentes no estructuradas: Corpora, Listas de sentidos frecuentes, Corporas anotados, Corporas RAW.

Los métodos de desambiguación de sentidos de las palabras se pueden clasificar en tres tipos como se puede observar en [29].

- Supervisados
- No supervisados
- Basados en Conocimiento

1.3.3.1. Supervisados

Los métodos de desambiguación supervisados, utilizan técnicas de aprendizaje de máquina, con el fin de aprender e inferir reglas a partir de corpus ya entrenados (los corpus significan textos etiquetados y desambiguados previamente), con el fin de procesar y desambiguar textos nuevos basados en las reglas inferidas.

En [34] se presenta las listas de decisión como un método que tiene un conjunto ordenado de reglas para categorizar los sentidos de las palabras de acuerdo a sus características, este método pretende entrenar una máquina con base a las reglas suministradas, para luego realizar la prueba con el texto a desambiguar en cuestión.

Los árboles de decisión presentados en [35], son un modelo predictivo el cual pretende representar las reglas de clasificación de sentidos, en manera de árbol, ésto se debe a que los arboles de decisión pueden ser recorridos recursivamente, indagando en profundidad el mejor sentido, hasta llegar a la mejor hoja posible, de acuerdo a unas características dadas.

Las redes bayesianas son un método simple de clasificación probabilística, basadas en una aplicación del teorema de bayes, se fundamentan en el cálculo de una condición de probabilidad para cada sentido S, en una palabra W, dadas unas características f. Este método fue propuesto en [36].

El concepto de redes neuronales aplicado al proceso de desambiguación semántica fue propuesto en [37]. Éste enfoque define las palabras, como neuronas las cuales se encuentran relacionadas entre sí de acuerdo a sus características semánticas. Una red neuronal, es un conjunto interconectado de neuronas que usan un modelo computacional para procesar información basado en un enfoque conexionista

La técnica de aprendizaje basada en ejemplos, implementa un mecanismo de clasificación de los sentidos, basados en ejemplos. La técnica retiene estos ejemplos en memoria como puntos de referencia, y una vez que obtiene un nuevo texto a desambiguar trata de intuir el mejor sentido de acuerdo a los ejemplos retenidos en memoria, esta técnica fue propuesta en [38].

Las máquinas de soporte vectorial [39], se encuentran basadas en la idea de aprender de manera lineal, a partir de un conjunto de ejemplos positivos y negativos situados en un plano, con el fin de minimizar posteriormente, la distancia a los ejemplos positivos y maximizar la distancia a los ejemplos negativos, de esta manera se obtendrá una mejor desambiguación del sentido de las palabras.

1.3.3.2. No supervisados

Estos métodos de desambiguación utilizan tan solo el contexto de una palabra, con el fin de desambiguar la palabra en cuestión. Se basan en la idea de que el mismo sentido de una palabra, tendrá palabras vecinas similares.

En [40] presentan la técnica del agrupamiento por contextos (Context clustering), la cual consiste en la idea de representar cada palabra dentro de un texto como un vector, estos vectores contienen características y pueden ser agrupados, de acuerdo características similares, a partir de ahí la técnica puede intuir probabilísticamente, cuáles palabras van agrupadas, generalmente bajo un contexto dado.

En [41] se presenta la técnica de agrupamiento por palabras (Word clustering), la cual consiste en el agrupamiento de palabras como vectores de características, pero de acuerdo a la medida de similitud semántica específica, esto conlleva a determinar el mejor sentido para una palabra de acuerdo a la similitud semántica que tenga con las demás palabras.

En [42] se presenta la técnica de grafos de concurrencia (concurrence graphs), esta técnica permite elegir el mejor sentido de acuerdo a la distancia teórica entre dos palabras, es decir, un texto puede ser representado como una grafo que está compuesto de nodos palabras interconectados entre sí de acuerdo a sus relaciones gramaticales (sintagmas, perífrasis etc.), de allí es posible determinar el mejor sentido de una palabra de acuerdo a la distancia respecto a otra en el grafo (texto).

1.3.3.3. Basados en conocimiento

En [43] se presenta Lesk simplificado, esta es una técnica de desambiguación, basada en conocimiento, la cual parte del concepto que las palabras de un vecindario de una porción de texto, tenderán a compartir un tema en común y por tanto, es posible identificar el sentido más correcto, basadas en las definiciones del diccionario. Una versión simplificada del algoritmo de Lesk, es comparar la definición o los ejemplos del diccionario de una palabra ambigua, con los demás términos contenidos en su vecindario.

En [44], proponen modificar el algoritmo original Lesk y crear la técnica de Lesk adaptada, debido a que presenta limitaciones al solo comparar los sinónimos de la palabra. Concretamente proponen aumentar la comparación con otros tipos de relaciones semánticas tales como hiperónimos (palabras más generales) e hipónimos (palabras más específicas) entre otras.

La propuesta consiste en tomar las palabras del contexto y comparar por pares. Sin embargo, no se realiza sobre las etiquetas del vecindario, sino que se compara en las definiciones de los sinónimos, hiperónimos, hipónimos, holónimos, merónimos, tropónimos y relaciones de atributo, para cada palabra en el vecindario.

El sentido más frecuente (o MFS, por sus siglas en inglés), es un algoritmo de desambiguación semántica polisémica, que permite encontrar el sentido mayormente utilizado en un corpus, para cada palabra. Este método, parte de un corpus anotado con sentidos, del cual se aprenden las frecuencias de sentidos individuales. Para cada palabra a desambiguar, un etiquetador morfosintáctico se usa para determinar la etiqueta PoS (Part-of-Speech), y luego se selecciona el sentido para esa etiqueta.

Aunque es un método sencillo, ha mostrado ser difícil de superar, debido a que presenta un desempeño de 62,5%. De hecho, tan sólo 5 de los 26 sistemas enviados al *Senseval-3,* han logrado mejores resultados [45].

La distancia conceptual, es una técnica de desambiguación, basada en conocimiento que mide la cercanía en el significado entre palabras en una frase, tomando como referencia una red jerárquica estructurada [46], esto permite realizar la desambiguación completa de la frase, al obtener los sentidos con la menor distancia conceptual entre sí.

El UKB [47] es una implementación de un método de desambiguación semántica basado en grafos, realizada por el grupo IXA de la Universidad del País Vasco. El método consiste en realizar recorridos sobre el grafo de cualquier base de conocimientos léxica (LKB) como WordNet. El mejor sentido posible, será determinado por el camino más corto que une todo los conceptos del contexto. El puntaje se determina con base en la adaptación que hace [48] al PageRank.

En la Tabla 1-3 se resumen todos los enfoques para resolver la ambigüedad del sentido de las palabras, junto con la técnica utilizada. Se citan además otros autores que trabajan sobre la misma técnica y el mismo enfoque.

Tabla 1-3 Resumen de los enfoques y técnicas utilizadas para atacar la ambigüedad del sentid de las palabras.

Autores	Técnica	Enfoque
[34]	Listas de decisión	Supervisado
[35]	Arboles de decisión	Supervisado
[36]	Redes Bayesianas	Supervisado
[37]	Redes Neuronales	Supervisado

[38]	Máquinas Vectoriales	Supervisado
[40]	Agrupamiento por contexto	No Supervisado
[41]	Agrupamiento por palabra	No Supervisado
[42]	Grafos de concurrencia	No Supervisado
[43]	Simplified Lesk	Basado en Conocimiento
[44]	Adapted Lesk	Basado en Conocimiento
[45]	MFS	Basado en Conocimiento
[46]	Distancia conceptual	Basado en Conocimiento
[47]	UKB	Basado en Conocimiento

1.4. Métricas de validación

1.4.1. Métricas de validación en la desambiguación de sentidos de las palabras

Las métricas de validación de las tareas de desambiguación, son usualmente creadas en términos de la evaluación, según [29], [49], [50], [51] y [52]. Las métricas utilizadas son: Precisión, Recuerdo y Cobertura. En la descripción de estas tres métricas, T, es la cantidad total de ejemplos del conjunto. V, es el número de ejemplos correctamente desambiguados y U, es el número de ejemplos desambiguados, cuya lista de sentidos probables, no está vacía (sin importar si están correctamente desambiguado o no), sin embargo, evaluar estas métricas por separado, no tendría ningún sentido, es por esto que finalmente los autores proponen una métrica que integre estas métricas en una sola y la denomina “Métrica F-1”, la cual determina armónicamente la precisión y el recuerdo, con el fin de determinar cuantitativamente, cuál método o técnica es mejor que otro, a continuación se especifican las ecuaciones y su respectiva explicación.

Tabla 1-4. Métricas utilizadas para la evaluación de los resultados en la desambiguación de los sentidos de las palabras en los elementos constitutivos.

Métricas	Ecuación	Definición
Cobertura (C_{wsd})	$C = \frac{U}{T}$	La cobertura es el porcentaje de elementos constitutivos a los que el sistema de

		desambiguación ha dado respuesta (número de instancias a las cuales el sistema de WSD, propuso una respuesta / número total de elementos constitutivos en el conjunto de prueba)
Recuerdo (R_{wsd})	$V = \frac{V}{T}$	El recuerdo es el porcentaje de elementos constitutivos que han sido correctamente desambiguadas dentro del conjunto de todos los elementos constitutivos introducidos (número de instancias correctamente desambiguadas / número total de instancias en el conjunto de prueba)
Precisión (P_{wsd})	$P = \frac{V}{U}$	La precisión es el porcentaje de elementos constitutivos, correctamente desambiguados por el sistema de WSD (número de instancias correctamente desambiguadas por el sistema WSD / número de instancias a las cuales el sistema de WSD propuso una respuesta).
$F1_{wsd}$	$F1 = \frac{2 * P * R}{P + R}$	El coeficiente F1, es un valor entre 0 y 1 que indica la efectividad de la tarea, siendo 1, el sistema perfecto. Esta métrica agrupa la precisión y

		el recuerdo con el fin de determinar, cómo fue el proceso general del modelo.
Rendimiento$_{wsd}$	$\gamma = \frac{T}{tiempo\ de\ respuesta\ (Segundos)}$	Tiempo de ejecución de la tarea encomendada en segundos.

1.4.2. Métricas de evaluación en la arquitectura de educción de requisitos.

Análogamente que para evaluar la desambiguación del sentido de las palabras, bajo el marco de la presente tesis se exponen las siguientes métricas relacionadas con la inconsistencia, duplicidad y no unificada en la arquitectura de educción de requisitos de software. Estas métricas son presentadas en la siguiente Tabla 1-5.

Tabla 1-5. Métricas de evaluación en la arquitectura de educción de requisitos.

Métricas	Ecuación	Definición
Precisión de la detección de inconsistencias	$PI_A = \frac{A_I}{A_T}$	La precisión es el porcentaje de elementos constitutivos correctamente identificados como inconsistentes por el sistema A_I, versus el número total de elementos constitutivos inconsistentes hallados por el sistema A_T.
Recuerdo de la detección de inconsistencias	$RI_A = \frac{A_I}{A_T + A_N}$	El recuerdo es el porcentaje de elementos constitutivos correctamente identificados como inconsistentes por el sistema A_I, versus el número total de elementos constitutivos inconsistentes hallados por el sistema A_T más el número total de elementos constitutivos inconsistentes no hallados por el sistema A_N.
Precisión de la detección de duplicidades	$PD_A = \frac{A_D}{A_{TD}}$	La precisión es el porcentaje de elementos constitutivos correctamente identificados, como Duplicados por el sistema A_D, versus el número total de elementos constitutivos duplicados hallados por el sistema A_{TD}.

Recuerdo de la detección de duplicidades	$RD_A = \frac{A_D}{A_{TD} + A_{ND}}$	El recuerdo es el porcentaje de elementos constitutivos correctamente identificados, como duplicados por el sistema A_D, versus el número total de elementos constitutivos duplicado, hallados por el sistema A_{TD}, más el número total de elementos constitutivos duplicados no hallados por el sistema A_{ND}.
Precisión de la detección de no unificidades	$PU_A = \frac{A_U}{A_{TU}}$	La precisión, es el porcentaje de elementos constitutivos correctamente identificados como no unificados por el sistema A_U, versus el número total de elementos constitutivos no unificados hallados por el sistema A_{TU}.
Recuerdo de la detección de no unificidades	$RU_A = \frac{A_U}{A_{TU} + A_{NU}}$	El recuerdo es el porcentaje de elementos constitutivos correctamente identificados como no unificados por el sistema A_U, versus el número total de elementos constitutivos inconsistentes hallados por el sistema A_{TU}, más el número total de elementos constitutivos inconsistentes no hallados por el sistema A_{NU}.

1.5. Comparación con enfoques actuales

Los trabajos relacionados se revisaron de acuerdo al criterio base asociado a la desambiguación de requisitos de software de manera automática. En la Tabla 1-6, se resumen los resultados con el fin de evitar la ambigüedad semántica en los requisitos de software.

En [11], presentan un modelo para la generación automática de modelos de requisitos de software utilizando herramientas morfosintácticas para analizar los requisitos de software, sin embargo el autor no hace referencia a la ambigüedad semántica, dado que simplemente se predefine cada término utilizado en un glosario de términos, por otro lado trata de formular los requisitos desde la perspectiva morfosintáctica y formal del lenguaje,

lo cual conlleva a que se remuevan las ambigüedades de tipo sintáctica y morfológica, finalmente estos requisitos son expresados en diagramas UML.

En [53], presentan un novedoso método para identificar la ambigüedad estructural y sintáctica en los requisitos de software, utilizando una técnica de aprendizaje de máquina, el autor entrena directamente diversos corpus con el fin de inducir al algoritmo, a que determine por ejemplo inducido, las ambigüedades en frases, que hacen parte de un requisito de software. Sin embargo el autor, no menciona la ambigüedad semántica presente en los requisitos de software, ni propone un método para atacar la ambigüedad, simplemente basa su trabajo en la detección de ambigüedades sintácticas en los requisitos de software.

En [54] se presenta un método manual para la extracción y validación de requisitos de software, usando técnicas de procesamiento de lenguaje natural como parseo, etiquetado y análisis de requisitos orientados a objetos, sin embargo no se menciona la calidad semántica de estos requisitos, ni se menciona el problema de ambigüedad semántica inherente a los requisitos de software extraídos del lenguaje natural.

En [55] y [56], se presenta un método para la obtención de requisitos de software y de reglas del negocio, utilizando lenguaje natural controlado, esto con el fin de evitar las ambigüedades de tipo léxico y en algunos casos de tipo semántico, ya que los autores realizan un marcado semántico de los requisitos, usando ontologías de dominios como glosarios, con el fin de tener precargado y predefinido la semántica y el sentido de cada palabra que será usado en el modelo. Esta ontología de dominio, se instancia de manera manual antes de realizar el proceso. Sin embargo el autor no realiza una desambiguación del sentido de las palabras ni evalúa la ambigüedad del sentido de las palabras en la especificación.

Para evitar las ambigüedades sintácticas (es decir ambigüedades de forma de la frase) presentes en los requisitos de software obtenidos a partir del lenguaje natural, en [57] y [58] presentan un meta-modelo para la representación de requisitos de software, usando reglas sintácticas y patrones lingüísticos preestablecidos, sin embargo estos meta-modelos, no atacan el problema de la ambigüedad semántica, ya que son lenguajes

controlados por tanto evitan el componente semántico que permita asociar el significado de cada elemento para el contexto del meta-modelo propuesto.

En [59], se presenta una herramienta para la generación de especificaciones de modelos de negocio, estos autores con el fin de evitar que los modelos referenciados o procesos de negocio, se encuentren ambiguos, definen en su modelo que el analista de software debe ingresar manualmente la definición de cada proceso del modelo, con el fin de eliminar las ambigüedades presentes en el lenguaje y procesos de negocio obtenidos. Sin embargo el autor no realiza la tarea de desambiguación de manera automática, ni relaciona los modelos de proceso de negocio con los requisitos de software.

En [60], se presenta una técnica para la decisión manual de ambigüedades en las especificaciones de requisitos de software en lenguaje natural. Los autores proponen llevar cada frase que componga el documento de educción de requisitos a lógica formal con el fin de detectar las ambigüedades y eliminarlas en el proceso manual de transformación a lógica formal descriptiva. Los autores utilizan una serie de herramientas, que ayudan al analista en ese proceso manual de la transformación de los requisitos a la lógica formal.

En la siguiente tabla, se resume el enfoque de las propuestas descritas.

Tabla 1-6 Resumen comparación de trabajos relacionados.

Enfoque	Usan Lenguaje Natural	Lenguaje Natural Controlado	Modelo de educción de requisitos usado	Ambigüedad Atacada
[11]	NO	SI	Modelos UML	Morfológica y Sintáctica.
[53]	SI	NO	Ninguna.	Sintáctica, mediante reglas.
[54]	Si	No.	Modelos Orientados a	Sintáctica.

			Objetos.	
[55] y [56]	Si	No.	Modelos de procesos de negocio.	Sintáctica y semántica
[57] y [58]	No	Si	Ninguna.	Sintáctica.
[59]	Si	No	Modelos de proceso de negocio.	Semántica.
[60]	Si	No	Modelo de clases	Sintáctica.

2. Ambigüedad semántica en requisitos de software

La interpretación que el analista trata de obtener con base en lo expresado por el interesado, está sujeta a un conjunto de ambigüedades propias del lenguaje natural, tanto hablado como escrito. Este conjunto de ambigüedades presentes en el lenguaje, es transversal a todos los idiomas y por tanto es ampliamente tratado en la literatura.

2.1. El lenguaje

Hay una inmensidad de definiciones sobre qué es el lenguaje humano, dependiendo de cada autor en cada época y en cada circunstancia.

- Por el lenguaje entendemos un sistema de códigos con cuya ayuda se designan los objetos del mundo exterior, sus acciones, cualidades y relaciones entre los mismos [61] y [62].
- El lenguaje es un conjunto finito o infinito de oraciones, cada una de las cuales posee una extensión finita y construida a partir de un conjunto finito de elementos [63], [64] y [65].

Definición 2.1. Formalmente podemos definir el lenguaje como un conjunto de símbolos I, que se agrupan para formar palabras W, y se agrupan para formar frases F, donde cada palabra W, tiene un sentido S y cada frase F, tiene un sentido Fs.

Por tanto para que un lenguaje sea llamado como tal, debe cumplir cuatro criterios:

I. Símbolos: Define el conjunto posible de símbolos.
II. Sintaxis: Define la corrección de un ensamblaje de símbolos.

III. Semántica: Define la capacidad de darle un "significado" a los ensamblajes de símbolos.
IV. Deducción: La capacidad de derivar nuevos ensamblajes de símbolos (de otros ya planteados) con un significado determinado.

2.2. La Ambigüedad

La ambigüedad es un fenómeno del lenguaje que ha existido a lo largo de la historia y ha evolucionado tanto como el lenguaje mismo. Concretamente la ambigüedad se da cuando un símbolo o un conjunto de símbolos, está sujeto a un conjunto de posible de interpretaciones o de significados.

Por tanto en primera instancia, es posible clasificar las ambigüedades del lenguaje en tres categorías correspondiente a los tres primeros criterios formativos del lenguaje mismo, a continuación se detalla cada una.

- Ambigüedad Morfológica.
- Ambigüedad Sintáctica o Estructural
- Ambigüedad Semántica o Pragmática
- Sin embargo los lenguajes no son solo conjuntos de símbolos escritos, sino también hablados y es por esto que surge también otro tipo de ambigüedad conocida como: Ambigüedad Fonética[2]

2.2.1. Ambigüedad Morfológica

Ocurre cuando una palabra que se encuentra en una frase representa más de un rol sintáctico o categoría gramatical dentro de la misma. También puede ser causada por la posibilidad de atribuir diferentes significados gramaticales a un mismo morfema. [66], [67], [68]. Las categorías gramaticales son clases de palabras, según la función que

[2] Este tipo de ambigüedad, no será expuesta en el presente trabajo debido a que solo se estudiarán las ambigüedades propias del lenguaje natural escrito.

desempeñan en la frase. Concretamente las categorías gramaticales para español son: Sustantivos, adjetivos, pronombres, verbos, adverbios, preposiciones, conjunción e interjección.

Un ejemplo de este tipo de ambigüedades, se dan en los requisitos de software como:

- El sistema bajo ciertas circunstancias debe fallar.
- El llenado de archivos debe ser completo.

Para la primera frase no es claro cuál es la categoría gramatical de la palabra “bajo”, (¿Verbo?, ¿Adverbio?, ¿Adjetivo?, ¿Sustantivo?). Por otro lado para segunda frase en la palabra llenado, no es fácil determinar cuál es su categoría gramatical dentro de la frase. (¿Verbo?, ¿Sustantivo?).

Definición 2.2. Formalmente definimos un problema de ambigüedad morfológica por la siguiente tupla:

$$<P, F, T>$$

En la cual:

- P, representa una palabra.
- F, representa la frase a la cual pertenece una etiqueta.
- T, representa el conjunto de categorías gramaticales posibles para la palabra P en una frase F.

Por tanto se da ambigüedad morfológica cuando para una palabra P, se tiene más de una categoría gramatical posible $|T| > 1$.

2.2.2. Ambigüedad Sintáctica

Se presenta cuando una frase tiene asociada más de una representación sintáctica, es decir, cuando más de un árbol sintáctico representa dicha frase. Esta ambigüedad se da porque la sola información sintáctica no es suficiente para realizar una decisión única de asignación de la estructura de la frase [69], [70], [71].

Un árbol sintáctico es una estructura de representación de la frase, que agrupa las categorías gramaticales en dependencias sintácticas según su función y su ubicación dentro de la misma. Una dependencia sintáctica es la agrupación de una o más categorías gramaticales de acuerdo a su función en la frase. Por ejemplo “el sistema” se puede agrupar en un sintagma nominal que representa al sujeto de la oración.

Un ejemplo de este tipo de ambigüedad se da en requisitos de software como:

- El sistema debe comprar los libros **baratos**.
- El sistema debe arreglar el software **rápido**.

En estos ejemplo es claro que no hay ambigüedad morfológica debido a que en ambos casos la palabra ambigua es un adjetivo, sin embargo no es claro a quién pertenece este adjetivo, es decir para la primera frase “baratos” puede ser el adjetivo de libros, en cuyo caso significa que había libros caros y baratos y que se compraron los últimos, pero también puede ser un complemento predicativo, en cuyo caso se quiere decir que quien compró los libros los consiguió baratos. Y para la segunda frase puede ser que hubiera varios camiones y de ellos arreglara el rápido, o puede ser que arreglara el camión con rapidez. Este tipo de ambigüedad se evidencia de una mejor manera al realizar los árboles sintácticos que representan la frase, a continuación se muestran los posibles arboles sintácticas para la primera frase:

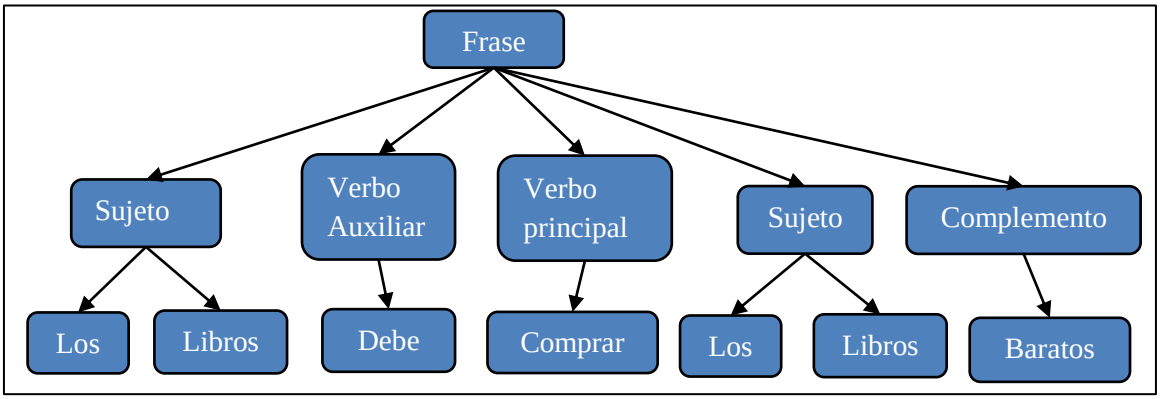


Figura 2.1 *Primer posible árbol sintáctico para la frase “El sistema debe comprar los libros baratos.”*
Fuente: Elaboración Propia.

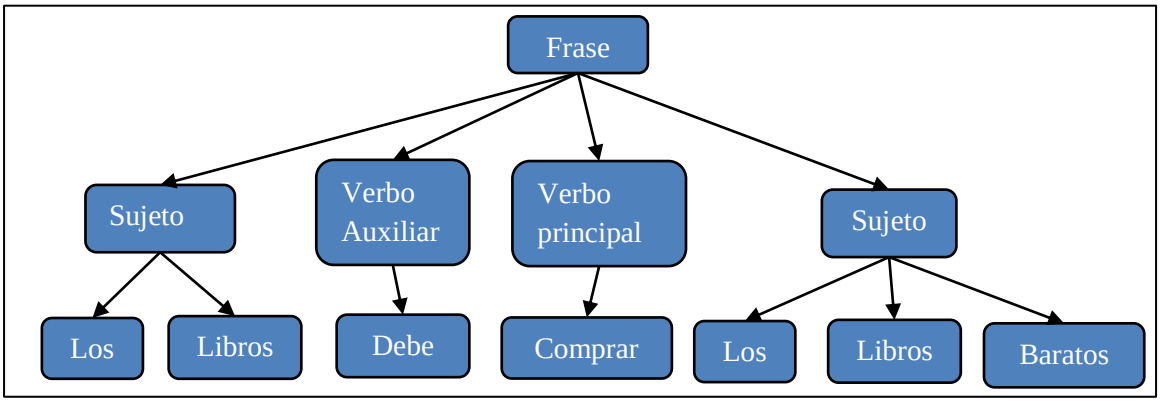


Figura 2.2 *Segundo posible árbol sintáctico para la frase "El sistema debe comprar los libros baratos."*
Fuente: Elaboración Propia.

Definición 2.3. Formalmente definimos un problema de ambigüedad sintáctica por la siguiente tupla:

$$<P, F, T, D>$$

En la cual:

- P, representa una palabra.
- F, representa la frase a la cual pertenece una palabra.
- T, representa la mejor categoría gramatical para la palabra P.
- D, representa el conjunto de dependencias sintácticas de una categoría gramatical T.

Por tanto se da ambigüedad sintáctica, cuando para una categoría gramatical T, se tiene más de una dependencia sintáctica posible $|D| > 1$.

2.2.3. Ambigüedad Semántica

Ocurre cuando una frase posee más de un significado o sentido; se refiere a fenómenos como la homonimia, (Palabras que se escriben igual pero con significado distinto) y la polisemia, (Palabras con múltiples significados), en los cuales la misma palabra puede tener distintos significados y por ende la frase misma puede tener distintas interpretaciones. La ambigüedad semántica se divide en varios tipos:

- Ambigüedad Semántica Polisémica. La ambigüedad semántica polisémica consiste en los múltiples significados que tiene una palabra, tal como puede quedar reflejado en un diccionario; la multiplicidad de significados se llama polisemia. Por ejemplo podría ser común en un proceso de educción de requisitos de software encontrar palabras o frases las siguientes:

Tabla 2-1. Ejemplos ambigüedades semánticas en requisitos de software en español.

Frase/Requisito	Posibles definiciones para la palabra con ambigüedad.
*"El sistema debe elaborar el diseño de un **banco**"*	1. *Asiento en que pueden sentarse varias personas.* 2. *Entidad financiera constituida en sociedad por acciones. Según sea su ejercicio mercantil, se le llama agrícola, de descuento, de emisión, de exportación, de fomento, hipotecario, industrial, etc.*
*"El sistema debe evitar usar el **gato**"*	1. *Se da este nombre a todos los félidos en general.* 2. *Máquina para levantar grandes pesos a poca altura, como la que se utiliza para elevar los vehículos al cambiar una rueda pinchada.*
*"El software podría realizar una **pila**"*	1. *Pieza grande de piedra u otra materia, cóncava y profunda, donde cae o se echa el agua para varios usos:* *pila de la cocina.* 2. *Pieza de piedra, cóncava, con pedestal, que hay en las iglesias parroquiales para administrar el sacramento del bautismo.* 3. *Generador de corriente eléctrica que transforma energía química en eléctrica.*

- Ambigüedad Semántica Referencial. Ocurre cuando una palabra o concepto tiene un significado de por sí difuso que no permite establecer con claridad a que elemento dentro del texto hace referencia, es decir una palabra cuyo contexto no permite determinar a qué o quién hace referencia. Un ejemplo de este tipo de ambigüedad se da en requisitos de software como:

  - El sistema le debe comprar flores.
  - El sistema garantizará la ejecución de la tarea.

En estos ejemplos no se logra establecer a que se refiere la palabra “le” ni a que se refiere (La tarea), por tanto no es posible establecer un sentido completo la frase dada la incompletitud del contexto.

Definición 2.4. Formalmente definimos un problema de ambigüedad semántica por la siguiente tupla:

$$<P, F, T, S>$$

En la cual:

- P representa una palabra.
- F representa la frase a la cual pertenece una palabra.
- T representa la mejor categoría gramatical para la palabra P.
- S representa el conjunto de sentidos o significados posibles para una palabra P con su respectiva categoría gramatical T.

Por tanto se da ambigüedad semántica cuando para una palabra P y una categoría gramatical T se tiene más de un sentido posible $|S| > 1$.

Durante esta tesis se enfrentará concretamente el problema de la ambigüedad semántica polisémica (Ambigüedad de los sentidos de las palabras). En la siguiente sección se presentan las reglas de validación y transformación de requisitos de software en elementos constitutivos usados en la presente tesis.

2.3. Transformación de requisitos de software en elementos constitutivos.

Un requisito de software, consiste en la descripción de una necesidad del cliente que un sistema informático debe satisfacer. Usualmente un requisito de software se encuentra escrito en un documento en lenguaje natural, este requisito de software es llevado por el analista o por un sistema computación a una arquitectura de educción de requisitos como KAOS, Casos de uso, Diagramas de Clases, Entidad Relación entre otras. Este requisito al incluirse en una arquitectura de educción de requisitos de software, se convierte en un elemento constitutivo como objetivos, clases, entidades, operaciones, agentes etc.

La transformación de un requisito de software en un elemento constitutivo desde el lenguaje natural, es tratada por diversos autores de manera automática [15] [14], sin embargo también los analistas de software realizan este proceso de manera empírica y manual. Esta transformación consiste en la elaboración de reglas morfosintácticas (Reglas de formación de la frase con base a las categorías y dependencias gramaticales), que permiten detectar cuando una frase contiene una estructura predefinida y llevarla a la estructura morfosintáctica del elemento constitutivo. Por ejemplo según [14]:

"El sistema debe registrar los clientes" siendo un requisito de software, llevado a un elemento constitutivo de tipo objetivo "Registrar los clientes" y un elemento constitutivo de tipo agente "El sistema".

Esta tesis realizará la validación de la transformación de los requisitos de software en elementos constitutivos, siguiendo el enfoque de [14], para la arquitectura de educción de requisitos de KAOS. Donde dado un conjunto de reglas de transformación X, se extraen elementos constitutivos A partir de una frase F.

Reglas morfosintácticas para la transformación de requisitos de software en elementos constitutivos.

Tabla 2-2. Reglas morfosintácticas para la transformación de requisitos de software en elementos constitutivos bajo la arquitectura de educción de requisitos KAOS.

Regla	Antecedente (Estructura que debe aparecer en la frase)	Consecuente (Estructura que se extrae de la frase)	Tipo de elemento constitutivo	Ejemplo Antecedente → Consecuente
R1	Sintagma Nominal + Perífrasis verbal modal + Verbo nominalizado + Complemento.	Verbo Principal + Verbo nominalizado + Complemento.	Objetivo	El sistema tiene que permitir el registro de los clientes → Permitir el registro de los clientes
R2	Complemento + "Que" + Sintagma Nominal + Verbo + Complemento.	Verbo + Complemento.	Objetivo	Yo quiero que el sistema registre los clientes → Registrar los clientes
R3	Verbo + "Que" + Sintagma Nominal + Verbo + Complemento.	Verbo + "Que" + Sintagma Nominal + Verbo + Complemento.	Objetivo	Garantizar que el sistema registre los clientes → Garantizar que el sistema registre los clientes.

R4	Sintagma Nominal + Perífrasis verbal modal + complemento.	Verbo Principal + complemento	Objetivo	El sistema tiene que registrar los clientes. → Registrar los clientes.
R5	Si una frase cumple alguna de las reglas R1 a R4, un agente será el sintagma nominal tipo sujeto de la frase.	Sintagma Nominal	Agente	El sistema tiene que registrar los clientes. → El sistema
R6	Si en una frase hay dos nombres que se encuentran unidos por las preposiciones "de" o "del", entonces el segundo sustantivo será una entidad, y el primero será un candidato para ser el atributo de la entidad previamente identificados.	Sintagma nominal	Entidad Atributo	El teléfono del gerente → Entidad Gerente Atributo Teléfono
R7	Si en una frase hay dos nombres que se encuentran unidos por el verbo “tiene”, entonces el segundo sustantivo será un atributo, y el primero será una entidad.	Sintagma nominal	Entidad Atributo	El gerente tiene teléfono → Entidad: Gerente Atributo: Teléfono
R8	Sustantivo + Verbo + (palabra libre) + sustantivo	Verbo	Operación	El sistema genera resultados → Generar

A continuación se explican las categorías y dependencias gramaticales usadas en la definición de las reglas presentadas en la tabla anterior.

Tabla 2-3. Estructuras gramaticales utilizadas en la definición de las reglas usadas para la transformación de requisitos de software en elementos constitutivos.

Estructura	Definición
Sustantivo	Nombre de persona animal o cosa.
Verbo	Palabra que describe la acción que se ejecuta en la oración.
Perífrasis Verbal	Agrupación de dos verbos, verbo auxiliar + verbo principal. (Verbo auxiliar puede ser Deber o tener que).
Sintagma Nominal	Parte de la frase que tiene como núcleo principal el sujeto. El sujeto puede estar acompañado de Determinantes o Adjetivos.
Complemento	El complemento se define como una agrupación de palabras, las cuales no contienen ningún verbo, pero por el contrario es permitido todo tipo de sustantivos, adverbios,

	adjetivos o palabras no funcionales, y no se limita el número de palabras presentes en el complemento, la única restricción es que el complemento no contenga ninguna forma verbal.
Sujeto	Es el sustantivo (Persona, Animal o cosa) que desempeña la acción en la oración junto son sus atributos si tiene.

En la siguiente sección se presentan los errores generados por la ambigüedad semántica en el proceso de educción de requisitos, para caracterizar estos errores se utilizan las reglas morfosintácticas expuestas en la presente sección.

2.4. Errores generados por la ambigüedad semántica en la educción de requisitos de software.

Dado un documento con la especificación de los requisitos de software y las reglas presentadas en la sección anterior, a continuación se presentan el conjunto de errores que posiblemente se generan al realizar el proceso de transformación de requisitos de software en elementos constitutivos, toda vez que tanto en el antecedente como en el consecuente, existe ambigüedad semántica, al no tener definido el sentido de cada palabra que conforma el requisito de software y por ende el elemento constitutivo.

2.4.1. Ambigüedad semántica

Este error, es el elemento que se tiene siempre presente al tener un elemento constitutivo, sin la definición de cada una de las palabras que compone el elemento constitutivo, por ejemplo para un elemento constitutivo de tipo objetivo, “Crear pilas”, sin una definición asociada a cada palabra, se genera el error de ambigüedad en el elemento constitutivo.

2.4.2. Inconsistencia

La inconsistencia se da cuando un elemento constitutivo es contradictorio u opuesto a otro elemento constitutivo [72]. Esto se determina una vez se soluciona el problema de la ambigüedad semántica y se tenga asociado un sentido para cada palabra que compone los elementos constitutivos.

Este error se identifica en la presente tesis, mediante la siguiente regla: si existen sentidos opuestos en las palabras que componen los elementos constitutivos y si comparten al menos un sintagma nominal en común entonces los elementos constitutivos involucrados son inconsistentes. En la siguiente figura se ilustra un ejemplo de este error.

	Elemento constitutivo 1			Elemento constitutivo 2		
	Crear	**Los**	**Carros**	**Destruir**	**Los**	**Carros**
Categoria Gramatical	Verbo	Articulo	Sustantivo	Verbo	Articulo	Sustantivo
Dependencia Gramatical	Verbo Principal	Sintagma Nominal		Verbo Principal	Sintagma Nominal	
Sentido	spa-30-01617192-v		spa-30-02958343-n	spa-30-01619929-v		spa-30-02958343-n
Definición	Mnufacturar un objeto		Vehículo Automotor de 4 ruedas	acabar con, causar la destrucción o perdición de		Vehículo Automotor de 4 ruedas

Figura 2.3 *Ejemplo problema de inconsistencia entre elementos constitutivos. Fuente: Elaboración Propia.*

2.4.3. Duplicidad

La duplicidad se da cuando dos elementos constitutivos significan lo mismo, así las palabras que los constituyen sean diferentes. Dado dos elementos constitutivos cuyas palabras que los definen tienen definidos sentidos iguales entonces se da duplicidad entre estos elementos constitutivos. En la siguiente tabla se presenta un ejemplo de duplicidad en elementos constitutivos.

Tabla 2-4. Ejemplo Duplicidad en elementos constitutivos

	Elemento constitutivo 1			Elemento constitutivo 2		
Palabra	**Crear**	**Los**	**Carros**	**Fabricar**	**Los**	**automóviles**
Categoría Gramatical	Verbo	Articulo	Sustantivo	Verbo	Articulo	Sustantivo
Dependencia Gramatical	Verbo Principal	Sintagma Nominal		Verbo Principal	Sintagma Nominal	
Sentido	spa-30-01617192-v		spa-30-02958343-n	spa-30-01619929-v		spa-30-02958343-n
Definición	Manufacturar un objeto		Vehículo Automotor de 4 ruedas	Manufacturar un objeto		Vehículo Automotor de 4 ruedas

2.4.4. No unificidad.

El problema de no unificidad se da cuando se presentan elementos constitutivos que tienen sentidos iguales pero las palabras utilizadas para la definición son diferentes en los elementos constitutivos. A continuación se presenta un ejemplo.

	Elemento constitutivo 1						
Palabra	**Crear**	**Los**	**Carros**				
Categoria Gramatical	Verbo	Articulo	Sustantivo				
Dependencia Gramatical	Verbo Principal	Sintagma Nominal					
Sentido	spa-30-01617192-v		spa-30-02958343-n				
Definición	Manufacturar un objeto		Vehículo Automotor de 4 ruedas				
	Elemento constitutivo 2						
Palabra	**Elaborar**	**Las**	**llantas**	**de**	**los**	**Automóviles**	
Categoria Gramatical	Verbo	Articulo	Sustantivo		Articulo	Sustantivo	
Dependencia Gramatical	Verbo Principal	Sintagma Nominal			Sintagma Nominal		
Sentido	spa-30-01617192-v		spa-30-13902336-n			spa-30-02958343-n	
Definición	Manufacturar un objeto		forma de un borde elevado de un objeto más o menos circular			Vehículo Automotor de 4 ruedas	
	Elemento constitutivo 3						
Palabra	**Minimizar**	**Los**	**Daños**	**sufridos**	**en**	**las**	**ruedas**
Categoria Gramatical	Verbo	Articulo	Sustantivo	verbo	Preposición	Articulo	Sustantivo
Dependencia Gramatical	Verbo Principal	Sintagma Nominal		Verbo auxiliar		Sintagma Nominal	
Sentido	spa-30-01617192-v		spa-30-02958343-n	spa-30-00668099-v			spa-30-13902336-n
Definición	Manufacturar un objeto		Vehículo Automotor de 4 ruedas	aguantar algo o a alguien desagradable			forma de un borde elevado de un objeto más o menos circular
	Elemento constitutivo 4						
Palabra	**Generar**	**Los**	**Planos**	**del**	**Automotor**		
Categoria Gramatical	Verbo	Articulo	Sustantivo	Articulo	Sustantivo		
Dependencia Gramatical	Verbo Principal	Sintagma Nominal		Sintagma Nominal			
Sentido	spa-30-01617192-v		spa-30-04076846-n		spa-30-02958343-n		
Definición	Manufacturar un objeto		una creación que es una representación visual o tangible de alguien o algo		Vehículo Automotor de 4 ruedas		

Figura 2.4 *Ejemplo elementos constitutivos no unificados*
Fuente: Elaboración Propia.

2.5. Conclusiones

En este capítulo se presentaron las ambigüedades más importantes que afectan el lenguaje natural escrito y por ende los requisitos de software tal como se explicó en el marco teórico. Las ambigüedades expuestas son: Ambigüedad Morfológica, Ambigüedad Sintáctica y Ambigüedad semántica. Estas ambigüedades se caracterizaron de manera formal y sistémica, se presentó el esquema utilizado por esta tesis para la transformación de requisitos de software en elementos constitutivos, y finalmente se presentaron los problemas asociados a la ambigüedad semántica (Ambigüedad, Inconsistencia, Duplicidad y No unificidad), los problemas enunciados en este capítulo son los problemas a los que esta tesis pretende dar solución.

Lo anterior se ejecutó para dar cumplimiento al primer objetivo de la presente tesis: “Identificar las ambigüedades semánticas que generan errores durante el proceso de educción de requisitos de software, con el fin de caracterizar estas ambigüedades formalmente y de manera sistémica.”

3. Evaluación de técnicas de desambiguación polisémica en ambientes de educción de requisitos de software

3.1. Introducción

El lenguaje natural escrito es menos susceptible a los problemas de ambigüedad, debido a que posee unas reglas que lo hacen más estricto que al lenguaje natural hablado. Sin embargo, el lenguaje natural escrito es también abundante en ambigüedades, entre éstas, se encuentra la ambigüedad polisémica o ambigüedad del sentido de las palabras.

La ingeniería de software es una de las ramas de las ciencias de la computación que intenta definir de manera clara y no ambigua los requisitos que un software debe cumplir. En este proceso de definición de requisitos intervienen diferentes actores tales como: Interesados (usuarios), analistas de requisitos, ingenieros de requisitos, entre otros. Pero, aun así, este proceso se torna difícil, debido a las ambigüedades presentes en el lenguaje utilizado por los interesados. Dado lo anterior, los analistas se ven obligados a crear mecanismos de representación intermedios (modelo de KAOS, Diagramas de clases, Diagrama Entidad Relación y Casos de uso, entre otros) para los interesados. Sin embargo, estos mecanismos de representación, siguen teniendo la ambigüedad propia del lenguaje[7].

Diversas técnicas de desambiguación polisémica han sido creadas a lo largo de la historia por diferentes grupos u organizaciones científicas [29], debido a que este problema es transversal a diversas áreas del conocimiento.

3.2. Técnicas de desambiguación

La desambiguación del sentido de las palabras, puede ser vista como una tarea de clasificación, en la que los sentidos de las palabras son clases y se asigna para cada

palabra W, uno o más sentidos basados en la evidencia del contexto T y fuentes externas de conocimiento. Un contexto en el marco de la presente tesis se define como el conjunto de palabras que acompañan a una palabra en una frase. A continuación se presentan en detalle las técnicas de desambiguación usadas en la presente tesis, esto con el fin de contrastar en la evaluación con las técnicas presentadas en estado del arte en la sección 1.5.

3.2.1. Lesk simplificado

El algoritmo Lesk debe cumplir con las siguientes reglas básicas:

- Para todos los sentidos de la palabra que se desea desambiguar, se debe contar la cantidad de palabras que se encuentran en el vecindario de la palabra ambigua y en la definición de cada sentido de la palabra en el diccionario seleccionado.
- El sentido que ha de ser elegido, será aquel que tenga el mayor número en el conteo realizado,
- No participan en el análisis las palabras vacías o “stop words” que son aquellas palabras diferentes a verbos, adjetivos o sustantivos.

La versión simplificada de este algoritmo compara la definición o los ejemplos del diccionario de una palabra ambigua, con los demás términos contenidos en su vecindario [43]. En la siguiente figura se presenta el seudocódigo del algoritmo *Simplified Lesk* utilizado para desambiguar palabras en una frase en la presente tesis.

```
1.function SimplifiedLesk (word, phrase) returns best_sense
2.      best_sense = NULL; //La primera vez el major sentido es nulo.
3.      /*La primera vez el no hay mejor sentido
4.      por tanto el max_counter es 0.*/
5.      max_counter = 0;
6.      context = set_of_words_in_the_phrase;//conjutno de palabras en la frase
7.      foreach sense in senses of Word
8.        /*Se obtiene el conjunto de palabras que contiene la
9.         definición del sentido, para ello se accede a un
10.        diccionario computacional.*/
11.       signature = set_of_words_in_sense_definition;
12.     /*Retorna el número de palabras encontradas en la definición del
13.       sentido que también se encuentran en las palabras de la frase. */
14.       counter = score(signature,context);
15.        full_senses = push(sense,counter/contex.size); //guardamos el sense y su score
16.     //Se compara si ya existe un sentido que obtuvo un counter mayor,
17.     if (counter>max_counter) then
18.         //de lo contrario el counter actual será el counter del mejor sentido.
19.          max_counter  = counter;
20.         //Asignamos el best_sense temporal.
21.         best_sense      = sense;
22.    endif
23.  endforeach
24.  /*Retornamos el mejor sentido encontrado,
25.     si no se encontró ninguno se devolverá NULL.*/
26.  return best_sense;
27.  return full_senses;
28.endfunction
```

Figura 3.1 *Algoritmo Simplified Lesk utilizado en la presente tesis. Fuente: Elaboración Propia.*

Donde *word* representa la palabra a desambiguar y *phrase* las palabras que conforman la frase donde se encuentra la palabra *word* a desambiguar. *Context* representa las palabras funcionales (adjetivos, verbos, adverbios y sustantivos) que acompañan la palabra *word* en *phrase*. *senses* son el conjunto de significados posibles de la palabra Word provistos por una base de conocimiento. *best_sense* representa el sentido con el mejor *counter* y *full_senses* será el arreglo que contiene la tupla (*sense*, *counter*) sentido y coeficiente de ese sentido, donde el coeficiente *counter* se calcula mediante la función *score* de la siguiente manera:

$$score = |\#palabras\ en\ la\ definición\ de\ diccionario\ de\ word \cap \#context| \quad (1)$$

Finalmente el algoritmo devuelve el *best_sense* y los *full_senses*.

Un ejemplo de operación del algoritmo Lesk se presenta a continuación: desambiguar la palabra “Pino” en la frase, “Los conos de pino cuelgan en un árbol”.

Lesk([Pino], [Conos] [Cuelgan][Arbol]);

Sentidos posibles de “Pino”:

1. tipo de árbol de hoja perenne con hojas en forma de aguja
2. consumirse a través de la tristeza o enfermedad.

El algoritmo debe seleccionar el sentido #1 para el “pino”, debido a que la palabra árbol se encuentra en la definición de este sentido y a su vez se encuentra la frase o vecindario de la palabra a desambiguar, mientras que en el sentido 2, no hay coincidencia con ninguna de las palabras de la frase o vecindario de la palabra a desambiguar.

3.2.2. Lesk adaptado

Autores como [44] proponen modificar el algoritmo original *Simplied Lesk*, debido a que presenta limitaciones al solo comparar las palabras en el vecindario *context* de la palabra a desambiguar *Word*, contra las palabras en las definiciones de la palabra *word* a desambiguar *senses*, esto genera en muchos casos una pobre desambiguación cuando el contexto de la palabra está limitado, o cuando las definiciones *senses de word* se encuentran limitadas [73].

Dado lo anterior en [44] se propone el algoritmo *adapted Lesk* el cual consiste en tomar las definiciones *senses* de las palabras del contexto c*ontext* donde *context* representa las palabras funcionales (adjetivos, verbos, adverbios y sustantivos) en la frase *phrase*, y comparar (usualmente de a dos palabras) con las definiciones de los sinónimos (sentidos similares), hiperónimos (sentidos más generales), hipónimos (sentidos más específicos), merónimos (sentidos que funcionan como parte de otro sentido), holónimos (sentidos que funcionan como todo de otro sentido) y tropónimos (hipónimos verbales) de cada palabra del contexto. En la siguiente figura se detalla el algoritmo adapted Lesk.

```
1.function AdaptedLesk (phrase) returns score:list<string>
2.        context[] = set_of_words_in_the_phrase; //Conjunto de palabras en la frase
3.        for i in 1..(context.length)
4.          sense[] ← senses(context[i]) //Obtenemos todos los sentidos de cada palabra del contexto
5.          for j in 1..(sense.length)
6.            glosses_related[i][j][] ← { definition (synonyms(sense[j])) +        //Asignamos en glosses_related el gloss
de
7.            definition (hiperonyms(sense[j])) + definition (holonyms(sense[j])) //palabras del contex que se encuentran
8.            + definition (meronyms(sense[j])) + definition (troponyms(sense[j])) //en cada una de las definiciones
10.         endfor
11.       endfor
12.       for i in 1..(context.length)
13.         for j in 1..(context.length)
14.               if(i != j)
15.                 score[i][j]←overlap(glosses_related[i],glosses_related[j])/contex.size //Calculamos el score de cada
16.               endif                                                    // sentido de cada palabra del context
17.         endfor
18.       endfor
19.       for i in 1..(context.length)
20.         sense[i] ← max_score(score[i]) //Preguntamos por el mejor sentido de cada palabra en el context.
21.       endfor
22.       return senses_by_word[]; //Lista de sentidos ordenados de acuerdo al score
23.endfunction
24.function Overlap (glossesA[][], glossesB[][]) returns higher:map<string,int>
25.       for ia in 1..glossesA[].length
26.         for ja in 1..glossesA[][].length
27.           for ib in 1..glossesB[].length
28.             for jb in 1..glossesB[][].length
29.                   score[ia] ←higher_consecutive_string_words (glossesA[ia][ja], glossesB[ib][jb])
30.           endfor
31.         endfor
32.       endfor
33.       endfor
34.       higher ← sense(glossesA[1]),score[1])
35.       for ia in 2..glossesA[].length
36.         if(score[1] > higher.score)
37.           higher ← sense(glossesA[ia]),score[ia])
38.         endif
39.       endfor
40.       return higher;
41. endfunction
```

Figura 3.2 *Algoritmo Adapted Lesk utilizado en la presente tesis. Fuente: Elaboración Propia.*

La comparación se hace mediante la función *overlap,* la cual consiste en hallar el tamaño de la cadena de palabras funcionales (adjetivos, verbos, adverbios y sustantivos), consecutivas más larga, que esté presente en los pares de definiciones.

Finalmente el algoritmo devolverá el *senses_by_word* que contiene los sentidos para cada palabra de la frase *phrase* y su respectivo score asociado.

3.2.3. Algoritmo MFS (Most Frequent Sense)

El sentido más frecuente (o MFS, por sus siglas en inglés), es un algoritmo de desambiguación semántica polisémica, que permite encontrar el sentido mayormente utilizado en una base de conocimiento, para cada palabra. Este método parte de una base de conocimiento anotada con los sentidos para cada palabra y un coeficiente de frecuencia el cual determina cual sentido *sense* es más usado para una palabra *word* [45]. En la siguiente figura se detalla el algoritmo Most Frequent Sense.

El algoritmo mostrado realiza un precargado de la base de conocimiento *kb* para posteriormente dada una palabra *word* se consulte sobre la variable dictionary, sus respectivos sentidos *senses* ordenados de acuerdo al *score.* Finalmente el algoritmo Most Frequent Sense, devuelve para una palabra *word* la lista de sentidos y su *score* asociado mediante el array *senses*.

```
1.function MFS (word) returns score:list<string>
2.        dictionary ← LoadMFS(); //Ponemos todos los sentidos adentro de la variable diccionario
3.        senses ← dictionary.get(word).sentidos; //Obtenemos todos los sentido de acuerdo a la palabra buscada
4.        return senses; //Devolvemos todos los sentidos para la palabra buscada ordenados por el sentido mas frecuente
5.endfunction
6.function LoadMFS() retorna map<String,map<String,int>>
7.        //Asignamos a la kb todas los sentidos para cada palabra
8.        kb = Load(All_scores_senses_database);  // contendios en una  base de conocimiento.
10.       Map c;
11.       for i in 1 .. kb.size
12.          if(!dictionary.contains(kb (i)))
13.             dictionary.put(kb (i).lema, kb (i).sense,kb(i).score,1); //Agregamos un Nuevo sentid al diccionario
14.          else
16.             cont = dictionary.get(kb (i).lema, kb (i).sense);
17.             dictionary.put(kb (i).lema, kb (i). sense,kb(i).score,cont++);
18.          endif
19.       endfor
20.       for i in 1 .. dictionary.size
21.          Sort_by_most_frequent_sense(dictionary (i));//Ordenamos la el diccionario
23.       endfor                                          // deacuerdo al sentido mas frecuente
24.       return dictionary;
25.endfunction
```

Figura 3.3 *Algoritmo Most Frequent Sense utilizado en la presente tesis. Fuente: Elaboración Propia.*

3.2.4. UKB

El método consiste en realizar recorridos sobre el grafo de cualquier base de conocimiento (LKB) Dada una base de conocimientos léxica (LKB), se construye un grafo no-dirigido *G=(V,E)* donde los nodos representan los conceptos *vi* de la LKB, y cada relación entre conceptos v_i y v_j se representa con un vértice no-dirigido $e_{i,j}$. Dado un texto de entrada, se extrae la lista W_i *(i=1…m)* de las palabras de funcionales (sustantivos, verbos, adjetivos y adverbios) que tienen una entrada en el diccionario y, por tanto, se pueden relacionar a conceptos de la LKB. Sea $senses_i$ **=** *{v1,…,*vi_m*}* los i_m sentidos asociados a la palabra Wi en el grafo de la base de conocimiento. De esta forma, las palabras polisémicas, se asocian a varios conceptos; las monosémicas, solo a uno. El resultado del proceso de desambiguación consiste en que cada sentido sense de la lista $senses_i$, recibe un puntaje. Así, para cada palabra a desambiguar, se selecciona el concepto que en *G* posee el máximo puntaje. En la siguiente figura se detalla el algoritmo implementado de UKB.

```
1.function loadUKB(phrase) return Map<String,Map<String,int>>
2.        read kb; //base de conocimiento etiquetada con el sentido for cada palabra
3.        Map dictionary;
4.        For i in 1 .. kb.size {
5.          For j in kb(j) - context.. kb(j)+ context {
6.                    sensesj [] ← synsets(kb(j));
7.                    Gkb ← relevant_concepts(kb(j)- context..kb(j)+ context);
8.                    For k in 1 .. sensesj.size{
9.                              shortest_path ← BFS(senses(k),Gkb);
10.                             Gd.add(shortest_path); //Grafo de desambiguación
11.                   endfor
12.         endfor
13.         score = PageRank(Gd);
14.         dictionary.add(kb(i),score); //Añadimos el sentido y el score para cada palabra en la frase.
15.       endfor
16.       return dictionary;
17.endfunction
18.function BFS(Graph G, Node a): List<Node>
19.       foreach (vertex u: u.explorado = falso);
20.       a. explored = falso;
21.       q = new list;
22.       q.push(a);
23.       while (q ≠ Ø)
24.         u = q.pop();
25.         foreach vertex (u, v)
26.           if (v.explored = false) then
27.             v. explored = true;
28.             q.push(v);
29.           endif
30.         endforeach
31.       endwhile
31.endfunction
```

Figura 3.4 *Algoritmo UKB utilizado en la presente tesis.*
Fuente: Elaboración Propia

El puntaje se determina con base en la adaptación que hacen [48] al PageRank[3]. La idea principal de este método es extraer un subgrafo G_{KB} de la LKB, cuyos vértices y relaciones son particularmente relevantes para el contexto dado. Este grafo se conoce como subgrafo de desambiguación G_D, y se construye de la siguiente forma: Para cada palabra *Wi* en el contexto de entrada, y cada concepto $v_i \in Sentidos_i$, se realiza una búsqueda en amplitud ("breadth-first search" o BFS) sobre G_{KB}, comenzando por el nodo

[3] En la Figura 3.4 no se muestra dicho algoritmo, dada su longitud y complejidad, para conocer más acerca de cómo funciona el PageRank remitirse a [48]).

v_i. Cada ejecución del BFS calcula la mínima distancia de las rutas entre v_i y el resto de conceptos de G_{KB}. Concretamente, interesa encontrar la mínima distancia de caminos entre v_i y los conceptos asociados al resto de las palabras en el contexto, Sea vdp_{vi} el conjunto de estas rutas más cortas.

$$v_j \epsilon \bigcup_{j \neq i} \epsilon sentidos_j \quad (2)$$

El cálculo de la ruta más corta utilizando BFS se repite para cada sentido de cada palabra en el contexto, almacenando la vdp_{vi} respectiva. Al final, se obtiene un conjunto de rutas de mínima distancia cada una con diferentes conceptos como punto de partida. Así, el grafo de desambiguación G_D es la unión de los vértices y aristas de las rutas más cortas:

$$G_D = \bigcup_{i=1}^{m} \left\{ mdp_{v_j} / v_j \epsilon \epsilon sentidos_i \right\} \quad (3)$$

El grafo de desambiguación G_D es un subgrafo del grafo original G_{KB}. De esta forma, se puede decir que captura los conceptos y relaciones más relevantes en la base de conocimientos para un contexto de entrada particular. Con base en este grafo, se puede calcular el pagerank. la idea principal del pagerank es que siempre que haya un enlace entre vi y vj en un grafo, se produce un voto del nodo i al nodo j, y así aumenta el puntaje del nodo j. además, la magnitud de este voto depende del puntaje del nodo i: entre más importante sea el nodo i, mayor será el puntaje de j. Finalmente el algoritmo UKB devolverá la variable dictionary que contiene los sentidos para cada palabra de la frase *phrase* y su respectivo score asociado.

3.3. Evaluación de las técnicas de desambiguación.

A continuación se presenta una tabla comparativa con la cual se establece un mecanismo de evaluación con el fin de determinar el método óptimo para la desambiguación del sentido de las palabras. La tabla se construyó con base en el estado del arte y a continuación se explican las métricas usadas para su evaluación.

Para evaluar este tipo de trabajos comúnmente se utiliza un corpus de referencia, es decir, un conjunto de palabras polisémicas en contextos específicos, donde el sentido de la palabra ha sido marcado previamente por un juez. La precisión es el porcentaje de

palabras correctamente desambiguadas por el sistema de WSD dado un corpus de referencia (número de instancias correctamente desambiguadas por el sistema WSD / número de instancias a las cuales el sistema de WSD propuso una respuesta). El recuerdo es el porcentaje de palabras que han sido correctamente desambiguadas dentro del conjunto de todas las palabras de prueba (número de instancias correctamente desambiguadas / número total de instancias en el conjunto de prueba).

La cobertura es el porcentaje de palabras a las que el sistema WSD, ha dado respuesta (número de instancias a las cuales el sistema de WSD, propuso una respuesta / número total de instancias en el conjunto de prueba).

Por ejemplo, suponiendo que se está manejando un conjunto de prueba con 100 palabras y el sistema WSD, sólo es capaz de proponer respuestas para 75 palabras y de estas 75, sólo 50 palabras han sido desambiguadas correctamente. La precisión en este ejemplo es de 50 / 75 = 0.66, el recuerdo es 50/100 = 0.50 y por último, la cobertura es de 75/100 = 0.75.

El parámetro de relaciones jerárquicas, indica si el método de desambiguación usa las relaciones jerárquicas entre sentidos para desambiguar, es decir, si un sentido es más general que otro o más específico que otro. El parámetro relaciones horizontales establece si es método de desambiguación usa relaciones horizontales entre sentidos, es decir si un sentido es sinónimo a otro o similar a otro.

El parámetro métodos estadísticos, determina si el método de desambiguación, parte sobre rankings de coeficientes anotados para los sentidos y finalmente el parámetro entrenamiento determina si el método de desambiguación se debe adaptar al texto o al contexto del texto o si funciona de manera independiente para cualquier texto, es decir si es de propósito general y listo para implementar.

Tabla 3-1 Evaluación técnicas de desambiguación referenciados en el presente trabajo.

Método de desambiguación		Promedio	Relaciones Jerárquicas	Relaciones Horizontales	Métodos Estadísticos	Entrenamiento
Most Frequent Sense [29]	**Cobertura**	92,00%	NO	NO	SI	NO
	Precisión	78,00%				
	Recuerdo	70,00%				

UKB [74]	**Cobertura**	90,00%	SI	SI	SI	NO
	Precisión	79,30%				
	Recuerdo	78,00%				
Lesk Adaptado[44]	**Cobertura**	80,00%	SI	SI	NO	NO
	Precisión	68,00%				
	Recuerdo	59,00%				
Lesk simplificado [43]	**Cobertura**	80,00%	NO	NO	NO	NO
	Precisión	50%				
	Recuerdo	53,00%				
RTE-2 [75]	**Cobertura**	70,00%	NO	NO	SI	SI
	Precisión	52,00%				
	Recuerdo	53,00%				
RTE-3 [75]	**Cobertura**	76,00%	NO	NO	SI	SI
	Precisión	57,00%				
	Recuerdo	59,00%				
Maximum Entropy [76]	**Cobertura**	70,00%	NO	NO	SI	SI
	Precisión	80,00%				
	Recuerdo	81,00%				
SVM [76]	**Cobertura**	75,00%	NO	NO	SI	SI
	Precisión	77,00%				
	Recuerdo	78,00%				
Naïve-Bayes [76]	**Cobertura**	74,00%	SI	NO	SI	NO
	Precisión	74,00%				
	Recuerdo	76,00%				
Kstar [76]	**Cobertura**	72,00%	NO	NO	SI	NO
	Precisión	71,00%				
	Recuerdo	72,00%				

En la siguiente figura se presentan los datos expuestos en la Tabla 3-1.

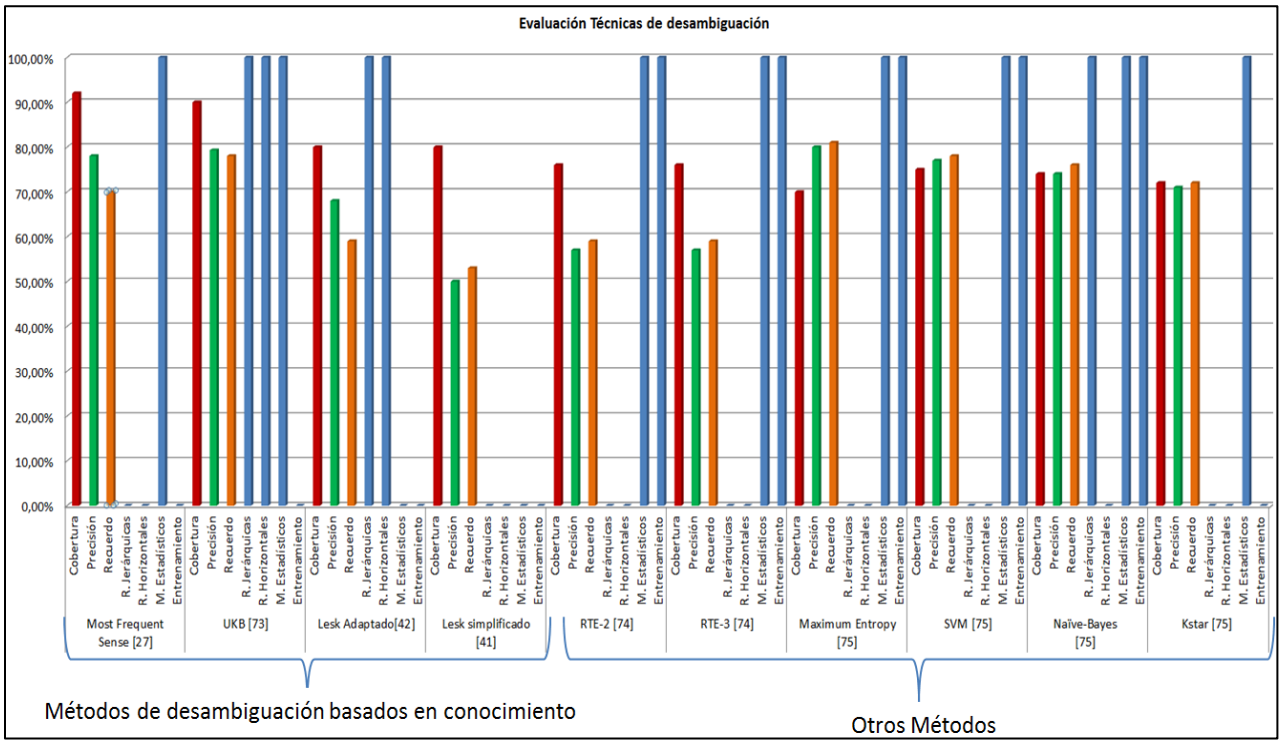


Figura 3.5 Evaluación de las técnicas de desambiguación.
Fuente: Elaboración Propia

3.4. Conclusiones

En este capítulo se presentaron en detalle las técnicas de desambiguación basadas en conocimiento. Se evaluaron dichas técnicas versus diversas técnicas de desambiguación referenciadas en el estado del arte, utilizando como métricas, la cobertura, precisión, recuerdo, manejo de relaciones jerárquicas, manejo de relaciones horizontales, métodos estadísticos y entrenamiento.

De la Tabla 3-1 y de la Figura 3.5 se puede concluir que las técnicas de desambiguación tienen una mayor cobertura respecto a las técnicas presentadas en el estado del arte. También se puede concluir que la relación cobertura/precisión es mayor en las técnicas de desambiguación basadas en conocimiento que las demás técnicas presentadas. Finalmente las técnicas de desambiguación basadas en conocimiento no requieren un entrenamiento previo y manejan relaciones jerárquicas y horizontales.

Dado lo anterior se concluye que los métodos basados en conocimiento, presentados en este capítulo, son los más adecuados, para minimizar errores en la identificación de requisitos, causados por la ambigüedad de los sentidos de las palabras durante la educción de requisitos de software.

Dicho lo anterior, con este capítulo se le pretende dar cumplimiento al segundo objetivo del presente trabajo: “Evaluar diferentes técnicas de resolución de ambigüedad de sentidos de palabras, con el fin de seleccionar la más adecuada para minimizar errores en la identificación de requisitos causados por la ambigüedad de los sentidos de las palabras durante la educción de requisitos de software”.

4. Modelo de desambiguación de los sentidos de las palabras para minimizar errores en la identificación de requisitos de software

4.1. Introducción

Este capítulo presenta un modelo de resolución de la ambigüedad de los sentidos de palabras para minimizar los errores en la identificación de requisitos de software. La capa principal de desambiguación que se lleva a cabo sobre elementos constitutivos usa las técnicas de desambiguación basadas en conocimiento expuestas en el capítulo 3. El modelo propuesto recibe los elementos constitutivos que componen la arquitectura de educción de requisitos de software, cada elemento constitutivo debe tener asociado el requisito de software de que originó el elemento constitutivo y la definición del elemento constitutivo. Posteriormente se realiza un proceso morfosintáctico de segmentación del elemento constitutivo por palabras y un proceso morfosintáctico de etiquetado para cada palabra. Luego se realiza una desambiguación usando el filtrado de coeficientes con el fin de determinar el mejor sentido para cada palabra que compone el elemento constitutivo. De esta forma, se entrega un elemento constitutivo desambiguado (Véase definición 4.3.). Luego el modelo determina los errores de inconsistencia, duplicidad, no unificidad y ambigüedad encontrados en los elementos constitutivos, para presentárselos al usuario con el fin de solucionar los errores encontrados de manera interactiva.

En la siguiente figura se presenta la estructura por capas del modelo propuesto.

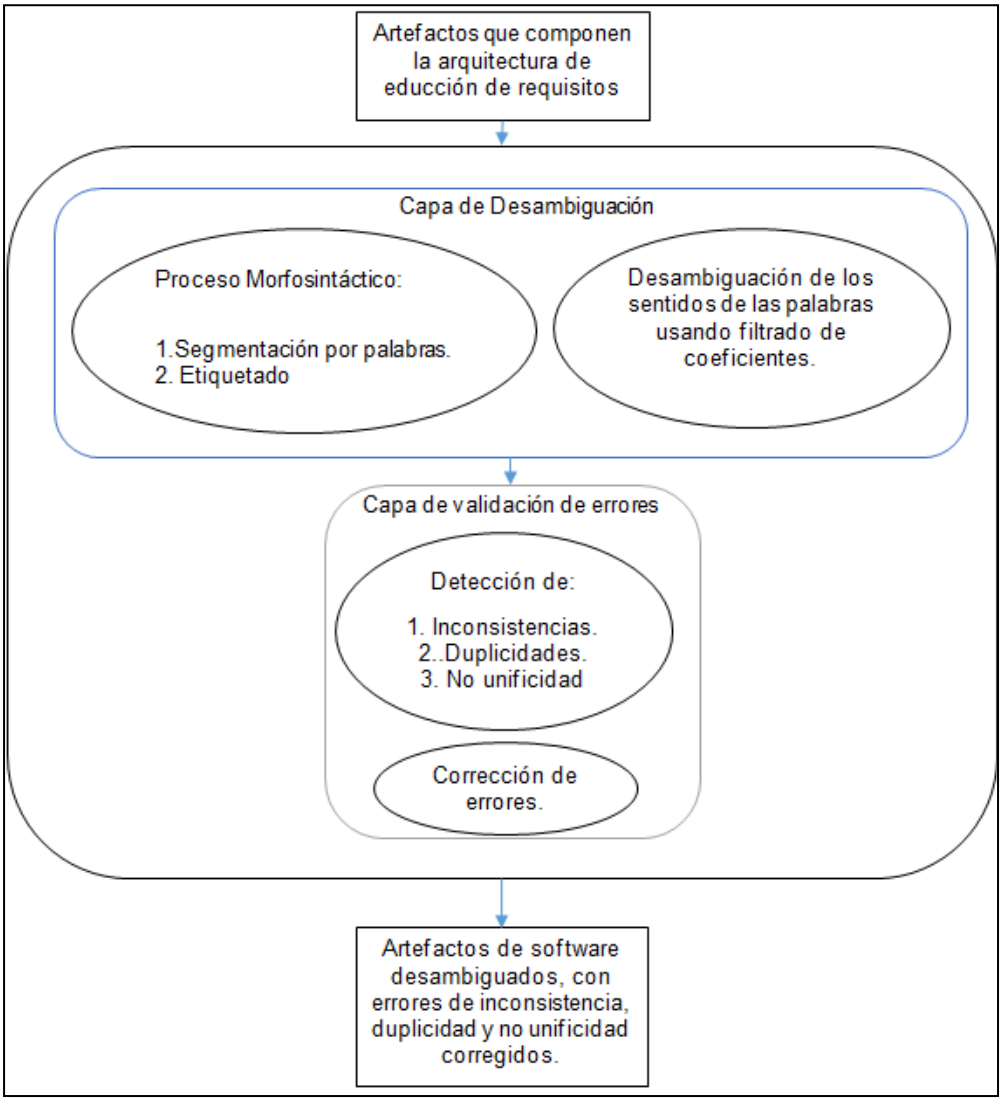


Figura 4.1 *Modelo de desambiguación de los sentidos de las palabras para minimizar errores en la identificación de requisitos de software.*
Fuente: Elaboración Propia

Una arquitectura de educción de requisitos provee la entrada del modelo propuesto, es decir entrega una lista de elementos constitutivos del modelo de educción de requisitos usado, y cada elemento constitutivo se define como una lista ordenadas de palabras. Cada elemento constitutivo contiene la frase fuente del requisito de software que originó el elemento constitutivo.

En las siguientes secciones se presenta la descripción de la capa de desambiguación y la capa de minimización de errores.

4.2. Capa de desambiguación

4.2.1. Proceso morfosintáctico

El modelo propuesto una vez recibe el elemento constitutivo y la frase fuente de donde se obtuvo dicho elemento constitutivo, realiza el proceso morfosintáctico de segmentación de cada palabra que compone la fuente del elemento constitutivo, esto convierte esta fuente del elemento constitutivo en un arreglo de palabras, y posteriormente el algoritmo indaga cuales palabras de la fuente se encuentran en la definición del algoritmo con el fin de asignarle el etiquetado correspondiente asignado en la frase fuente de elemento constitutivo. Finalmente el algoritmo retorna el arreglo de palabras que componen la frase fuente y el arreglo de palabras que componen la definición del elemento constitutivo junto con la etiqueta morfosintáctica asociada a cada palabra. El pseudocódigo del algoritmo utilizado para hacer la segmentación de las palabras que componen la definición del elemento constitutivo es:

```
1.function segmentar_etiquetar (fuente, definicion) returns array palabras_fuente, palabras_definicion
2.        palabras_fuente=split(frase, " ");//Particionamos la frase de acuerdo a los espacios entre palabras.
3.        palabras_definicion=split(definicion, " ");//Particionamos la definicion del artefacto
4.        foreach pf in palabras_fuente
5.          /*Se obtiene la etiqueta morfosintactica.*/
6.          etiqueta=obtenerEtiqueta(palabra); //Obtenemos la etiqueta
7.          pf.etiqueta=etiqueta;
8.          foreach pd in palabras_fuente
9.                    if(pd==pf) //Preguntamos si la palabra de la fuente es igual a la palabra
10.                             pd.etiqueta.=pf.etiqueta; // de la definicion, si es igual le asignamos
11.                   endif                              // también la etiqueta.
12.               endforeach
13. Endfunction
14.  /*Retornamos las palabras con la etiqueta de la fuente y de la definicion*/
15.  return palabras_fuente, palabras_definicion;
16.endfunction
17.
18.function obtenerEtiqueta(palabra) returns etiqueta
19.        diccionario =loadDiccionarioMorfosintactico();//Cargamos un diccionario que contiene
20.                                                      //todas las etiquetas morfosintácticas
21.                                                      //asociadas a cada palabra.
22.        foreach pd in diccionario //Para cada palabra en el diccionario
23.               if(palabra==pd) //Si la palabra en el diccionario es igual a la palabra que insertamos
24.                        return pd.etiqueta; //entonces le asignamos la etiqueta.
25.               Endif
26.        Endforeach
27.endfunction
```

Figura 4.2 *Algoritmo de segmentar y etiquetar.*

Luego a este proceso morfosintáctico se procede a realizar la desambiguación de cada palabra mediante el modelo de filtrado de coeficientes presentado en la sección 4.2.2. Antes de ello se presentan algunas definiciones.

Definición 4.2. Una palabra desambiguada se define mediante la tupla:

<*palabra*, *etiqueta*, *lema*, *dependencia_sintactica*, *sentido*>

Donde:

- Palabra representa el conjunto de caracteres que compone la palabra. Ejemplo: “carro”, “vasa”, “ganar”, “blanco”.
- Etiqueta representa la categoría gramatical a la cual pertenece la palabra. Ejemplo: “Verbo”, “Adjetivo”, “Sustantivo”, “Determinante”, “Adverbio”.
- Lema representa la raíz de la palabra es decir su origen gramatical. Ejemplo: La raíz de “corrió” es “correr”, la raíz de “los” es “el”.
- Dependencia_sintactica representa la función de la palabra en la frase. Ejemplo: En la frase “Juan come arroz” la dependencia sintáctica de “Juan” es “Sujeto de la frase”, la dependencia sintáctica de “arroz” es “Objeto de la frase”.
- Sentido representa la definición de la palabra de acuerdo a un diccionario computacional o base de conocimiento. Se define mediante un identificador único y un texto que describe ese sentido. Ejemplo: Un sentido de la palabra “Banco” es “spa-30-08420278-n” y su descripción es “*una institución financiera que acepta depósitos y canaliza el dinero en actividades de préstamo*”.

Definición 4.3. Un elemento constitutivo no ambiguo se define mediante la tupla.

<definición, fuente, palabras_definicion>

- *definición* representa las palabras que componen el elemento constitutivo.
- *fuente* representa la frase del requisito de software que dio origen al elemento constitutivo.
- *palabras_definicion* es un arreglo que contiene las palabras desambiguadas (véase definición 4.2) que componen el elemento constitutivo.

En la sección siguiente se presenta el modelo de resolución de ambigüedad de sentidos de palabras mediante el filtrado de coeficientes usando mínimos cuadrados.

4.2.2. Filtrado de coeficientes de sentidos de palabras mediante la técnica de mínimos cuadrados para la desambiguación semántica polisémica

La técnica de desambiguación propuesta recibe el arreglo de palabras de la frase fuente obtenidas por el algoritmo de etiquetado y segmentación. El algoritmo que se propone para el proceso de desambiguación polisémica de una palabra se compone del conjunto de técnicas de desambiguación basadas en conocimiento expuestas en la sección 3.2 de la presente tesis, este conjunto de algoritmos arrojan una lista de sentidos con su respectivo *score* para cada una de las palabras en el arreglo palabras_fuente.

Luego esos coeficientes son analizados por un mecanismo superior que valida el vector de coeficientes obtenido por cada algoritmo y lo procesa mediante un filtrado de coeficientes que utiliza una regresión de mínimos cuadrados, con el fin de obtener un score definitivo que garantice un mejor sentido al unir las cuatro técnicas utilizadas. En la siguiente figura se detalla el proceso:

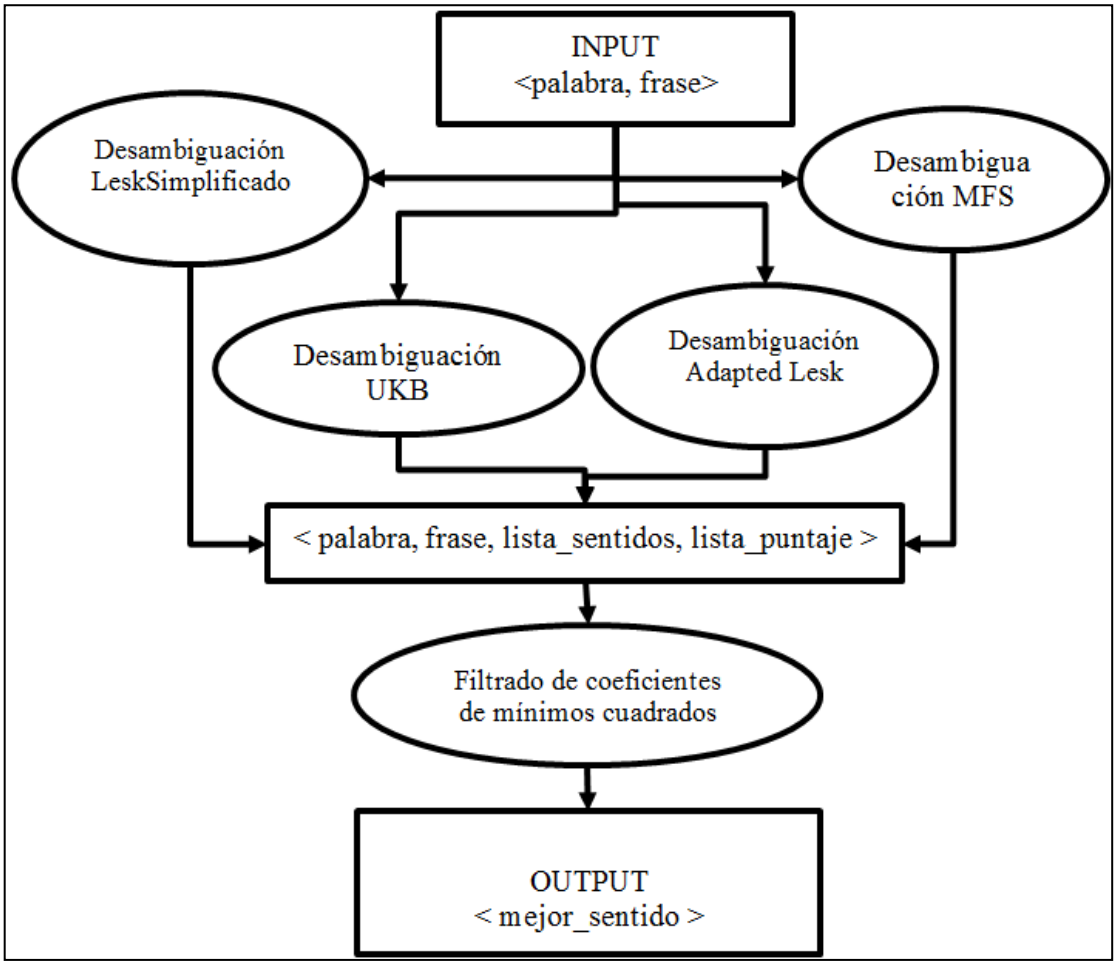


Figura 4.3 *Proceso por etapas del modelo de desambiguación*
Fuente: Elaboración Propia.

El primer paso para el filtrado de coeficientes consisten en leer el primer sentido arrojado por cada uno de los métodos desarrollados y expuestos en la sección 3.2 MFS, UKB, Simplified Lesk y Adapted Lesk. Luego, dado que el objetivo es obtener el mejor sentido, para el filtrado de coeficientes se implementa un filtro de mínimos cuadrados para la estimación de un escalar [77], por tanto se busca el coeficiente $\partial Score_i$ para cada sentido *i* de una palabra *w*. Se utilizarán cuatro aproximaciones (una por cada método de desambiguación) para cada sentido de cada palabra w_i. La ecuación principal utilizada es:

$$\partial Score_i = \overline{w}_i - \frac{\sum_{j=1}^{n=4}(x_{ij} - \overline{w}_i)^2}{n=4} \quad (4)$$

Donde $\partial Score_i$ representará el coeficiente arrojado por el filtro de mínimos cuadrados para el sentido *i*, x_{ij} representa el coeficiente para sentido *i* en el método de desambiguación *j*, n representa el número de coeficientes obtenido para nuestro modelo siempre será 4 ya que se implementaron cuatro métodos de desambiguación, y $\overline{w_i}$ representa la media de los coeficientes de los métodos de desambiguación *j* dada por la siguiente ecuación:

$$\overline{w}_i = \frac{\sum_{j=0}^{n=4} x_{ij}}{n=4} \quad (5)$$

Una vez se obtiene el $\partial Score_i$ para cada sentido *i* se procede a realizar la comparación de estos coeficientes con el fin de determinar el score definitivo e integrado de cada sentido, y se procede a compararlo, por tanto el mejor sentido *i* de una palabra será el sentido con el mejor $\partial Score$. En la siguiente figura se presenta el pseudocódigo del algoritmo de filtrado de coeficientes de mínimos cuadrados.

```
1 function filtrado_coeficientes (palabra) returns mejor_sentido
2.      sentidosMFS         = MFS(palabra, frase); //Obtenemos los sentidos sando el algoritmo
3.                                                 //MFS expuesto en figura 3.3
4.      sentidosUKB         = loadUKB(palabra, frase); //Obtenemos los sentidos sando el algoritmo
5.                                                     //expuesto en loadUKB figura 3.4
6.      sentidosSimLesk     = simplifiedLesk();    //Obtenemos los sentidos usando el algoritmo
7.                                                 //simplifiedLesk expuesto en la figura 3.1
8.      sentidosAdapLesk=adaptedLesk();            //Obtenemos los sentidos usando el algoritmo
9.                                                 //adaptedLesk expuesto en la figura 3.2
10.     /*Este for analiza cada uno de los sentidos indagando por los
11.     scores de los sentidos*/
12.     for(i=0;i<sentidosMFS.size();i++)
13.             mejor_sentido=NULL; // La primera vez el mejor sentido
14.             mejor_sentido.score=0; //esta vacio y el mejor_score es 0
15.             score1=sentidosMFS[i].rank;
16.             for(j=0;j<sentidosUKB.size();j++)
17.                     if(sentidosMFS[i].sentido==sentidosUKB[j].sentido)
18.                             score2=sentidosUKB[j].rank;
19.                             elminarSentido(sentidosUKB[j]); //Una vez tenemos el score del sentido
20.                                                             //lo eliminamos para efectos de optimización
21.                     Endif
22.                     for(k=0;k<sentidosSimLesk.size();k++)
23.                             if(sentidosMFS[i].sentido==sentidosSimLesk[k].sentido)
24.                                     score3=sentidosSimLesk[k].rank;
30.                                     elminarSentido(sentidosSimLesk[k]); //Una vez tenemos el score
31.                                                                         //lo eliminamos para efectos de optimización
32.                             Endif
33.                             for(h=0;h<sentidosAdapLesk.size();h++)
34.                                     if(sentidosMFS[i].sentido==sentidosAdapLesk[h].sentido)
35.                                             score4=sentidosSimLesk[k].rank;
36.                                             elminarSentido(sentidosAdapLesk[h]);
37.                                             //lo eliminamos para efectos de optimización
38.                                     Endif
39.                             Endfor
40.                     Endfor
41.             Endfor
42.             promedio=(score1+socre2+score3+score4)/4;
43.             sumatoria_minimo=(score1-promedio)² +(score2-promedio)² + (score3-promedio)² +(scor4-promedio)²;
44.             score_actual=promedio-(sumatoria_minimos/4);
45.             if(mejor_sentido.score<score_acutal)
46.                     mejor_sentido=sentidosMFS[i];
47.                     mejor_sentido.score=score_actual;
48.             Endif
49.     Endfor
50.     return mejor_sentido;
51.endFunction
```

Figura 4.4 *Algoritmo de filtrado de coeficientes para encontrar el mejor sentido de una palabra.*
Fuente: Elaboración Propia.

En la siguiente tabla se muestra un ejemplo de este filtrado de coeficientes para la palabra “pino” en la frase “Los conos de pino cuelgan en un árbol”.

Tabla 4-1 Filtrado de coeficientes para un sentido i.

Sentido			Método de desambiguación j					
			1	2	3	4		
i	**Definición**	**Identificador**	**Simplified Lesk**	**Adapted Lesk**	**MFS**	**UKB**	**Wi**	**∂Scorei**
1	un árbol de coníferas	SPA-11608250-n	0,01250	0,03750	0,03326	0,01642	0,02492	0,02481
2	cualquier árbol o arbusto gimnospermas con conos de rodamientos	SPA-13108841-n	0,01250	0,06250	0,00000	0,00000	0,01875	0,01809
3	el acto de apoyo a ti mismo por tus manos solas en una posición boca abajo	SPA-00436187-n	0,00000	0,01250	0,05847	0,01347	0,02111	0,02061
4	(gimnasia) un ejercicio diseñado para desarrollar y fuerza pantalla y la agilidad y el equilibrio (por lo general realizado con o en algunos aparatos de gimnasia)	SPA-00435778-n	0,00000	0,01250	0,00000	0,00000	0,00313	0,00310
5	recta de grano blanco y duradera a la madera resinosa menudo amarillenta de cualquiera de los numerosos árboles del género Pinus	SPA-11608885-n	0,01250	0,03750	0,00828	0,01420	0,01812	0,01799
6	la sustancia lignificada fibroso duro bajo la corteza de los árboles	SPA-15098161-n	0,01250	0,03750	0,00000	0,00000	0,01250	0,01227

Como se puede ver en la Tabla 4-1 el mejor sentido para la palabra “pino” en la frase “Los conos de pino cuelgan en un árbol” seria el sentido 1 (i=1) el cual tiene un $\partial Score_1 = 0{,}02481$ siendo este el máximo $\partial Score$.

Un vez que se tiene el mejor sentido para cada palabra que compone la definición del elemento constitutivo se procede a la capa de detección de errores sobre los elementos constitutivos, esta capa se expone a continuación.

4.3. Capa de validación de errores en elementos constitutivos

4.3.1. Detección de errores en elementos constitutivos.

Loa capa de detección de errores analiza los elementos constitutivos una vez son desambiguados por la capa de desambiguación su función consiste en aplicar un conjunto de reglas semánticas para detectar los elementos constitutivos inconsistentes, duplicados y no unificados, aunque estos errores fueron definidos en la Sección 2.4 a continuación se expone las reglas para identificar cada uno de los errores.

4.3.1.1. Detección de inconsistencias en elementos constitutivos

El problema de inconsistencia en elementos constitutivos fue expuesto en la sección 2.4.2 de la presente tesis, allí se presenta la definición de este problema. Para identificar inconsistencias entre elementos constitutivos se debe seguir la siguiente regla:

- Dos elementos constitutivos son inconsistentes si para un sintagma nominal con idéntico sentido se tiene un sentido de un verbo o un sentido de un adjetivo contradictorio u opuesto.

Formalmente se determina si una palabra en la definición de un elemento constitutivo es contradictoria a otra de la siguiente manera:

$$\exists\, Palabra_i \;\; en\; ECD1.definicion \wedge Palabra_j\; en\; ECD2.definicion \;\; \therefore$$
$$\left(Palabra_i.sentido \;=\; Palabra_j.sentido\right) \vee$$
$$(Palabra_i.sentido\; esAntonimoDe\; sinonimos(Palabra_j.sentido) \rightarrow$$
$$ECD1.inconsistente = true \wedge ECD2.inconsistente = true$$

Figura 4.5 *Detección formal de palabras contradictorias en elementos constitutivos. Fuente: Elaboración propia.*

4.3.1.2. Identificación de duplicidad en elementos constitutivos.

El problema de duplicidad en elementos constitutivos fue expuesto en la sección 2.4.3 de la presente tesis, allí se presenta la definición de este problema. Para la identificación de duplicidades en elementos constitutivos se sigue la siguiente regla:

- Dos elementos constitutivos son duplicados si para cada sentido de las palabras que componen el elemento constitutivo 1 se encuentran iguales sentidos en las palabras que componen el elemento constitutivo 2.

La regla anterior se define formalmente como:

$$\forall\, Palabra_i \;\; en\; ECD1.definicion \wedge Palabra_j\; en\; ECD2.definicion \;\; \therefore$$
$$\left(Palabra_i.sentido \;=\; Palabra_j.sentido\right) \vee$$
$$\rightarrow ECD1.duplicado = true \wedge ECD2.duplicado = true$$

Figura 4.6 *Regla formal para la detección de elementos constitutivos duplicados. Fuente: Elaboración propia.*

4.3.1.3. Identificación de no unificidad en elementos constitutivos

El problema de no unificidad en elementos constitutivos fue expuesto en la sección 2.4.4 de la presente tesis, allí se presenta la definición de este problema. Para la identificación de no unificidad en elementos constitutivos se sigue la siguiente regla:

- Dos elementos constitutivos están no unificados si existen palabras diferentes en la definición de los elementos constitutivos pero con un mismo sentido en ambos elementos constitutivos.

Esta regla se define formalmente como:

$$\exists\, Palabra_i \ en\ ECD1.definicion \wedge Palabra_j\ en\ ECD2.definicion \ \therefore$$
$$\left(Palabra_i.sentido = Palabra_j.sentido\right) \wedge Palabra_i.palabra \neq Palabra_j.palabra$$
$$\rightarrow ECD1.unificado = false \wedge ECD2.unificado = false$$

Figura 4.7 *Regla formal para la identificación de elementos constitutivos no unificados. Fuente: Elaboración propia.*

4.3.2. Corrección de errores en elementos constitutivos

Una vez se tienen identificados los errores de duplicidad, inconsistencia y no unificidad el modelo procede a realizar la validación y corrección de los errores con la intervención del usuario, de la siguiente manera:

En la primera fase se presentan las inconsistencias encontradas por el modelo en los elementos constitutivos, se tiene entonces que un elemento constitutivo puede ser inconsistente con <1> o <n> elementos constitutivos. Para una tupla de dos elementos constitutivos a_1 y a_2 inconsistentes entre sí, se considera que el sistema elimina la inconsistencia entre los dos elementos constitutivos al eliminar uno de ellos de la arquitectura de educción de requisitos. El usuario será quien decide que elemento constitutivo remover de la especificación.

Una vez se eliminan las inconsistencias en la arquitectura de educción de requisitos, se procede a presentar los elementos constitutivos duplicados, se tiene entonces que un elemento constitutivo puede ser duplicado <1> o <n> elementos constitutivos. Para una tupla de dos elementos constitutivos a_1 y a_2 duplicados, se considera que el sistema elimina la duplicidad entre los dos elementos constitutivos al eliminar uno de ellos de la arquitectura de educción de requisitos. El usuario será quien decide que elemento constitutivo remover de la especificación.

Posteriormente se presentan los elementos constitutivos no unificados, para los cuales el modelo procede a hacer la unificación de los elementos constitutivos con base a las palabras de uno de los elementos constitutivos duplicados. El usuario es quien elige con base a que elemento constitutivo se unificarán los demás elementos constitutivos.

Finalmente el modelo presenta un listado de los elementos constitutivos no inconsistentes, no duplicados y unificados al usuario.

4.4. Modelo de integración de propósito general usando wrappers.

Con el fin de integrar el modelo presentado, con cualquier sistema automático o manual de educción de requisitos que genere arquitecturas de educción de requisitos, en la presente tesis se propone un adaptador (wrapper) como modelo de integración de propósito general. Un adaptador es un mecanismo que permite la comunicación con otros modelos o sistemas externos y el modelo presentado. El adaptador provee una plantilla con la estructura de entrada que debe ser ingresada, y realiza el proceso de validación de las reglas de formación de documento que debe ser ingresado, adicionalmente es este modelo es el que realiza la validación de las reglas morfosintácticas presentadas en la sección 2.3 sobre los requisitos de software ingresados. Finalmente es este modelo quien envía los elementos validados al modelo de desambiguación de los sentidos de las palabras propuesto en las secciones anteriores. A continuación se presenta un esquema del modelo adaptador propuesto:

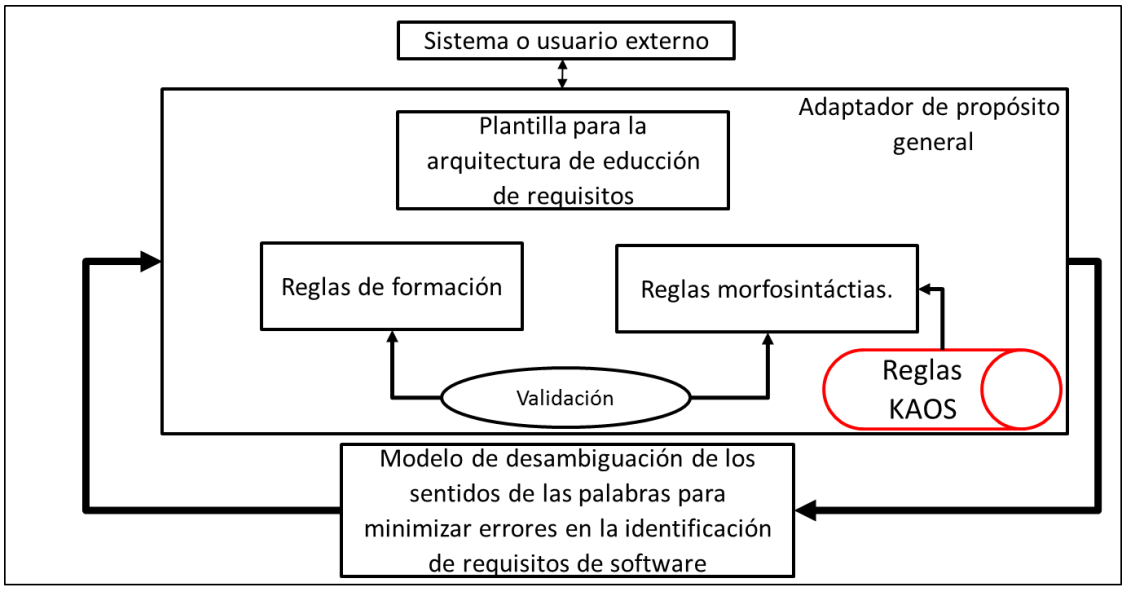


Figura 4.8. *Esquema del modelo adaptador (Wrapper) de propósito general.*

Fuente: Elaboración propia.

Este modelo provee al sistema consumidor la plantilla que debe ser instanciada para presentar los datos. Esta planilla define como debe ser presentada la arquitectura de educción de requisitos de software.

La plantilla de la arquitectura de educción de requisitos, la cual se encuentra definida mediante la siguiente tupla:

<nombre_arquitectura, categorías_elementos constitutivos, lista_elementos constitutivos>

Donde

- nombre_arquitectura representa el nombre de la arquitectura de educción de requisitos, por ejemplo KAOS, Entidad-Relación, Casos de uso, etc.
- Categorías elementos constitutivos representa los nombres de los grupos de elementos constitutivos que son admisibles en la arquitectura de educción de requisitos.
- lista_elementos constitutivos representa la lista de elementos constitutivos que serán procesados por el modelo propuesto.

Un elemento constitutivo se encuentra definido mediante la siguiente tupla

<tipo, definición, fuente>

Donde

- Tipo representa el nombre de la categoría del elemento constitutivo al cual pertenece, por ejemplo objetivo pertenece a la categoría objetivos, agente pertenece a la categoría agentes. Etc.
- Definición representa el conjunto de palabras que componen el elemento constitutivo.
- Fuente representa la frase requisito de donde se obtuvo el elemento constitutivo, por ejemplo “el sistema debe registrar los clientes” es la fuente del requisito de software “objetivo” y la definición del requisito es “registrar los clientes”.

Dado que requiere el modelo adaptador luego de que valida las reglas de formación de la entrada requiere la validación de reglas morfosintácticas que permitan determinar que elementos constitutivos están correctamente formados a partir de un requisito de software. Una regla bien formada se encuentra definida mediante la siguiente tupla:

<tipo, antecedente, consecuente>

Donde

- Tipo representa el tipo del elemento constitutivo que se busca identificar.
- Anteceden se define como el conjunto de etiquetas morfosintácticas ordenadas que se deben encontrar en la frase que origino el elemento constitutivo.
- Consecuente se define como el conjunto de elementos que una vez se encuentren en la frase fuente se convierten en componentes en el elemento constitutivo.

El conjunto de reglas morfosintácticas que son validadas en el modelo adaptador que concretamente se configuraron para éste modelo son las reglas morfosintácticas para identificar elementos constitutivos del modelo de KAOS presentadas en la sección 2.3 de la presente tesis.

Para realizar la validación de las reglas morfosintácticas el modelo adaptador utiliza los procesos morfosintácticos de etiquetado y segmentación expuestos en la Figura 4.2 *Algoritmo de segmentar y etiquetar.* para luego comparar las estructuras obtenidas con los antecedentes de las reglas en la fuente del elemento constitutivo, y con los consecuentes de las reglas en la definición del elemento constitutivo. A continuación se presenta el pseudocódigo del proceso de validación de las reglas morfosintácticas.

```
1.function validar_reglas (tipo,fuente, definicion) returns boolean
2.          reglas=cargar_reglas();  cargamos las reglas morfosintacticas asociadas a KAOS.
3.          palabras_definicion, palabras_fuente = segmentar_etiquetar(fuente, definicion);//Llamamos  a la función
4.          cumpleAntecedentes=false; tipo=null; contador=0;
5.          foreach ant in reglas.antecedentes.tipo
6.                      foreach ant.etiqueta in ant
7.                                  foreach pf in palabras_fuente
8.                                              if(pf.etiqueta = ant.etiqueta)
9.                                                          contador++
10.                                                         remove(pf); //eliminamos  el elemento para el que cumplio
11.                                                         remove(ant.etiqueta);
12.                                             Endif
13.                                 Endforeach
14.                     Endforeach
15.                     if(contador==ant.etiquetas)
16.                         //Se cumple el antecedente y se procede a verificar el consecuente
17.                                 cumpleAntecedente= true;
18.                                 cumpleConsecuente= validar_consecuente(palabras_definicion,reglas.consecuente[tipo])
19.                                 if(cumpleConsecuente, cumpleAntecedente)
20.                                             return true; //Si cumple el antecedente y el consecuente se cumple la regla y se
21.                                 Endif       //retorna verdadero a la validación.
22.                     Endforeach
23.         Endforeach
24.    return false; //Si nunca se devolvió verdadero la regla no se cumplio por tanto se devuelve falso.
25.endfunction
26.function validar_consecuente (palabras_definicion, consecuente)
27.         contador=0;             cumpleConsecuente=false;
28.         foreach con.etiqueta in consecuente                     //solo comparamos con los
29.                     foreach pd in palabras_definicion           //consecuentes de la regla
30.                                 if(pd.etiqueta = con.etiqueta)  //que cumplió el antecedente.
31.                                             contador++
32.                                             remove(pd);
33.                                             remove(con.remove);
34.                                 Endif
35.                     Endforeach
36.         Endforeach
37.         if(contador==ant.etiquetas)
38.                     cumpleConsecuente= true;
39.         if(cumpleConsecuente)
40.                     return true; //Si cumplio  el consecuente
41.         Endif
42.         return false;
43.endfunction
```

Figura 4.9 *Algoritmo de validación de reglas morfosintácticas asociadas a KAOS, en elementos constitutivos.*
Fuente: Elaboración propia.

4.5. Conclusiones

En este capítulo se presentó el modelo propuesto de desambiguación de los sentidos de las palabras usando filtrado de coeficientes que agrupa las técnicas de desambiguación basada en conocimiento que fueron presentadas en el capítulo 3. Se presentó una capa de detección de errores en elementos constitutivos la cual identifica inconsistencia,

duplicidad y no unificidad en elementos constitutivos aplicando reglas semánticas. Además se propuso el enfoque basado en la interacción con el usuario para la corrección de los errores identificados en la especificación.

Finalmente se presentó un modelo adaptador (Wrapper) para la integración del modelo de desambiguación de los sentidos de las palabras con cualquier arquitectura de educción de requisitos. Este modelo adaptador ofrece una plantilla para la configuración y validación de los elementos introducidos. Adicionalmente este modelo adaptador, valida las reglas morfosintácticas de KAOS presentadas en la sección 2.3 en los elementos constitutivos entrados.

Dicho lo anterior, con este capítulo se le pretende dar cumplimiento al tercer y cuarto objetivo del presente trabajo: “Definición de un modelo de resolución de ambigüedad de sentidos de palabras, con base a las técnicas seleccionadas en el objetivo anterior, que permita minimizar los errores ocasionados por la ambigüedad de los sentidos de las palabras, en un proceso de educción de requisitos de software”, El cuarto objetivo del presente trabajo “Definición de un modelo de integración de propósito general que permita la adición del modelo de resolución de ambigüedad de sentidos de palabras propuesto a una arquitectura de educción de requisitos de software”.

5. NEONPL: Un desambiguador semántico polisémico para la identificación de requisitos de software.

En este capítulo, se describe la aplicación de software, que ha sido implementada utilizando como fundamento la base teórica del marco de trabajo, presentado en los capítulos 2, 3 y 4. El objetivo general de la aplicación que aquí se define, es lograr la desambiguación semántica polisémica de elementos constitutivos. Particularmente se especifican la arquitectura del sistema desarrollado, al igual que su esquema funcional.

5.1. Arquitectura del sistema

La arquitectura general del sistema aquí desarrollado, llamado NEONPL, se presenta en la Figura 5.1. Esta arquitectura tiene como entrada un documento en XML con la arquitectura de educción de requisitos, siguiendo una plantilla XMLSchema proveída por el módulo adaptador.

El sistema NEONPL, procesa la información contenida en la entrada del sistema a lo largo de sus capas internas y arroja como salida del sistema, un documento en XML con la especificación no ambigua de los elementos constitutivos, en un lenguaje de especificación de requisitos de software, llamado KAOS.

El sistema NEONPL, se desarrolló bajo el lenguaje de programación JAVA, para los componentes de etiquetado, segmentación, desambiguación y detección de inconsistencias. Para los componentes web, se utilizó el lenguaje PHP con HTML y CSS, toda vez que estos, permiten enriquecer la experiencia de interacción con el usuario.

Se usó como motor de almacenamiento y consultas SQL, el servidor MySQL community edition.

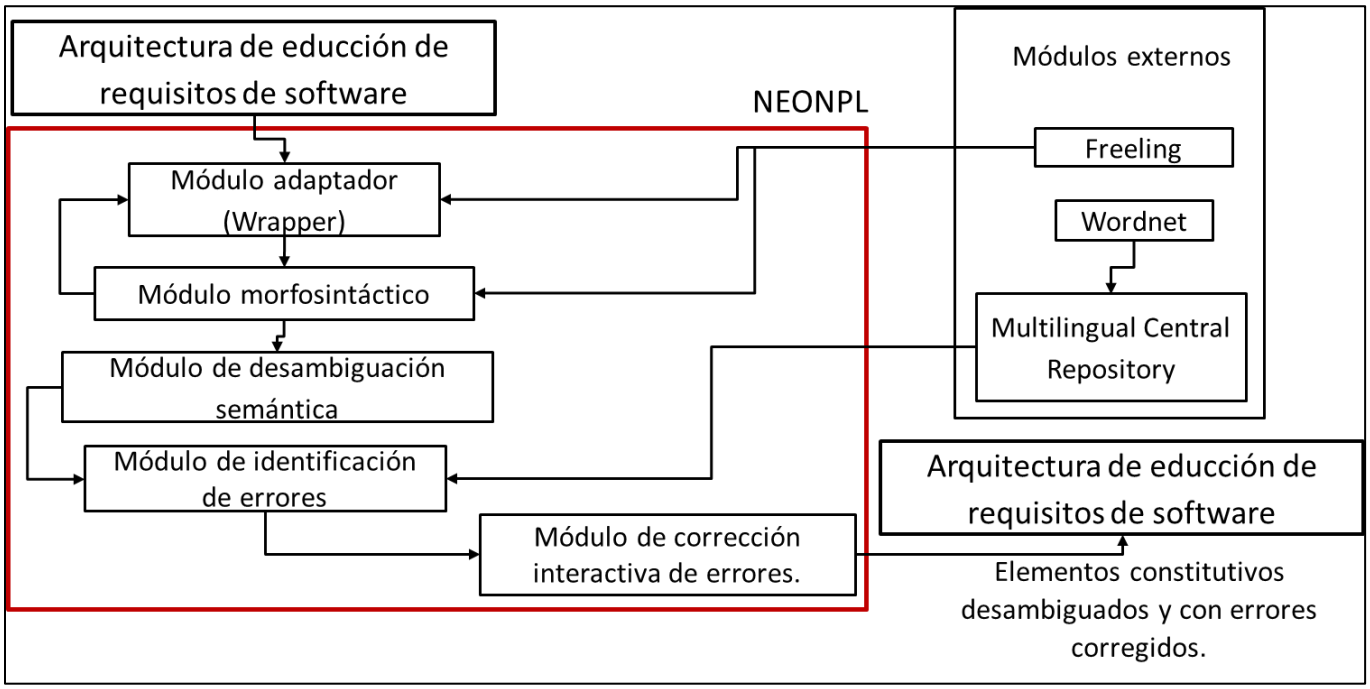


Figura 5.1 *Arquitectura general del modelo propuesto.*
Fuente: Elaboración Propia.

El sistema NEONPL, interactúa vía API con tres componentes externos Freeling, Wordnet y Wordnet en español mediante el multilingual central repository, los cuales se describen a continuación.

5.1.1. Freeling

El módulo NEONPL, invoca al paquete Freeling [78] (Freeling, es una Suite OpenSource para el análisis y procesamiento de textos multilingüe), con el fin de realizar los respectivos análisis morfosintácticos de segmentación, etiquetado y lematización expuestos en la Figura 4.2.

La arquitectura del módulo de Freeling, tiene un enfoque simple de dos capas cliente-servidor: Una capa para el servicio lingüístico que proporciona los servicios de análisis (morfológico, sintáctico, etiquetado, dependencias, entre otros), y una capa de aplicación que, haciendo las veces de cliente, invoca los servicios necesarios de los analizadores y usa los resultados de acuerdo con el objetivo de la aplicación [78]. Concretamente NEONPL, consume de freeling los siguientes servicios:

- Tokenización: Consiste en la división de acuerdo a espacios en blanco, de las palabras que definen un elemento constitutivo o la frase, por ejemplo “el banco rojo”, seria convertido a [EL] [BANCO] [ROJO].
- Etiquetado: Consiste en la asignación de una categoría gramatical a cada palabra tokenizada en el proceso anterior, las categorías gramaticales son leídas de una fuente de conocimiento léxico, que agrupa las palabras y les asigna la mejor categoría gramatical posible, dentro del árbol de taxonomías propio de la fuente de conocimiento. Las categorías léxicas son por ejemplo: Verbos, Sustantivos, Adjetivos, Determinantes, Etc., cada una de las cuales tiene una nomenclatura asociada.
- Lematización: Consiste en la extracción de la raíz de la palabra, es decir, llevar la palabra a su forma más básica entendible. Para esto, se recurre nuevamente a una fuente basada en conocimiento que contenga almacenada los lemas asociados a cada palabra y que provea el mejor lema asociado, por ejemplo, el lema de “recorrerá”, será “recorrer”.
- Fragmentación: Consiste en la asignación de una categoría sintáctica a las palabras o grupos de palabras, esto significa recurrir a las reglas mismas de formación del lenguaje por ejemplo “determinante + sustantivo” = Sintagma Nominal. Las fuentes léxicas basadas en conocimiento, extraen reglas de corpus de palabras previamente etiquetados, de manera que determinan, dado un conjunto de palabras, la mejor categoría correspondiente a las estructuras gramaticales.

De estas tareas básicas que se obtienen de freeling, se crean tareas más complejas dentro del módulo de NEONPL. A continuación se presenta lo porción de clases del paquete freeling usado por el sistema NEONPL:

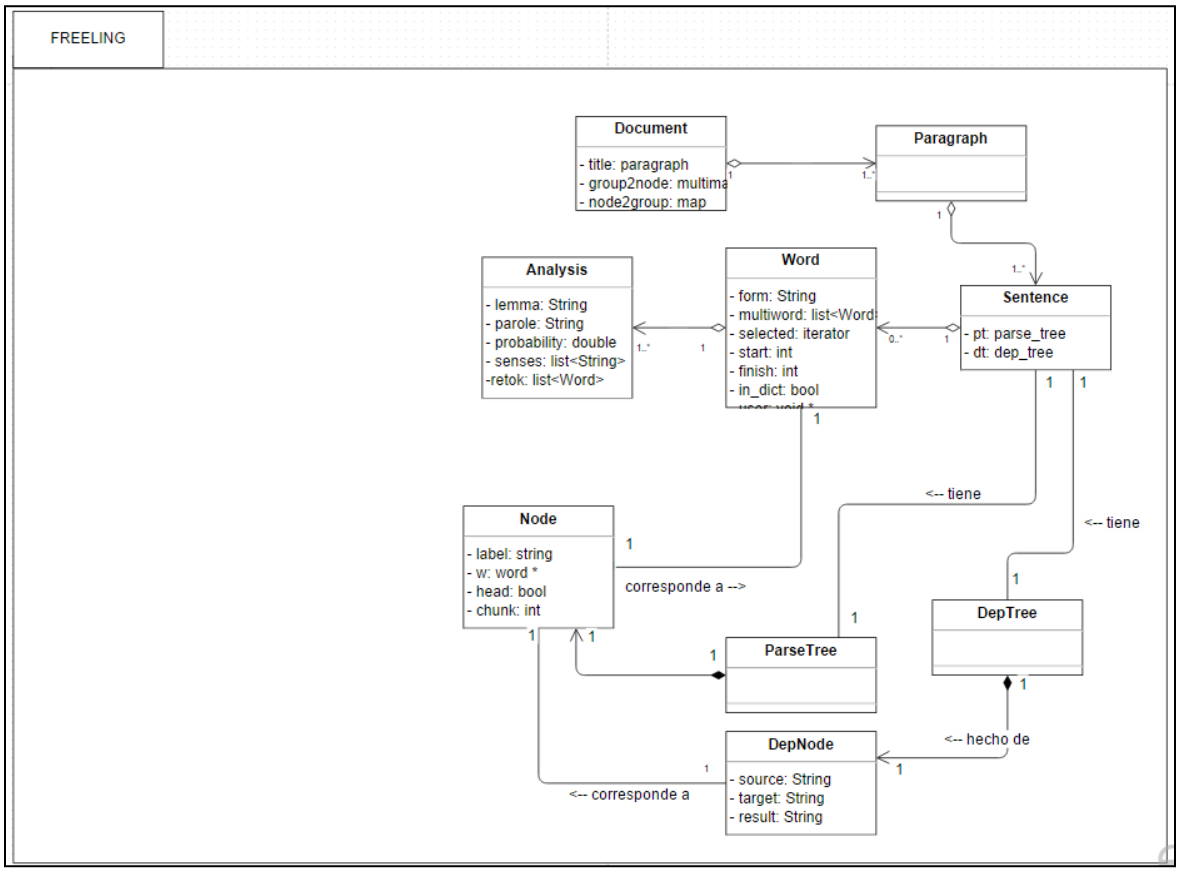


Figura 5.2 *Conjunto de clases del paquete freeling usado por el sistema NEONPL.*

5.1.2. Multilingual Central Repository y Wordnet.

Multilingual Central Repository, es una expansión multilingüe de la ontología Wordnet, es decir Wordnet se caracteriza por ser la base de conocimiento léxica más grande del mundo, sin embargo las definiciones, tags y relaciones semánticas están estructuradas para el idioma inglés, es por esto que se hace necesario el uso de Multilingual central Repository, ya que Multilingual central Repository contiene la traducción al español de gran cantidad de las definiciones y relaciones semánticas de Wordnet [46].

Las tareas de desambiguación basadas en conocimiento, requieren como su nombre lo indica, una fuente de conocimiento y es por esto que el sistema NEONPL requiere la utilización de las bases de datos léxicas como Multilingual Central Repository.

Concretamente Multilingual central repository es usado por el sistema NEONPL, como una base de datos relacional bajo el motor MySql, de la cual se extrae la siguiente información mediante un conjunto de consultas SQL:

- Sentidos por palabra. (Utilizado para la identificación de Duplicidades y no unificidad)
- Antónimos por palabra. (Utilizado para la identificación de Inconsistencias)
- Hiperónimos por sentido. (Utilizado para la identificación de Inconsistencias)
- Hipónimos por sentido. (Utilizado para la identificación de Inconsistencias)
- Definición por sentido. (Utilizado para describir el elemento constitutivo)
- Definiciones por palabra. (Utilizado para describir el elemento constitutivo)

A continuación se presentan el conjunto de tablas usadas por el sistema NEONPL, en la base de datos del multilingual central repository.

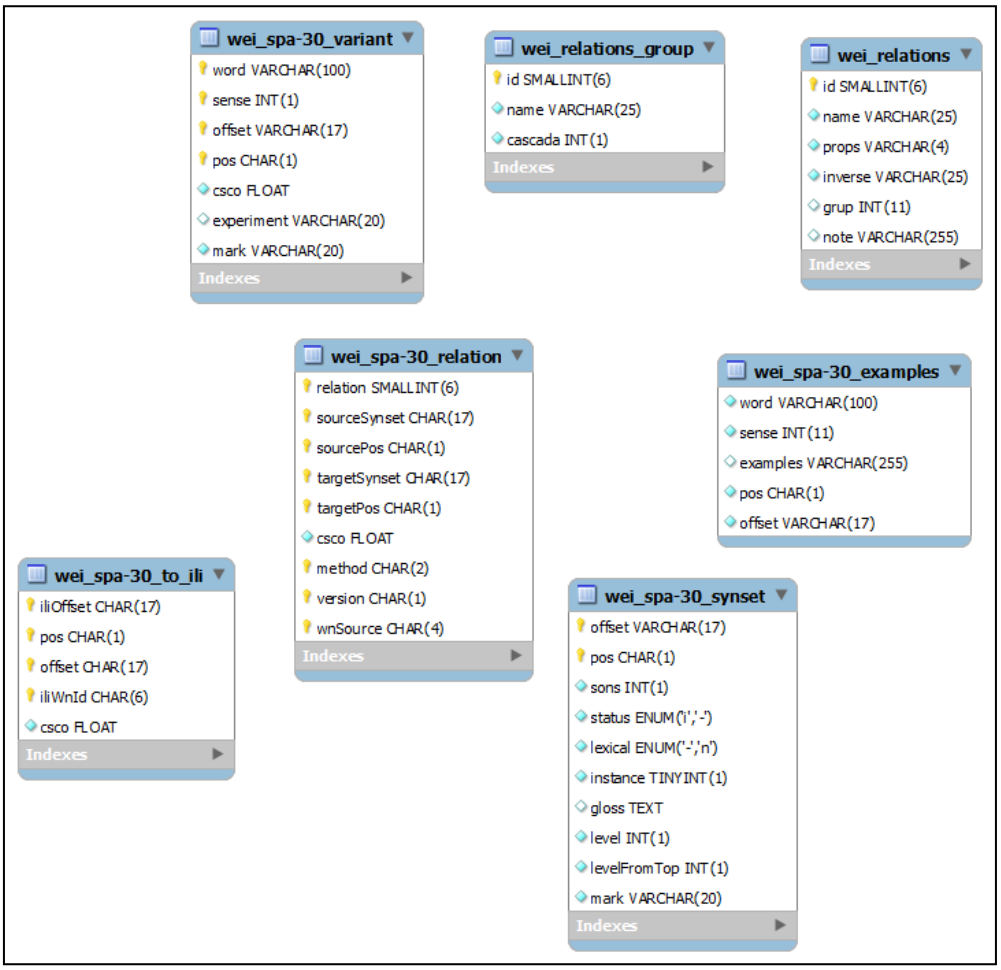


Figura 5.3 *Tablas de multilingual central repository usadas por el sistema NEONPL. Fuente: Elaboración propia.*

5.2. NEONPL, esquema funcional

El esquema funcional del sistema NEONPL, es un esquema modular que permite la separación excluyente de tareas, es decir la ejecución de las tareas de cada módulo, se hace de manera lineal e incremental ya que para realizar una tarea se hace necesario que las tareas inmediatamente anteriores, hayan sido realizadas. En este sentido por ejemplo, no es posible realizar una desambiguación sin que primero se hayan realizado las tareas de procesamiento morfosintáctico.

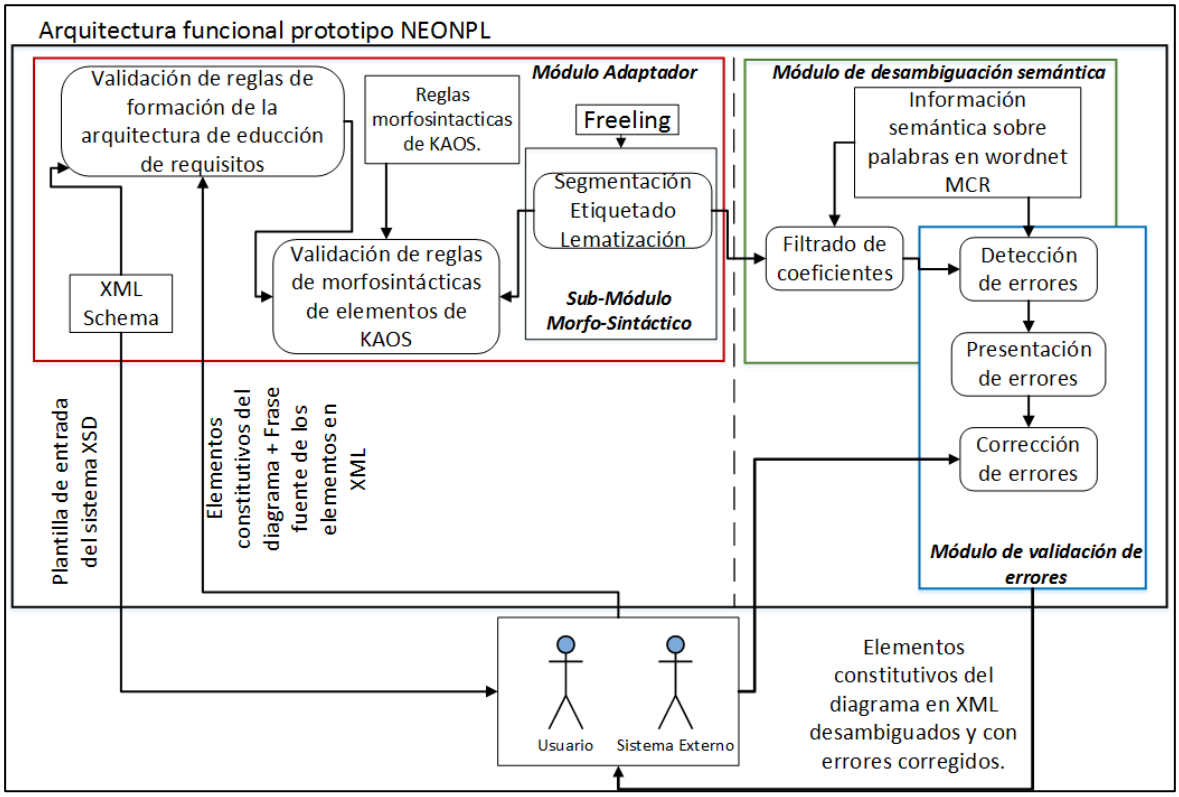


Figura 5.4 *Esquema funcional del sistema NEONPL.*
Fuente: Elaboración Propia.

Con el fin de proporcionar un mayor detalle de la funcionalidad y composición de cada módulo, a continuación se abordara cada uno de manera más detallada.

5.2.1. Módulo Adaptador

El módulo adaptador, provee al usuario o sistema consumidor, la plantilla formato para presentar los datos al sistema NEONPL, esta plantilla se encuentra definida en formato XML y tiene un XMLSchema asociado, ya que con base al XMLSchema, el sistema NEONPL, valida las reglas de formación del documento XML presentado. A continuación se presenta un ejemplo de una arquitectura de educción de requisitos presentada en el formato XML siguiendo la plantilla XMLSchema:

```
<?xml version="1.0" encoding="UTF-8"?>
<KAOS>
    <Objetivos>
        <objetivo id="1">
            <definicion>Registrar los clientes.</definicion>
            <fuente>El sistema tiene que registrar los clientes</fuente>
        </objetivo>
    </Objetivos>
    <Agentes>
        <agente id="2">
            <definicion>El sistema.</definicion>
            <fuente>El sistema tiene que registrar los clientes</fuente>
        </agente>
    </Agentes>
</KAOS>
```

Figura 5.5 *Ejemplo arquitectura de educción de requisitos presentada en formato XML siguiendo la plantilla de la figura 5.4.*
Fuente: Elaboración Propia.

XML Schema, es un lenguaje de esquema, utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa. A continuación se presenta el esquema XMLSchema, sobre el cual el módulo adaptador, valida que el documento XML se encuentre bien formado.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:annotation>
        <xsd:documentation xml:lang="es">
            Esquema arquitectura de educción de requisitos de software.
        </xsd:documentation>
    </xsd:annotation>
     <xsd:element name="KAOS">
         <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Objetivos" type="listaObjetivos" minOccurs="0"
maxOccurs="1"/>
                <xsd:element name="Agentes" type="listaAgentes" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="Operaciones" type="listaOperaciones" minOccurs="0"
maxOccurs="1"/>
                <xsd:element name="Entidades" type="listaEntidades" minOccurs="0"
maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
   </xsd:element>
   <xsd:complexType name="listaObjetivos">
        <xsd:sequence>
            <xsd:element name="objetivo" type="objetivo" minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="listaAgentes">
        <xsd:sequence>
            <xsd:element name="agente" type="agente" minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="listaOperaciones">
        <xsd:sequence>
            <xsd:element name="operacion" type="operacion" minOccurs="1"
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="listaEntidades">
        <xsd:sequence>
            <xsd:element name="entidad" type="entidad" minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="objetivo">
        <xsd:sequence>
            <xsd:element name="definicion" type="xsd:string"  minOccurs="1" maxOccurs="1"/>
            <xsd:element name="fuente" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:integer"/>
    </xsd:complexType>
    <xsd:complexType name="agente">
        <xsd:sequence>
            <xsd:element name="definicion" type="xsd:string" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="fuente" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:integer"/>
    </xsd:complexType>
    <xsd:complexType name="operacion">
        <xsd:sequence>
            <xsd:element name="definicion" type="xsd:string" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="fuente" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:integer"/>
    </xsd:complexType>
    <xsd:complexType name="entidad">
        <xsd:sequence>
            <xsd:element name="definicion" type="xsd:string" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="fuente" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:integer"/>
    </xsd:complexType>
</xsd:schema>
```

Figura 5.6 *XMLSchema de la arquitectura de educción de requisitos de software sobre la cual se valida el documento XML con la arquitectura de educción de requisitos. Fuente: Elaboración Propia.*

Este módulo provee también una interfaz gráfica tipo web, que permite subir el archivo XML, descargar la plantilla xsd, pero a su vez admite como servicio web, el archivo XML. La interfaz de este módulo adaptador es presentada en la siguiente figura.

Figura 5.7 *Interfaz del módulo adaptador para recibir la arquitectura de educción de requisitos de software en formato XML.*
Fuente: Elaboración propia.

Una vez se validan las reglas de formación de la arquitectura de educción de requisitos, el sistema procede a verificar las reglas morfosintácticas asociadas a los elementos constitutivos de KAOS. Las reglas se encuentran configuradas en una tabla SQL, llamada “rules (reglas)”, la cual contiene las reglas en términos de antecedente y consecuentes. A continuación se presenta una vista de la tabla reglas, con las reglas instanciadas presentadas en la sección 2.3, donde Verbo, es representado como /V/, Sustantivo, es representado como /N/, Adjetivo como /A/, Adverbio como /Q/ determinante, como /F/. El consecuente se presenta en términos del antecedente, es decir 0+1, significaría elemento 0 + elemento 1 del antecedente.

	id	tipo_artefacto	antecedente	description	concecuente
Editar Copiar Borrar	20	Agente	/F/+/N/+/V/+/F/	Agente	0+1
Editar Copiar Borrar	21	Objetivo	/F/+/N/+/V/+/V/+/F/+/V/+/F/+/F/+/N/	Regla R1 de Objetivos.	3+4+5+6+7+8
Editar Copiar Borrar	22	Objetivo	Que+/F/+/N/+/V/+/F/+/F/+/F/	Regla R2 para identificar Objetivos	3+4+5+6
Editar Copiar Borrar	23	Objetivo	/V/+que+/F/+/N/+/V/+/F/+/N/	Regla R3 para Identificar Objetivos	0+1+2+3+4+5+6
Editar Copiar Borrar	24	Objetivos	/F/+/N/+/V/+/F/+/V/+/F/+/N/	Regla R4 Para identificar objetivos	4+5+6
Editar Copiar Borrar	25	Agente	/F/+/N/+/V/+/V/+/F/+/V/+/F/+/F/+/N/	Regla R5 para identificar Agentes	1
Editar Copiar Borrar	26	Agente	Que+/F/+/N/+/V/+/F/+/F/+/F/	Regla R5 para identificar Agentes	2
Editar Copiar Borrar	27	Agente	/V/+que+/F/+/N/+/V/+/F/+/N/	Regla R5 para identificar Agentes	3
Editar Copiar Borrar	28	Agente	/F/+/N/+/V/+/F/+/V/+/F/+/N/	Regla R5 para identificar Agentes	1
Editar Copiar Borrar	29	Entidad	/N/+de+/N/	Regla R6 para identificar Entidades	2
Editar Copiar Borrar	30	Entidad	/F/+tiene+/N/	Regla R7 para identificar entidades	1
Editar Copiar Borrar	31	Operacion	/N/+/V/+/F/+/N/	Regla R8 para identificar Operaciones	1

Marcar todos *Para los elementos que están marcados:* Cambiar Borrar Exportar

Número de filas: 25

Figura 5.8 *Configuración de reglas morfosintácticas para la validación de los elementos constitutivos en la arquitectura de educción de requisitos.*

Fuente: Elaboración Propia.

Antes de realizar el proceso de validación de las reglas morfosintácticas, asociadas a los elementos constitutivos de KAOS, el modulo adaptador, realiza el proceso de segmentación y etiquetado, llamando a la función segmentar y etiquetar, la cual consume este servicio, desde en la API de freeling. El pseudocódigo fue presentado en la Figura 4.2.

Una vez se tiene segmentado y etiquetado la definición y la fuente que hacen parte del elemento constitutivo, este módulo procede a realizar la validación de las reglas morfosintácticas. El pseudocódigo del algoritmo que realiza la validación de las reglas, fue presentado en la Figura 4.9 del capítulo 4, sin embargo a continuación se presenta su implementación en el lenguaje JAVA.

```
public void setArtefactos(ListSentence ls, int artefactoid) {
        for (int i = 0; i < ls.size(); i++) {//Para cada oracion
            Sentence s = ls.get(i);
            for (int k = 0; k < reglas.size(); k++) {
                ArrayList<Proposicion> antecedente = reglas.get(k).getAntecedente();
                int contBuenos = 0;//Cuenta cuantos elementos cumplen la regla
                for (int l = 0; l < antecedente.size(); l++) {
                    for (int j = l; j < s.size(); j++) {//Para cada palabra
                        if (!antecedente.get(l).isPalabra()) {//Si es un tag
                            if (s.get(j).getTag().startsWith(antecedente.get(l).getTag())) {//Va
cumpiendo la regla
                                contBuenos++;
                                j = (int) s.size();//Para salir del ciclo
                            } else {//Ya no cumple la regla
                                j = (int) s.size();//Para salir del ciclo interno
                                l = (int) antecedente.size();//Para salir del ciclo externo
                            }
                        } else {//Si no es tag, es decir, si es una palabra
                            if (antecedente.get(l).getTag().equals("F") || antecedente.get(l).getTag
().equals(s.get(j).getLemma())) {//Va cumpiendo la regla
                                contBuenos++;
                                j = (int) s.size();//Para salir del ciclo
                            } else {//Ya no cumple la regla
                                j = (int) s.size();//Para salir del ciclo interno
                                l = (int) antecedente.size();//Para salir del ciclo externo
                            }
                        }
                    }
                }
                if (contBuenos == antecedente.size()) {//Si se cumplió la regla

                    ArrayList<Integer> consecuente = reglas.get(k).getConsecuente();
                    String texto = "";
                    for (int j = 0; j < consecuente.size(); j++) {
                        try {
                            texto += s.get(consecuente.get(j)).getLemma() + " ";
                        } catch (Exception e) {
                            System.out.println("Error");
                        }
                    }
                    ArrayList<WordS> palabras_definicion=artefactos.get
(artefactoid).getPalabras_definicion();
                    String def_total="";
                    for (int i2 = 0; i2 < palabras_definicion.size(); i2++) {
                        def_total+=palabras_definicion.get(i2).getLemma()+" ";
                    }
                    def_total = def_total.substring(0, def_total.length() - 3);

                    texto = texto.substring(0, texto.length() - 1);
                    if(texto.equals(def_total)){
                        artefactos.get(artefactoid).setReglas(true);
                    }
                    else
                    {
                        artefactos.get(artefactoid).setReglas(false);
                    }
                }
            }
        }
    }
```

Figura 5.9 *Algoritmo de Validación de reglas morfosintácticas implementado en JAVA. Fuente: Elaboración Propia.*

Este algoritmo lee las reglas de la base de datos de reglas y procede a realizar la validación vs cada elemento constitutivo. Una vez identifica que para un elemento constitutivo se cumple una regla, cambia la variable elemento constitutivo. Reglas a “true” pero en caso contrario elemento constitutivo.reglas = “false”.

Una vez se validan las reglas morfosintácticas, el módulo adaptador termina su trabajo procede a instanciar las clases elemento constitutivo y palabra (presentadas en el diagrama de clases de la

Figura **5.21** *Diagrama de clases del sistema NEONPL*.) y envía una señal a módulo de desambiguación, para que proceda a realizar el filtrado de coeficientes sobre las palabras de la fuente del elemento constitutivo ya etiquetadas.

5.2.2. Módulo de desambiguación de elementos constitutivos usando filtrado de coeficientes.

El módulo de desambiguación de filtrado de coeficientes recibe los elementos constitutivos y las palabras etiquetadas y procede a realizar la desambiguación. El módulo de desambiguación tiene implementado en el lenguaje JAVA, los algoritmos de desambiguación Lesk Simplificado, Lesk Adaptado, UKB y MFS y para cada palabra en la frase fuente del elemento constitutivo, procede a realizar la desambiguación en cada uno de estos algoritmos, para luego crear el vector de coeficientes de cada método de desambiguación y encontrar el mejor coeficiente entre estos sentidos. Es este el que presenta como el mejor sentido, mediante la ecuación (4) de la sección 4.2.2.

A continuación se presenta el código del algoritmo de filtrado de coeficientes implementado en el lenguaje JAVA.

```
public void filtrar_coeficientes() {
        int count = 0;
        for (int i = 0; i < palabras_fuente.size(); i++) {
            double score = 0;
            double max_filtro = 0;
            String best_sense = "";
            String spa0 = "Vacio";
            String def = "Vacio";
            String definicion_sentido = "Vacio";
            for (int j1 = 0; j1 < palabras_fuente.get(i).getMfs().size(); j1++) {
                double sumatoriaMinimos = 0;
                double minimo1 = 0, minimo2 = 0, minimo3 = 0, minimo4 = 0;
                score = palabras_fuente.get(i).getMfs().get(j1).rank;
                minimo1 = palabras_fuente.get(i).getMfs().get(j1).rank;
                spa0 = palabras_fuente.get(i).getMfs().get(j1).spa;
                def = palabras_fuente.get(i).getMfs().get(j1).definicion;
                for (int j2 = 0; j2 < palabras_fuente.get(i).getUkb().size(); j2++) {
                    if (palabras_fuente.get(i).getUkb().get(j2).spa.equals(spa0)) {
                        score += palabras_fuente.get(i).getUkb().get(j2).rank;
                        minimo2 = palabras_fuente.get(i).getUkb().get(j2).rank;
                    }
                }
                for (int j3 = 0; j3 < palabras_fuente.get(i).getSimlesk().size(); j3++) {
                    if (palabras_fuente.get(i).getSimlesk().get(j3).spa.equals(spa0)) {
                        score += palabras_fuente.get(i).getSimlesk().get(j3).rank;
                        minimo3 = palabras_fuente.get(i).getSimlesk().get(j3).rank;
                    }
                }
                for (int j4 = 0; j4 < palabras_fuente.get(i).getAdaplesk().size(); j4++) {
                    if (palabras_fuente.get(i).getAdaplesk().get(j4).spa.equals(spa0)) {
                        score += palabras_fuente.get(i).getAdaplesk().get(j4).rank;
                        minimo4 = palabras_fuente.get(i).getAdaplesk().get(j4).rank;
                    }
                }
                double promedio = score / 4;
                sumatoriaMinimos = Math.pow((minimo1 - promedio), 2) + Math.pow((minimo2 - promedio), 2)
+ Math.pow((minimo3 - promedio), 2) + Math.pow((minimo4 - promedio), 2);
                double filtro = promedio * (sumatoriaMinimos / 4);
                if (filtro > max_filtro) {
                    max_filtro = filtro;
                    best_sense = spa0;
                    definicion_sentido = def;
                    palabras_fuente.get(i).setDef_sentido(definicion_sentido);
                    palabras_fuente.get(i).setSentido(best_sense);
                }
            }
        }
        int max = palabras_definicion.size();
        count = 0;
        for (int i2 = 0; i2 < max; i2++) {
            for (int i = count; i < palabras_fuente.size(); i++) {
                if (palabras_definicion.get(i2).getLemma().equals(palabras_fuente.get(i).getLemma())) {
                    palabras_definicion.get(i2).setLemma(palabras_fuente.get(i).getLemma());
                    palabras_definicion.get(i2).setSentido(palabras_fuente.get(i).getSentido());
                    palabras_definicion.get(i2).setTag(palabras_fuente.get(i).getTag());
                    palabras_definicion.get(i2).setDef_sentido(palabras_fuente.get(i).getDef_sentido());
                    count = i;
                    i = palabras_fuente.size();
                }
            }

        }
    }
```

Figura 5.10 *Algoritmo de filtrado de coeficientes presentado en lenguaje JAVA. Fuente: Elaboración Propia.*

Una vez encuentra el mejor sentido para cada palabra, procede a asignar los sentidos a las palabras de la definición del elemento constitutivo, es decir a aquellas palabras que

se encuentran en la fuente del elemento constitutivo y en la definición del elemento constitutivo.

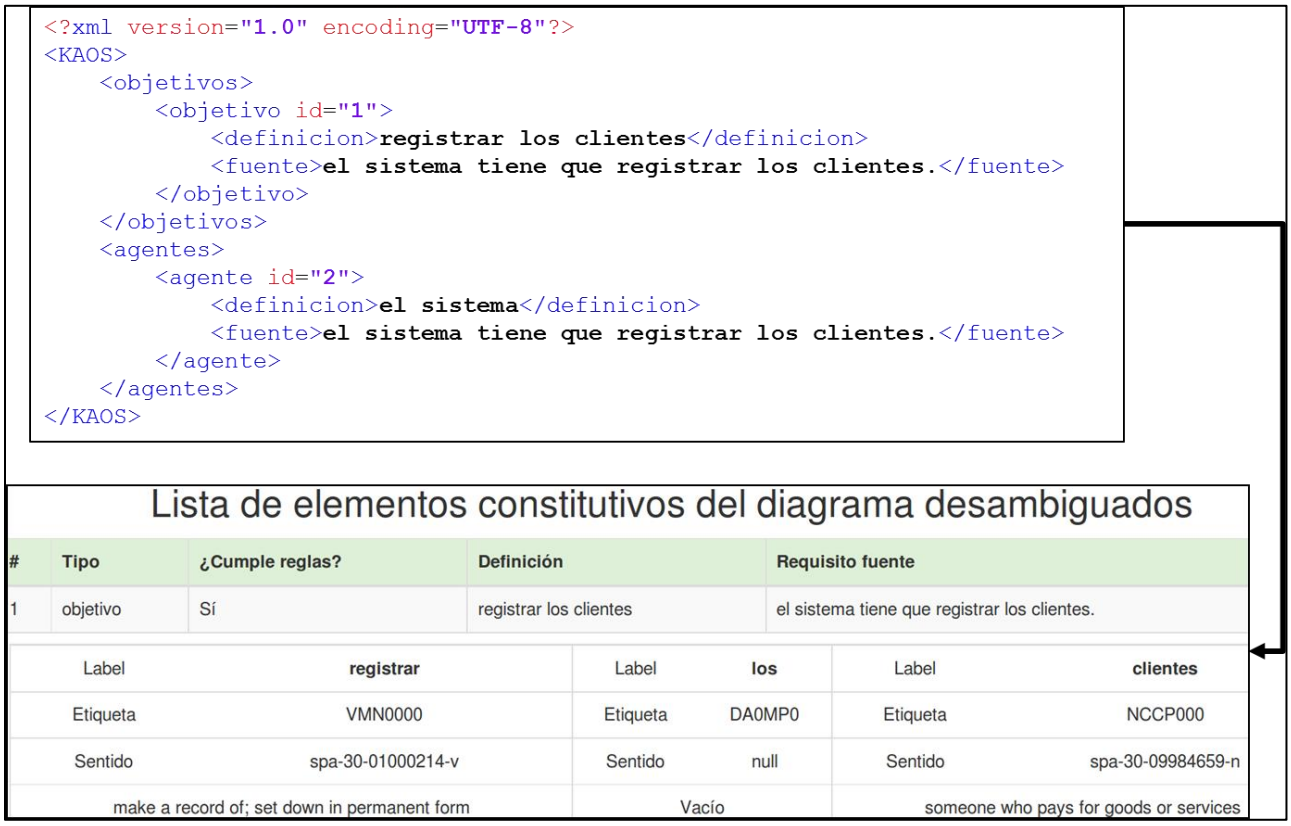
Finalmente el módulo desambiguador, indaga la definición de los sentidos en la base de datos de Wordnet para español, (Multilingual central Repository) y asigna esta definición en la definición del sentido. Para indagar en el multilingual central repository sobre la definición de un sentido, se utiliza la siguiente consulta SQL.

```
SELECT gloss FROM `wei_spa-30_synset` WHERE offset= sentido_palabra
```

Figura 5.11. *Consulta SQL para la extracción de la definición de un sentido asociado a una palabra.*
Fuente: Elaboración Propia.

Donde *sentido_palabra* representa el mejor sentido obtenido por el módulo de desambiguación de filtrado de coeficientes.

Una vez se realiza el proceso de desambiguación el módulo de desambiguación, inserta los sentidos de las palabras de la definición, en la base de datos NEONPL (el esquema entidad relación de la base de datos del sistema NEONPL, es presentado en la Figura 5.22 *Diagrama Entidad-Relación Implementado por el sistema NEONPL* y posteriormente le presenta los resultados obtenidos al usuario mediante una interfaz gráfica web. Un ejemplo de un archivo de entrada en XML su respectiva interfaz gráfica luego de ser desambiguado, se puede observar en la siguiente figura.



```
<?xml version="1.0" encoding="UTF-8"?>
<KAOS>
    <objetivos>
        <objetivo id="1">
            <definicion>registrar los clientes</definicion>
            <fuente>el sistema tiene que registrar los clientes.</fuente>
        </objetivo>
    </objetivos>
    <agentes>
        <agente id="2">
            <definicion>el sistema</definicion>
            <fuente>el sistema tiene que registrar los clientes.</fuente>
        </agente>
    </agentes>
</KAOS>
```

Lista de elementos constitutivos del diagrama desambiguados

#	Tipo	¿Cumple reglas?	Definición	Requisito fuente
1	objetivo	Sí	registrar los clientes	el sistema tiene que registrar los clientes.

Label	registrar	Label	los	Label	clientes
Etiqueta	VMN0000	Etiqueta	DA0MP0	Etiqueta	NCCP000
Sentido	spa-30-01000214-v	Sentido	null	Sentido	spa-30-09984659-n
make a record of; set down in permanent form		Vacío		someone who pays for goods or services	

Figura 5.12 *Interfaz gráfica de presentación de los elementos constitutivos desambiguados.*
Fuente: Elaboración Propia.

Posteriormente es enviada una señal al módulo de identificación de errores, para que proceda a realizar las consultas sobre las palabras ya desambiguadas y a detectar las inconsistencias, duplicidades y no unificidades en los elementos constitutivos. Esto se hace mediante tres botones presentados en la interfaz gráfica del usuario, una vez se presentan los resultados de la desambiguación.

5.2.3. Módulo de identificación de errores y corrección interactiva de errores con el usuario.

En primer lugar el módulo de detección de errores procede a la identificación de inconsistencias, este algoritmo compara por pares los elementos constitutivos dentro de la arquitectura de educción. El algoritmo implementado en lenguaje JAVA es el siguiente:

```
    public void setInconsistente(ArrayList<Artefacto> artefactos, int artefactoId) {
        this.inconsistente = false;
        for (int i = 0; i < artefactos.size(); i++) {
            if (artefactoId != i) {
                for (int j = 0; j < artefactos.get(i).getPalabras_definicion().size(); j++) {
                    for (int k = 0; k < this.palabras_definicion.size(); k++) {
                        if (this.esOpuesto(this.palabras_definicion.get(k).getSentido(), artefactos.get
(i).getPalabras_definicion().get(j).getSentido())) {
                            this.inconsistente = true;
                            j = artefactos.get(i).getPalabras_definicion().size();
                            k = this.palabras_definicion.size();
                            this.inconsistente__con += "," + artefactos.get(i).getId();
                        }
                    }
                }
            }
        }
    }

    public boolean esOpuesto(String spa1, String spa2) {
        try {
            mn.conectar();
            ArrayList<String> antonimos = mn.antonims(spa1);
            for (int i = 0; i < antonimos.size(); i++) {
                if (spa2.equals(antonimos.get(i))) {
                    return true;
                }
            }
            mn.desconectar();
        } catch (Exception ex) {

        }
        return false;
    }
```

Figura 5.13 *Algoritmo de detección de inconsistencias implementado en JAVA. Fuente: Elaboración Propia.*

Para cada palabra que compone la definición de los elementos constitutivos, extrae los antónimos de dicha palabra explotando la relación de antonimia existente entre sentidos, almacenada en el multilingual central repository. La consulta usada es la siguiente:

```
SELECT `targetSynset` FROM `wei_spa-30_relation` where (`relation`=34 or `relation`=12) and
    (`sourceSynset`= (SELECT `targetSynset` FROM `wei_spa-30_relation` where (`relation`=33) and
(`sourceSynset`= sentido_palabra) limit 1))
    UNION SELECT `targetSynset` FROM `wei_spa-30_relation` where (`relation`=33) and (`sourceSynset`=
sentido_palabra);
```

Figura 5.14 *Consulta SQL para la extracción de antónimos de un sentido. Fuente: Elaboración Propia.*

Donde sentido_palabra, es el sentido del cual se desean extraer los antónimos. La relación 34 se encuentra definida en multilingual central repository como antónimos cercanos, la relación 34, significa sinónimos cercanos, y la relación 12, significa hipóginos, por tanto la consulta indaga sobre los antónimos del sentido buscado, en los

antónimos de los sentidos sinónimos y en los antónimos de los sentidos hijos (Hipóginos).

Posteriormente se presenta al usuario los errores de inconsistencia identificados, es decir presenta al usuario las duplas de elementos constitutivos que son inconsistentes entre sí, para que el usuario elija cual remover desde una interfaz gráfica con el fin de eliminar la inconsistencia entre ellos.

La interfaz gráfica que se presenta al usuario para validar los errores de inconsistencia es la siguiente:

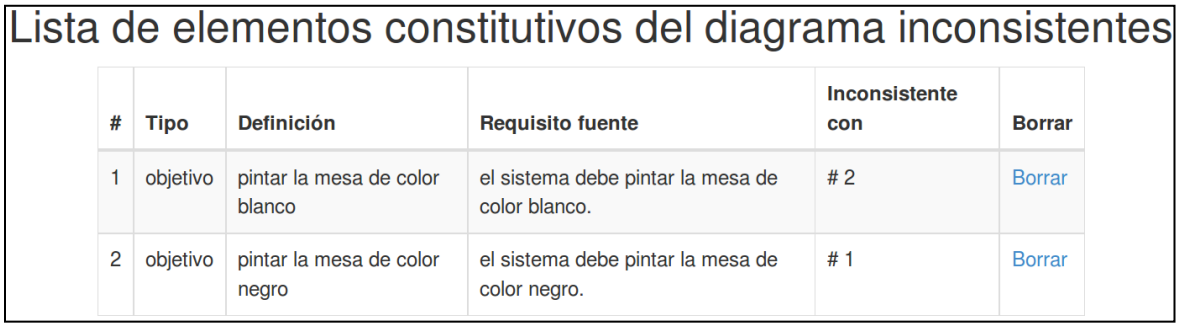


Figura 5.15 *Interfaz gráfica presentación de errores de inconsistencia. Fuente: Elaboración Propia.*

Luego de presentar y solucionar con el usuario los errores de inconsistencia, procede a detectar los errores de duplicidad de los elementos constitutivos. Para detectar los errores de duplicidad, el sistema utiliza las etiquetas almacenadas en la base de datos.

Se utiliza la comparación por pares de las definiciones de los elementos constitutivos, haciendo uso del siguiente algoritmo en el lenguaje PHP[4]:

[4] Para mostrar este algoritmo, se usó el lenguaje de programación PHP, con el fin de mostrar la versatilidad del software implementado. Adicionalmente se usó PHP, con el fin de enriquecer la experiencia basada en la web del usuario.

```php
<?php
public function detectar_duplicidades()
    {
        $artefactos=$this->general->artefactos();
        foreach ($artefactos as $a1) {
        $duplicado="";
            foreach ($artefactos as $a2) {
                if($a1->id!=$a2->id)
                {
                    $palabras1=$this->general->palabras_by_artefacto($a1->id);
                    $palabras2=$this->general->palabras_by_artefacto($a2->id);
                    if($this->contar_iguales($palabras1,$palabras2))
                    {
                        $duplicado.=$a2->id.",";
                        $duplicado=str_replace("null", "", $duplicado);
                        $this->general->duplicado($a1->id,$duplicado);
                    }
                }
            }
        }
        $artefactos=$this->general->artefactos_duplicados();
        $data["artefactos"]=$artefactos;
    }
    public function contar_iguales($p1,$p2)
    {
        $contador=0;
        for ($i=0; $i < count($p1); $i++) {
            for ($j=0; $j < count($p2); $j++) {
                if($p1[$i]->sentido==$p2[$j]->sentido)
                {
                    $contador++;
                    unset($p2[$j]);
                    $p2=array_values($p2);
                    break;
                }
            }
        }
        if($contador==count($p1))
        {
            return true;
        }
        return false;
    }
?>
```

Figura 5.16 *Algoritmo de detección de duplicidades en elementos constitutivos. Fuente: Elaboración Propia.*

Posteriormente se presenta al usuario los errores de duplicidad identificados, es decir presenta al usuario las duplas de elementos constitutivos que son duplicados entre sí, para que el usuario elija cual remover de desde una interfaz gráfica, con el fin de eliminar la duplicidad entre ellos.

La interfaz gráfica que se presenta al usuario para validar los errores de duplicidad es la siguiente:

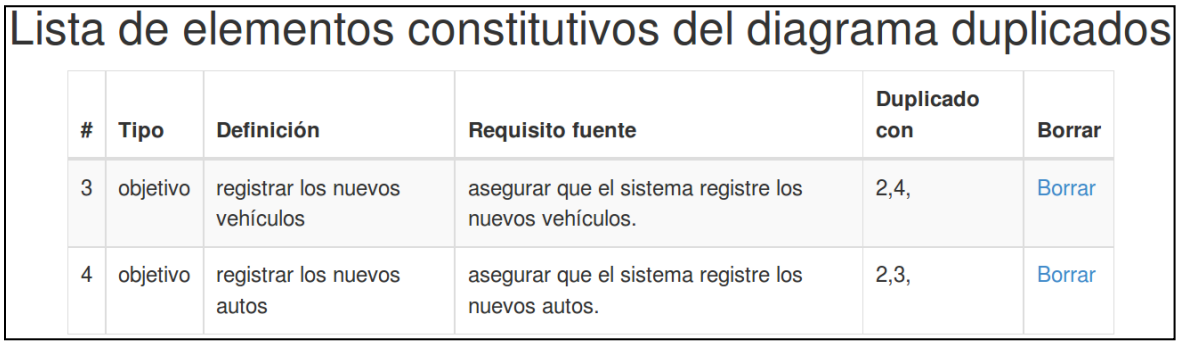


Figura 5.17. *Interfaz gráfica de presentación de errores de duplicidad. Fuente: Elaboración propia.*

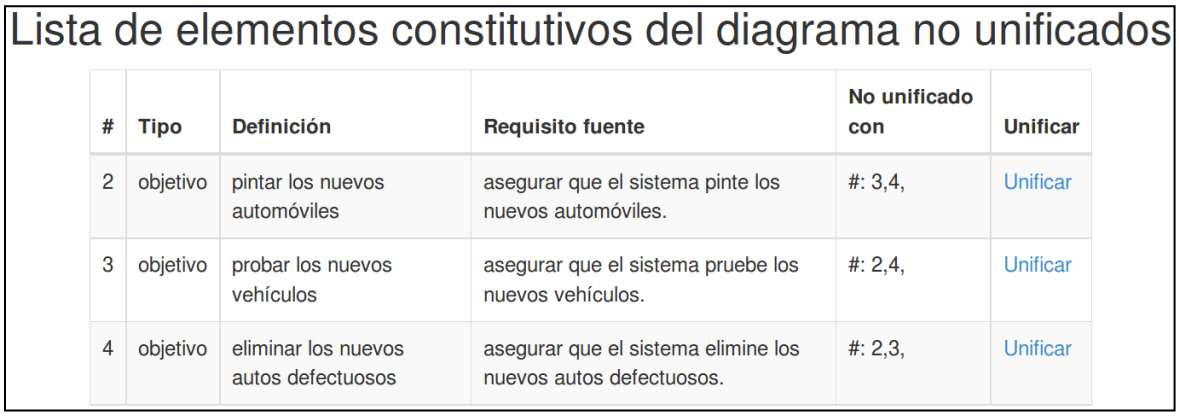
Finalmente luego de solucionar los errores de duplicidad procede a detectar los errores de no unificidad en los elementos constitutivos, mediante la comparación por pares, preguntando cuales sentidos de la definición del elemento constitutivo 1, se encuentran en la definición del elemento constitutivo 2, pero con una palabra diferente. El algoritmo usado para la detección de no unificidades es el siguiente:

```php
public function detectar_no_unificidades()
    {
        $artefactos=$this->general->artefactos();
        foreach ($artefactos as $a1) {
        $no_unificado="";
            foreach ($artefactos as $a2) {
                if($a1->id!=$a2->id)
                {
                    $palabras1=$this->general->palabras_by_artefacto($a1->id);
                    $palabras2=$this->general->palabras_by_artefacto($a2->id);
                    if($this->esta_no_unificado($palabras1,$palabras2))
                    {
                        $no_unificado.=$a2->id.",";
                        $no_unificado=str_replace("null", "", $no_unificado);
                        $this->general->no_unificado($a1->id,$no_unificado);
                    }
                }
            }
        }
        $artefactos=$this->general->artefactos_duplicados();
        $data["artefactos"]=$artefactos;
        $this->load->vars($data);
    }
    public function esta_no_unificado($p1,$p2)
    {
        $contador=0;
        for ($i=0; $i < count($p1); $i++) {
            for ($j=0; $j < count($p2); $j++) {
                if(
                    ($p1[$i]->sentido==$p2[$j]->sentido)
                     and
                     ($p1[$i]->palabra!=$p2[$j]->palabra)
                     and
                     ($p1[$i]->sentido!="null")
                   )
                {
                        return true;
                }
            }
        }
        return false;
    }
```

Figura 5.18. *Algoritmo para la detección de no unificidades en elementos constitutivos. Fuente: Elaboración propia.*

Finalmente se presenta al usuario los errores de no unificidad identificados, es decir, presenta al usuario las duplas de elementos constitutivos que son no unificados entre sí, para que el usuario elija con base a que elemento constitutivo desea unificar el/los demás elementos constitutivos no unificados.

La interfaz gráfica que se presenta al usuario para validar los errores de no unificidad, es la siguiente:



Lista de elementos constitutivos del diagrama no unificados

#	Tipo	Definición	Requisito fuente	No unificado con	Unificar
2	objetivo	pintar los nuevos automóviles	asegurar que el sistema pinte los nuevos automóviles.	#: 3,4,	Unificar
3	objetivo	probar los nuevos vehículos	asegurar que el sistema pruebe los nuevos vehículos.	#: 2,4,	Unificar
4	objetivo	eliminar los nuevos autos defectuosos	asegurar que el sistema elimine los nuevos autos defectuosos.	#: 2,3,	Unificar

Figura 5.19 *Interfaz para la presentación de errores de no unificidad al usuario. Fuente: Elaboración propia.*

Finalmente el sistema NEONPL retorna un documento en XML con la arquitectura de educción de requisitos completamente desambiguada y con los errores de inconsistencia, duplicidad y no unificidad eliminados de la especificación. El documento de salida sigue un formato como el que se muestra en la siguiente figura.

```
<?xml version="1.0" encoding="UTF-8"?>
<KAOS>
    <Objetivos>
        <objetivo id="1">
            <definicion>Registrar los carros.</definicion>
            <fuente>El sistema tiene que registrar los carros</fuente>
            <palabras_definicion>
                <palabra id="P1">
                    <palabra>Registrar</palabra>
                    <lema>Registrar</lema>
                    <etiqueta>VMSP3S0</etiqueta>
                    <sentido>spa-30-01000214-v</sentido>
                    <sentido_definicion>Hacer un registro de; Configurar
Permanentemente</sentido_definicion>
                </palabra>
                <palabra id="P2">
                    <palabra>los</palabra>
                    <lema>el</lema>
                    <etiqueta>DA0MP0</etiqueta>
                    <sentido>NULL</sentido>
                    <sentido_definicion>NULL</sentido_definicion>
                </palabra>
                <palabra id="P3">
                    <palabra>carros</palabra>
                    <lema>carro</lema>
                    <etiqueta>NCMP000</etiqueta>
                    <sentido>spa-30-02958343-n</sentido>
                    <sentido_definicion>Vehículo motorizado de cuatro ruedas por lo general
impulsado por un motor de combustión interna</sentido_definicion>
                </palabra>
            </palabras_definicion>
        </objetivo>
    </Objetivos>
    <Agentes>
        <agente id="2">
            <definicion>El sistema.</definicion>
            <fuente>El sistema tiene que registrar los carros</fuente>
            <palabras_definicion>
                <palabra id="P4">
                    <palabra>El</palabra>
                    <lema>el</lema>
                    <etiqueta>DA0MP0</etiqueta>
                    <sentido>NULL</sentido>
                    <sentido_definicion>NULL</sentido_definicion>
                </palabra>
                <palabra id="P5">
                    <palabra>sistema</palabra>
                    <lema>sistema</lema>
                    <etiqueta>NCMP000</etiqueta>
                    <sentido>spa-30-08435388-n</sentido>
                    <sentido_definicion>Un grupo de partes independientes interrelacionadas entre si
que funcionan como un todo</sentido_definicion>
                </palabra>
            </palabras_definicion>
        </agente>
    </Agentes>
</KAOS>
```

Figura 5.20 *Documento XML de salida del sistema NEONPL con la arquitectura de educción de requisitos desambiguada.*
Fuente: Elaboración propia.

5.3. Diagrama de Clases NEONPL

La clase Tseb, es el controlador principal de la aplicación. Éste se encarga de inicializar una interfaz que permite al usuario ingresar un documento en XML, con la arquitectura de educción de requisitos en formato XML. La función Wrapper de la clase Tseb, permite validar las reglas de formación del documento, luego este wrapper, instancia la clase Elemento constitutivo y asocia las palabras a cada elemento constitutivo mediante la clase WordS (Palabra semántica). Esto se hace usando los algoritmos de etiquetado y segmentación de la librería freeling, posteriormente sobre estos elementos constitutivos instanciados procede a la validación de las reglas morfosintácticas. La clase WordS posee información que será presentada al final del proceso de manera semántica: El sentido original que el usuario indica (de manera opcional), el sentido hallado por la aplicación, el label (o lema), la definición (o gloss), la etiqueta sintáctica (tag), la forma en que se presenta la palabra originalmente (form, o palabra sin lematización) y la forma más común de expresar, el mismo concepto en inglés.

La clase elemento constitutivo por su parte posee los atributos reglas, inconsistente_con, duplicado_con, no_unificado_con, estos atributos una vez se instancias son los que permiten presentar al usuario, los elemento constitutivo con errores entre sí.

A continuación se presenta el diagrama de clases de la aplicación NEONPL, que resume la descripción dada anteriormente:

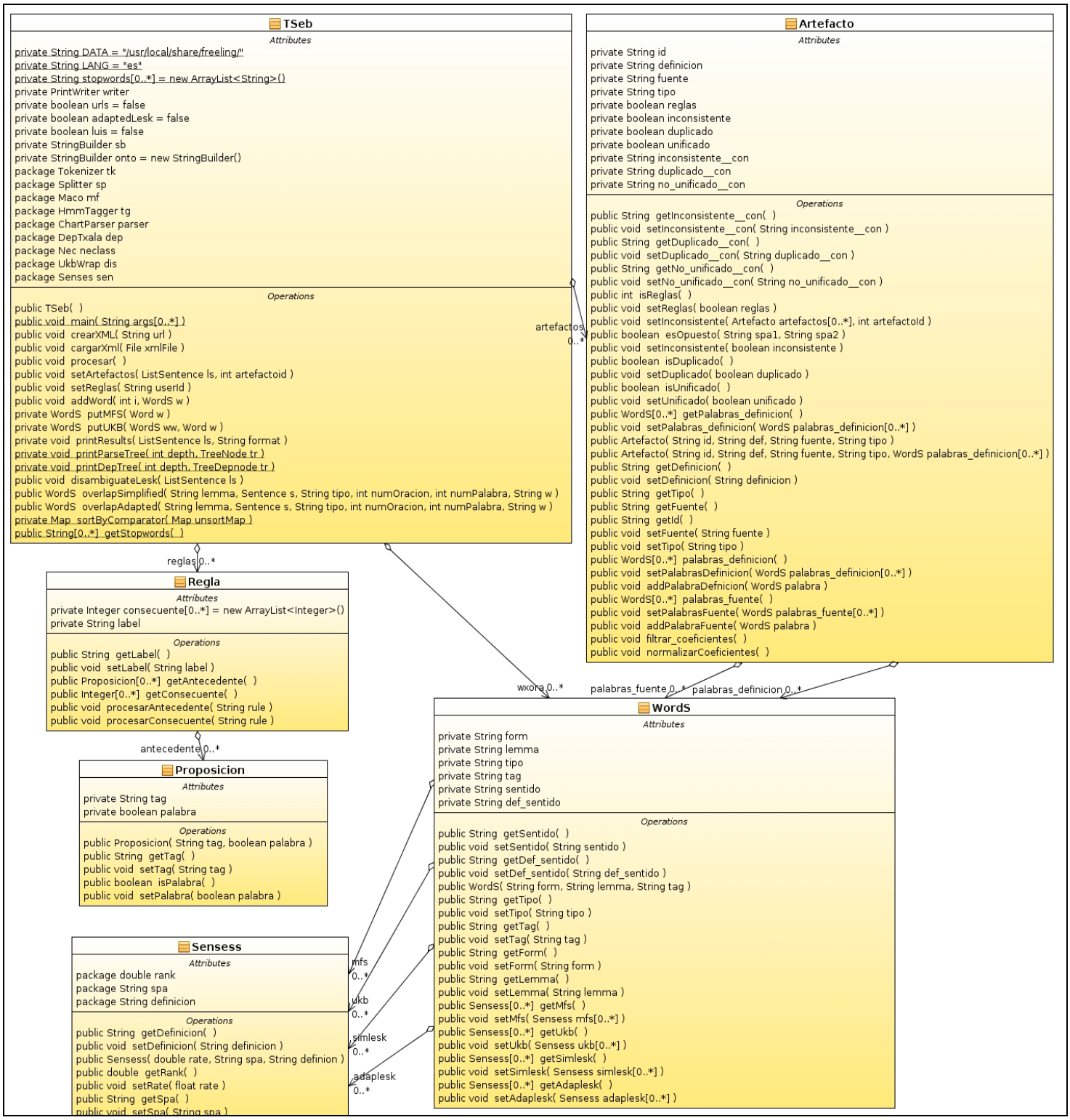


Figura 5.21 *Diagrama de clases del sistema NEONPL.*

Fuente: Elaboración propia.

Adicionalmente se presenta el modelo Entidad-Relación que implementa el sistema NEONPL.

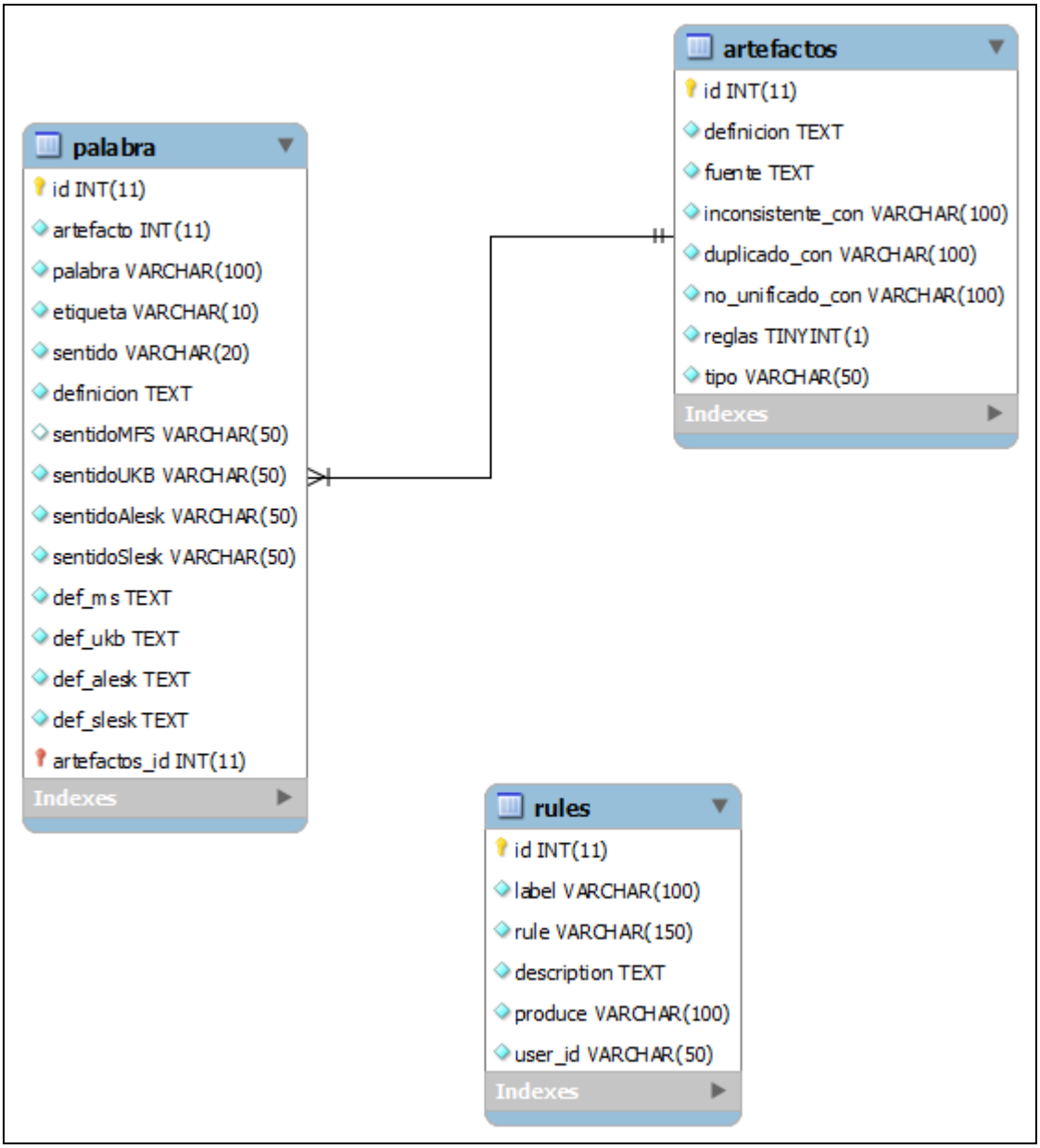


Figura 5.22 *Diagrama Entidad-Relación Implementado por el sistema NEONPL. Fuente: Elaboración propia.*

5.4. Esquema de Secuencias NEONPL

El proceso general de sistema NEONPL, inicia dándole la posibilidad al usuario de ingresar un documento con los elementos identificados por un sistema de educción de requisitos. Por tanto, se cuenta con una interfaz para que el usuario ingrese el documento XML con la especificación de requisitos: Este debe estar expresado como en la sección 5.1 del presente capítulo, toda vez que inicialmente el modulo adaptador realiza una validación de las reglas de formación del documento. Esta verificación se realiza en las funciones crearXml() y cargarXML().

Así, lo primero es que el usuario cargue el archivo y se presione el botón “Procesar”. El sistema comienza cargando los paquetes del procesador Freeling, que permiten realizar la segmentación y el etiquetado de las palabras que componen los elementos constitutivos. El resultado de este proceso es tener cada palabra etiquetada sintácticamente.

Para cada palabra de cada frase, se obtiene su definición, mediante una consulta en la base de datos. Con ésta, y las palabras del contexto, se realiza la desambiguación semántica Lesk simplificado.

Para la desambiguación con el Lesk adaptado, es necesario tener, no solo la definición de la palabra, sino de cada una de las palabras del contexto, junto con sus hiperónimos, hipóginos, homónimos y sinónimos. Se ejecuta esta consulta SQL. Luego se procede al filtrado de coeficientes y se obtiene el mejor sentido para cada palabra(WordS) que compone la definición del elemento constitutivo.

Con toda esta información, se crea una instancia de la clase WordS, indicándole el lema, el sentido que el sistema halló como el mejor, la etiqueta, el tipo de palabra (vacía o funcional), la palabra en inglés asociada al mismo sentido, la forma en que se presenta la palabra en su contexto, las demás palabras que se asocian al mismo concepto y la definición. Finalmente, se instancia un elemento constitutivo, con la lista de objetos de la clase Palabra.

Luego de esta primera parte, el sistema realiza la detección de inconsistencias, duplicidades y no unificidades para presentarle al usuario mediante interfaz gráfica, la posibilidad de corregir dichos errores.

A continuación se presenta el diagrama de secuencias de NEONPL en las siguientes figuras:

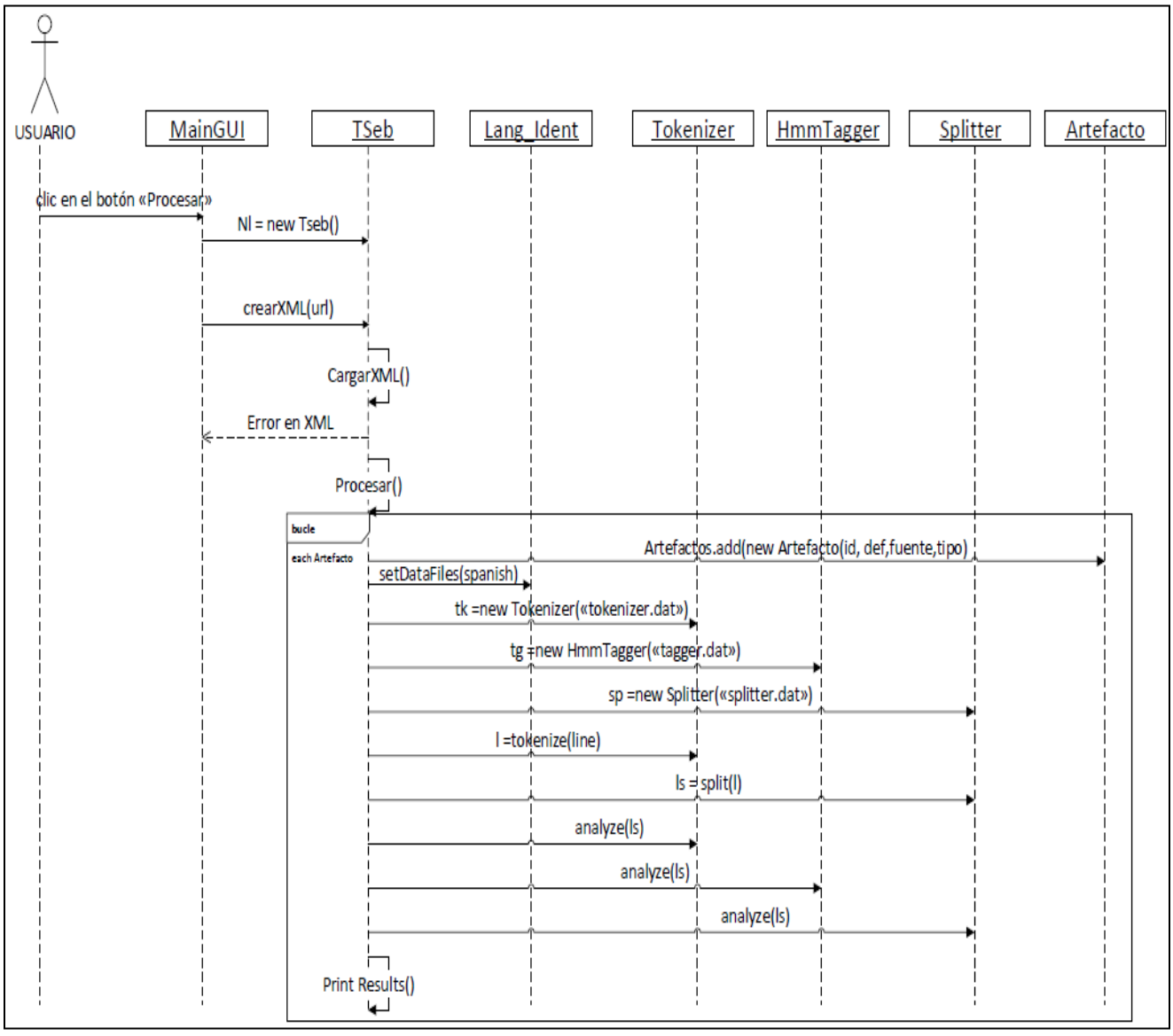


Figura 5.23 *Primera parte del esquema de secuencias del sistema NEONPL. Fuente: Elaboración Propia.*

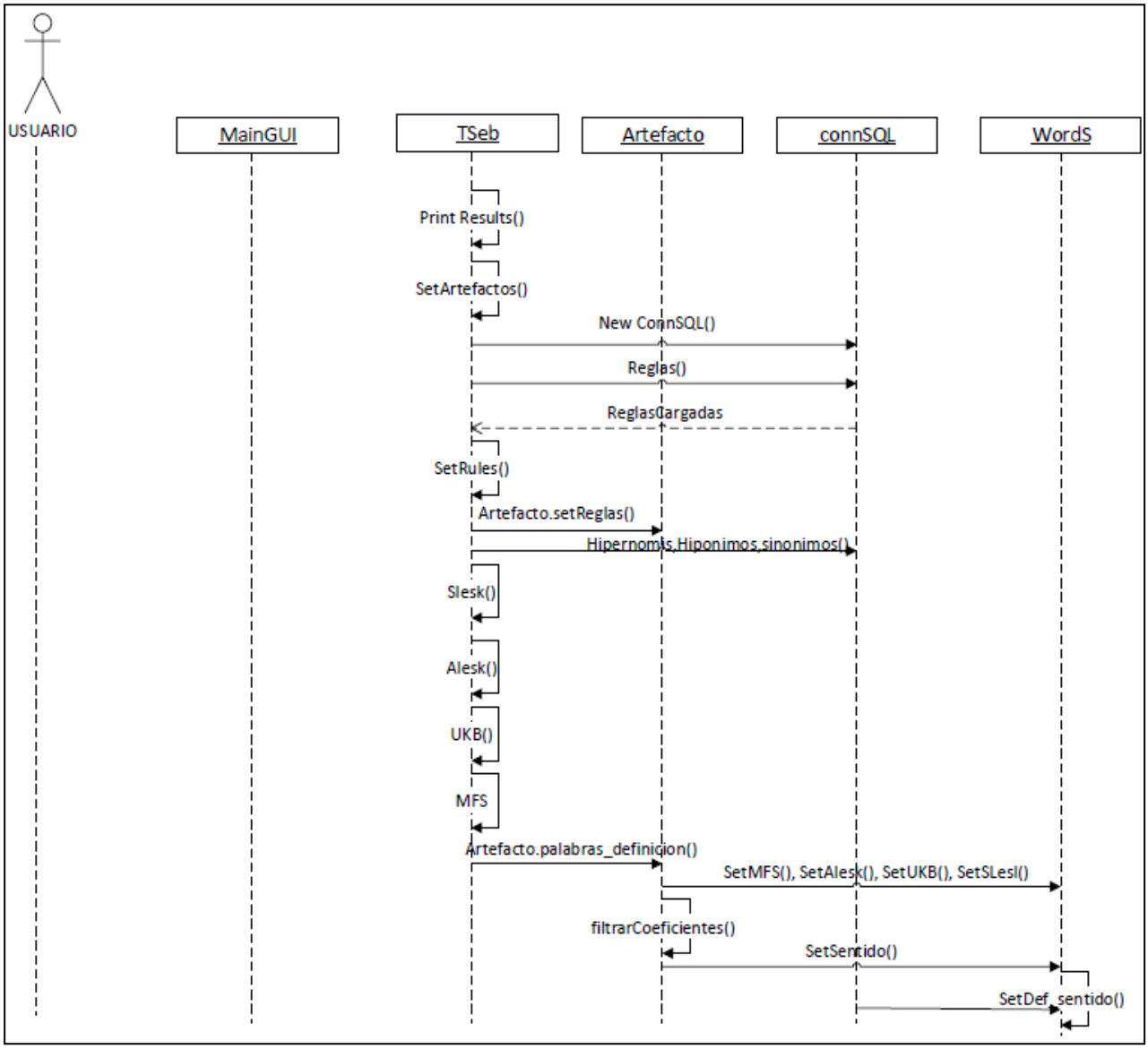


Figura 5.24 *Segunda parte del diagrama de secuencias del sistema NEONPL. Fuente: Elaboración Propia.*

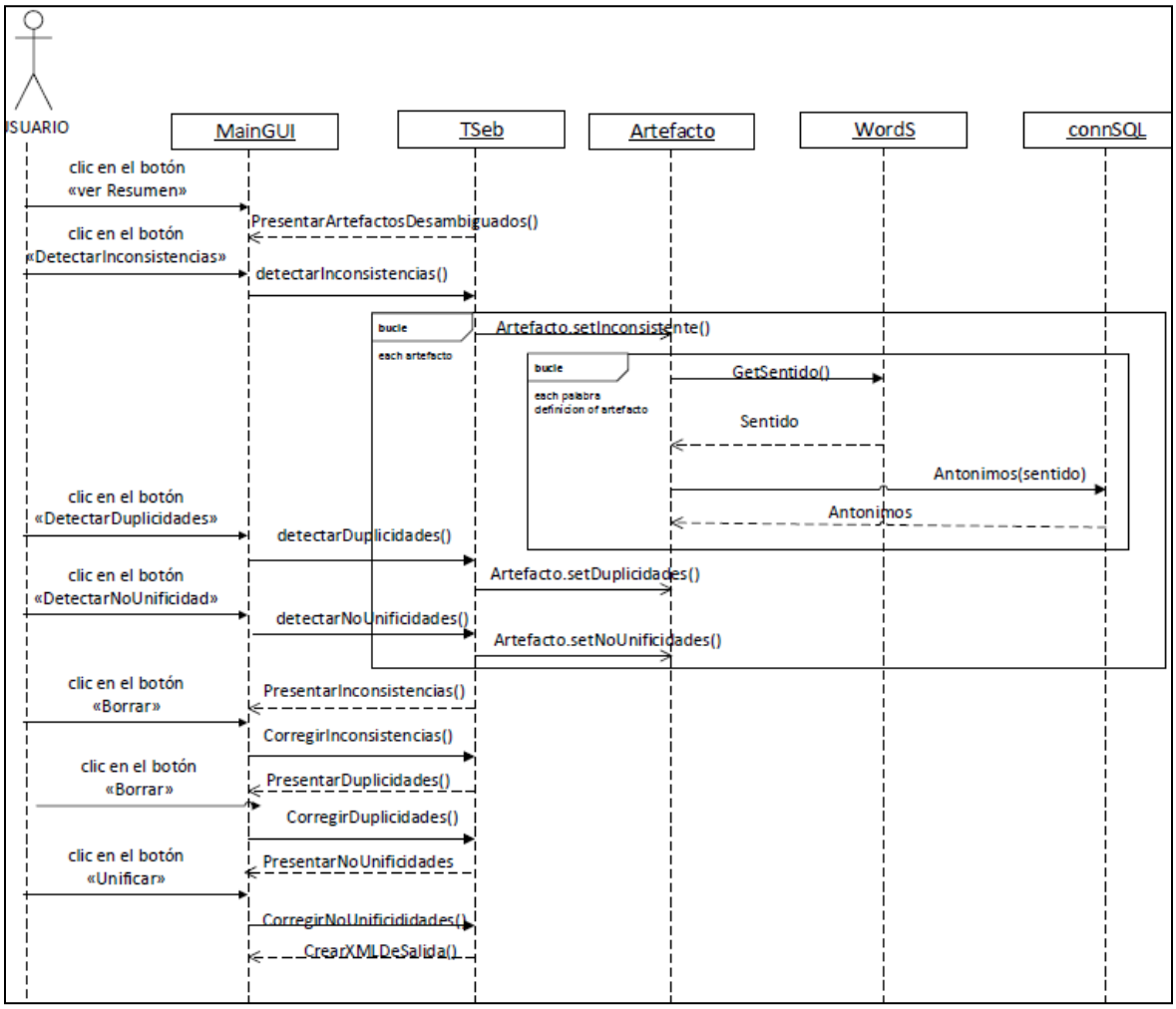


Figura 5.25 *Tercera parte del esquema de secuencias del sistema NEONPL*
Fuente: Elaboración Propia.

5.5. Conclusiones

En este capítulo se presentó el prototipo propuesto de tres módulos que integran el procesamiento morfosintáctico, wrapper de propósito general, el proceso de desambiguación de elementos constitutivos usando filtrado de coeficientes y las técnicas de identificación de errores en los elementos constitutivos.

Este sistema aplica las técnicas de desambiguación propuestas en el capítulo tres, y se implementa de manera programática el modelo teórico propuesto en el capítulo 4, finalmente se presentó el esquema funcional y todas sus características internas.

Con este capítulo se pretende dar cumplimiento al objetivo V de manera parcial, el cual consiste en “Validar el modelo de resolución de ambigüedad de sentidos de palabras y el modelo de integración de propósito general propuestos mediante la implementación de un ambiente de prueba que permita evaluar el desempeño de una arquitectura existente de educción de requisitos de software con y sin el uso de los modelos propuestos con el fin de validar sus fortalezas y debilidades”.

6. Pruebas y resultados

Este capítulo proporciona la evaluación, de los aspectos centrales de esta tesis, en una variedad de dominios de educción de requisitos, implementados para esta etapa. Los textos detallados de los casos de prueba utilizados se describen en el anexo A. Los resultados que se muestran en este capítulo se han obtenido utilizando como cliente-servidor, un computador de procesador Intel Core 2 Quad Q8200 (2.33 GHz), DDR2 3544 MB 800 MHz. Sistema Operativo Ubuntu 12LTS 32 Bits. Las tablas incluidas en la presente sección, relacionan los tiempos consumidos en cada proceso.

6.1. Evaluación de la desambiguación de los sentidos de las palabras en los elementos constitutivos.

En primer lugar se evalúa la desambiguación del módulo central de filtrado de coeficientes, concretamente se evalúa cobertura, precisión y recuerdo (estas métricas fueron presentadas en la sección 1.4.1.).

Para la evaluación se utilizaron los textos de los casos de estudio presentados en el anexo A. A continuación se presentan las definiciones usadas para realizar las pruebas.

Definición 6.1. Un elemento constitutivo se considera desambiguado si posee un sentido para cada palabra funcional que lo compone es decir si todos los adjetivos, verbos, sustantivos y adverbios poseen un sentido asociado.

Definición 6.2. Un elemento constitutivo se considera correctamente desambiguado si todos los sentidos de las palabras funcionales que lo componen son correctos según el usuario.

Definición 6.3 Un elemento constitutivo se considera incorrectamente desambiguado si al menos uno de los sentidos de las palabras funcionales que lo componen se encuentran incorrectamente desambiguadas según el usuario.

Definición 6.4 Un elemento constitutivo se considera no desambiguado si al menos uno de los sentidos de las palabras funcionales que lo componen no posee un sentido asociado.

Para los siguientes casos de estudio, su respectiva entrada se presenta en el anexo A, junto con las tablas extendidas de los resultados de salida.

6.1.1. Caso de estudio ambulancia

Para el caso de estudio de la ambulancia, presentado en el anexo A en la sección A.1., se obtuvieron los siguientes resultados en términos de la desambiguación de las palabras en los elementos constitutivos.

Tabla 6-1 ***Tabla general de resultados de desambiguación obtenidos para el caso de estudio de la ambulancia.***

# Elemento constitutivo	Tipo	Desambiguado	Correctamente Desambiguado	Incorrectamente Desambiguado	No desambiguado
1	Objetivo	SI	SI	NO	NO
2	Objetivo	SI	SI	NO	NO
3	Objetivo	SI	SI	NO	NO
4	Objetivo	SI	SI	NO	NO
5	Objetivo	SI	SI	NO	NO
6	Objetivo	NO	NA[5]	NA	SI
7	Objetivo	NO	NA	NA	SI
8	Objetivo	NO	NA	NA	SI
9	Objetivo	SI	SI	NO	NO
10	Objetivo	NO	NA	NA	SI
11	Objetivo	NO	NA	NA	SI

[5] La sigla NA significa No Aplica.

12	Objetivo	NO	NA	NA	SI
13	Objetivo	NO	NA	NA	SI
14	Objetivo	NO	NA	NA	SI
15	Objetivo	NO	NA	NA	SI
16	Objetivo	SI	SI	NO	NO
17	Objetivo	SI	SI	NO	NO
18	Agente	NO	NA	NA	SI
19	Agente	SI	SI	NO	NO
20	Agente	SI	SI	NA	NO
21	Agente	NO	NA	NA	SI
22	Entidad	SI	SI	NO	NO
23	Entidad	NO	NA	NA	SI

Con base a la tabla anterior se extrajeron las métricas de cobertura, precisión y recuerdo y son presentadas en la siguiente tabla.

Tabla 6-2 Resultado métricas evaluadas para el caso de estudio ambulancia.

$Cobertura_{WSD}$	0,47826087
$Precisión_{WSD}$	1,00
$Recuerdo_{WSD}$	0,48
$F1_{WSD}$	0,647058824

Totales	23
Desambiguados	11
Desambiguados Correctos	11
Desambiguados Incorrectos	0
No desambiguados	12

6.1.2. Caso de estudio ascensor

Para el caso de estudio del ascensor, presentado en el anexo A en la sección A.2., se obtuvieron los siguientes resultados en términos de la desambiguación del sentido de las palabras en los elementos constitutivos.

Tabla 6-3 ***Tabla general de resultados de desambiguación obtenidos para el caso de estudio del ascensor.***

# Elemento constitutivo	Tipo	Desambiguado	Correctamente Desambiguado	Incorrectamente Desambiguado	No desambiguado
1	Objetivo	SI	SI	NO	NO
2	Objetivo	SI	NO	SI	NO
3	Objetivo	SI	SI	NO	NO
4	Objetivo	SI	SI	NO	NO

5	Objetivo	SI	SI	NO	NO
6	Objetivo	SI	SI	NO	NO
7	Objetivo	SI	SI	NO	NO
8	Objetivo	SI	NO	SI	NO
9	Objetivo	SI	SI	NO	NO
10	Objetivo	SI	SI	NO	NO
11	Objetivo	NO	NA	NA	SI
12	Objetivo	NO	NA	NA	SI
13	Objetivo	NO	NA	NA	NO
14	Objetivo	SI	SI	NO	NO
15	Agente	SI	SI	NO	NO
16	Agente	SI	SI	NO	NO
17	Agente	SI	SI	NO	NO
18	Agente	SI	SI	NO	NO
19	Agente	SI	SI	NO	NO
20	Agente	SI	SI	NO	NO
21	Agente	SI	SI	NO	NO
22	Agente	SI	SI	NO	NO
23	Agente	SI	SI	NO	NO
24	Agente	SI	SI	NO	NO
26	Agente	SI	SI	NO	NO

Con base a la tabla anterior se extrajeron las métricas de cobertura, precisión y recuerdo y son presentadas en la siguiente tabla.

Tabla 6-4 Resultado métricas evaluadas para el caso de estudio ascensor.

$Cobertura_{WSD}$	0,846153846
$Precisión_{WSD}$	0,91
$Recuerdo_{WSD}$	0,77
$F1_{WSD}$	0,833333333

Totales	26
Desambiguados	22
Desambiguados Correctos	20
Desambiguados Incorrectos	2
No desambiguados	2

6.1.3. Caso de estudio Pizzería

Para el caso de estudio de la pizzería, presentado en el anexo A en la sección A.3., se obtuvieron los siguientes resultados en términos de la desambiguación del sentido de las palabras en los elementos constitutivos.

Tabla 6-5 ***Tabla general de resultados de desambiguación obtenidos para el caso de estudio de la pizzería.***

# Elemento constitutivo	Tipo	Desambiguado	Correctamente Desambiguado	Incorrectamente Desambiguado	No desambiguado
1	Objetivo	SI	SI	NO	NO
2	Objetivo	SI	SI	NO	NO
3	Objetivo	SI	SI	NO	NO
4	Objetivo	SI	SI	NO	NO
5	Objetivo	SI	SI	NO	NO
6	Objetivo	SI	SI	NO	NO
7	Objetivo	SI	SI	NO	NO
8	Objetivo	SI	NO	SI	NO
9	Objetivo	SI	SI	NO	NO
10	Objetivo	SI	SI	NO	NO
11	Agente	SI	SI	NA	SI
12	Agente	SI	SI	NA	SI
13	Agente	SI	SI	NA	NO
14	Agente	NO	NA	NA	SI
15	Agente	SI	SI	NO	NO
16	Agente	SI	SI	NO	NO
17	Agente	SI	SI	NO	NO
18	Agente	SI	SI	NO	NO
19	Agente	SI	SI	NO	NO
20	Agente	SI	SI	NO	NO

Con base a la tabla anterior se extrajeron las métricas de cobertura, precisión y recuerdo y son presentadas en la siguiente tabla.

Tabla 6-6 Resultado métricas evaluadas para el caso de estudio pizzería.

$Cobertura_{WSD}$	0,95	Totales	20
$Precisión_{WSD}$	0,95	Desambiguados	19
$Recuerdo_{WSD}$	0,90	Desambiguados Correctos	18

$F1_{WSD}$	0,923076923

Desambiguados Incorrectos	1
No desambiguados	3

6.1.4. Caso de estudio Carrefour

Para el caso de estudio de Carrefour, presentado en el anexo A en la sección A.4., se obtuvieron los siguientes resultados en términos de la desambiguación del sentido de las palabras en los elementos constitutivos.

Tabla 6-7 ***Tabla general de resultados de desambiguación obtenidos para el caso de estudio de Carrefour.***

# Elemento constitutivo	Tipo	Desambiguado	Correctamente Desambiguado	Incorrectamente Desambiguado	No desambiguado
1	Objetivo	NO	NA	NA	SI
2	Objetivo	SI	SI	NO	NO
3	Objetivo	SI	SI	NO	NO
4	Objetivo	SI	SI	NO	NO
5	Objetivo	SI	SI	NO	NO
6	Objetivo	SI	SI	NO	NO
7	Objetivo	SI	SI	NO	NO
8	Objetivo	SI	NO	SI	NO
9	Objetivo	SI	SI	NO	NO
10	Objetivo	SI	SI	NO	NO
11	Objetivo	SI	SI	NO	NO
12	Agente	SI	SI	NO	NO
13	Agente	SI	SI	NA	NO
14	Agente	SI	SI	NO	NO
15	Agente	NO	NA	NA	SI
16	Agente	SI	SI	NO	NO

Con base a la tabla anterior se extrajeron las métricas de cobertura, precisión y recuerdo y son presentadas en la siguiente tabla.

Tabla 6-8 Resultado métricas evaluadas para el caso de estudio de Carrefour.

$Cobertura_{WSD}$	0,875
$Precisión_{WSD}$	0,93

Totales	16
Desambiguados	14

$Recuerdo_{WSD}$	0,81
F1WSD	0,866666667

Desambiguados Correctos	13
Desambiguados Incorrectos	1
No desambiguados	2

6.1.5. Caso de estudio Colcerámicas.

Para el caso de estudio de Carrefour, presentado en el anexo A en la sección A.5., se obtuvieron los siguientes resultados en términos de la desambiguación del sentido de las palabras en los elementos constitutivos.

Tabla 6-9 ***Tabla general de resultados de desambiguación obtenidos para el caso de estudio del sistema de juegos.***

# Elemento constitutivo	Tipo	Desambiguado	Correctamente Desambiguado	Incorrectamente Desambiguado	No desambiguado
1	Objetivo	SI	SI	NO	NO
2	Objetivo	SI	SI	NO	NO
3	Objetivo	NO	NA	NA	SI
4	Objetivo	NO	NA	NA	SI
5	Objetivo	SI	SI	NO	NO
6	Objetivo	SI	SI	NO	NO
7	Objetivo	SI	SI	NO	NO
8	Objetivo	SI	NO	SI	NO
9	Objetivo	SI	SI	NO	NO
10	Objetivo	SI	SI	NO	NO
11	Objetivo	SI	SI	NO	NO
12	Objetivo	SI	SI	NO	NO
13	Agente	SI	SI	NA	NO
14	Agente	SI	SI	NO	NO
15	Agente	NO	NA	NA	SI
16	Agente	SI	SI	NO	NO
17	Agente	SI	NO	SI	NO
18	Agente	SI	NO	SI	NO
19	Agente	SI	SI	NO	NO
20	Agente	SI	SI	NO	NO

Con base a la tabla anterior se extrajeron las métricas de cobertura, precisión y recuerdo y son presentadas en la siguiente tabla.

Tabla 6-10 Resultado métricas evaluadas para el caso de estudio de Colcerámicas.

$Cobertura_{WSD}$	0,85
$Precisión_{WSD}$	0,82
$Recuerdo_{WSD}$	0,70
$F1_{WSD}$	0,756756757

Totales	20
Desambiguados	17
Desambiguados Correctos	14
Desambiguados Incorrectos	3
No desambiguados	3

6.1.6. Caso de estudio aseguradora

Para el caso de estudio de la aseguradora, presentado en el anexo A en la sección A.6., se obtuvieron los siguientes resultados en términos de la desambiguación del sentido de las palabras en los elementos constitutivos.

Tabla 6-11 ***Tabla general de resultados de desambiguación obtenidos para el caso de estudio de la aseguradora***

# Elemento constitutivo	Tipo	Desambiguado	Correctamente Desambiguado	Incorrectamente Desambiguado	No desambiguado
1	Objetivo	SI	SI	NO	NO
2	Objetivo	SI	SI	NO	NO
3	Objetivo	SI	SI	NO	NO
4	Objetivo	SI	SI	NO	NO
5	Objetivo	SI	SI	NO	NO
6	Objetivo	SI	SI	NO	NO
7	Objetivo	SI	SI	NO	NO
8	Objetivo	SI	SI	NO	NO
9	Objetivo	SI	SI	NO	NO
10	Objetivo	SI	SI	NO	NO
11	Agente	SI	SI	NO	NO

Con base a la tabla anterior se extrajeron las métricas de cobertura, precisión y recuerdo y son presentadas en la siguiente tabla.

Tabla 6-12 Resultado métricas evaluadas para el caso de estudio de Aseguradora.

$Cobertura_{WSD}$	1
$Precisión_{WSD}$	1,00
$Recuerdo_{WSD}$	1,00

Totales	11
Desambiguados	11
Desambiguados Correctos	11

$F1_{WSD}$	1

Desambiguados Incorrectos	0
No desambiguados	0

6.1.7. Caso de estudio sistema de Juegos.

Para el caso de estudio del sistema de juegos, presentado en el anexo A en la sección A.7., se obtuvieron los siguientes resultados en términos de la desambiguación del sentido de las palabras en los elementos constitutivos.

Tabla 6-13 ***Tabla general de resultados de desambiguación obtenidos para el caso de estudio del sistema de juegos.***

# Elemento constitutivo	Tipo	Desambiguado	Correctamente Desambiguado	Incorrectamente Desambiguado	No desambiguado
1	Objetivo	SI	SI	NO	NO
2	Objetivo	SI	SI	NO	NO
3	Objetivo	SI	SI	NO	NO
4	Objetivo	NO	NA	NA	SI
5	Objetivo	SI	SI	NO	NO
6	Objetivo	SI	SI	NO	NO
7	Objetivo	SI	SI	NO	NO
8	Objetivo	SI	SI	NO	NO
9	Objetivo	SI	SI	NO	NO
10	Objetivo	SI	SI	NO	NO
11	Objetivo	SI	SI	NO	NO
12	Objetivo	SI	SI	NO	NO

Con base a la tabla anterior se extrajeron las métricas de cobertura, precisión y recuerdo y son presentadas en la siguiente tabla.

Tabla 6-14 Resultado métricas evaluadas para el caso de estudio de Aseguradora.

$Cobertura_{WSD}$	0,916666667
$Precisión_{WSD}$	1,00
$Recuerdo_{WSD}$	0,92
$F1_{WSD}$	0,956521739

Totales	12
Desambiguados	11
Desambiguados Correctos	11
Desambiguados Incorrectos	0
No desambiguados	1

6.2. Resumen de resultados de desambiguación de los sentidos de las palabras sobre los casos de estudio.

La siguiente tabla agrupa los resultados de precisión cobertura y recuerdo obtenidos para los casos de estudio presentados en la sección anterior.

Tabla 6-15 Resumen resultados por casos de estudio.

	Ambulancia	Ascensor	Pizzería	Carrefour	Colcerámicas	Aseguradora	Juego	Promedio
Cobertura$_{WSD}$	0,4783	0,8462	0,9500	0,8750	0,8500	1,0000	0,9167	0,8452
Precisión$_{WSD}$	1,0000	0,9091	0,9474	0,9286	0,8235	1,0000	1,0000	0,9441
Recuerdo$_{WSD}$	0,4783	0,7692	0,9000	0,8125	0,7000	1,0000	0,9167	0,7967
F1$_{WSD}$	0,6471	0,8333	0,9231	0,8667	0,7568	1,0000	0,9565	0,8548
Elementos constitutivos	23	26	20	16	20	11	12	

6.2.1. Análisis de Cobertura

En las siguientes gráficas se presenta la gráfica que describe la cobertura obtenida por el sistema sobre los casos de estudio presentados en la sección anterior y se presenta la cobertura versus el número de elementos constitutivos introducidos.

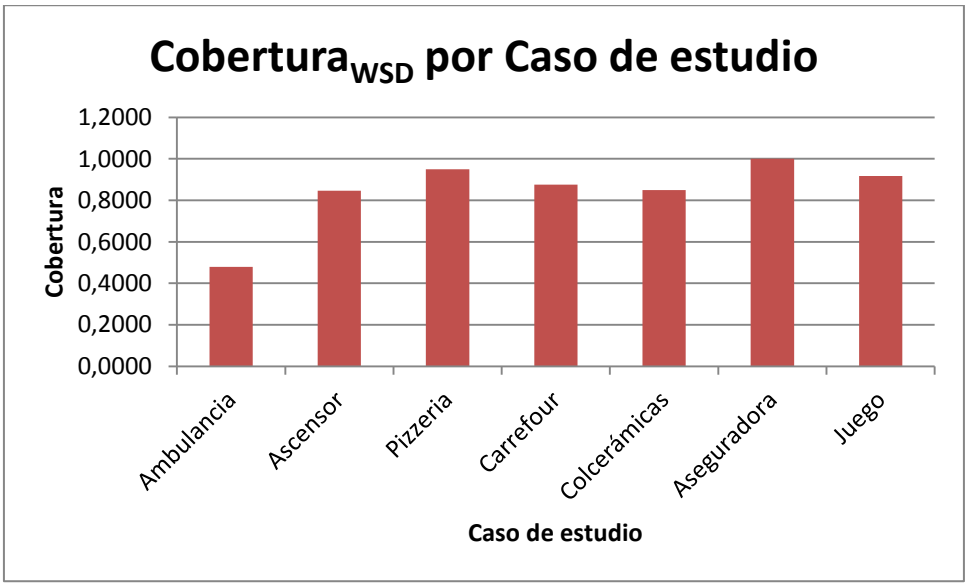


Figura 6.1 *Cobertura por caso de estudio.*
Fuente: Elaboración propia.

Particularmente se puede notar que el caso de estudio de la ambulancia es el caso de estudio con menor cobertura, esto se debió a que Wordnet para español (Multilingual Central Repository) no tenía un sentido definido para la palabra “Ambulancia” por tanto todos los elementos constitutivos que en su definición tenían la palabra ambulancia la desambiguación falló.

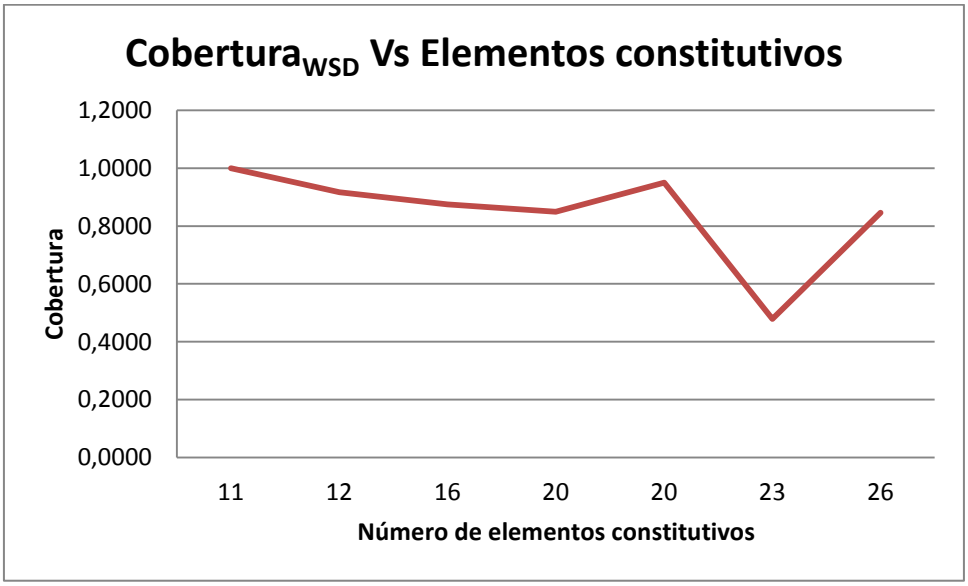


Figura 6.2 *Cobertura versus elementos constitutivos.*
Fuente: Elaboración propia.

De la gráfica anterior se puede concluir que la cobertura disminuye levemente a medida que aumentan los elementos constitutivos introducidos.

La cobertura promedio se halló en 0.8454 esto significa que aproximadamente en el 85% de los casos el sistema de desambiguación de los sentidos de las palabras dio una respuesta.

6.2.2. Análisis de Recuerdo

En las siguientes gráficas se presenta la gráfica que describe el recuerdo obtenido por el sistema sobre los casos de estudio presentados en la sección anterior y se presenta el recuerdo versus el número de elementos constitutivos introducidos.

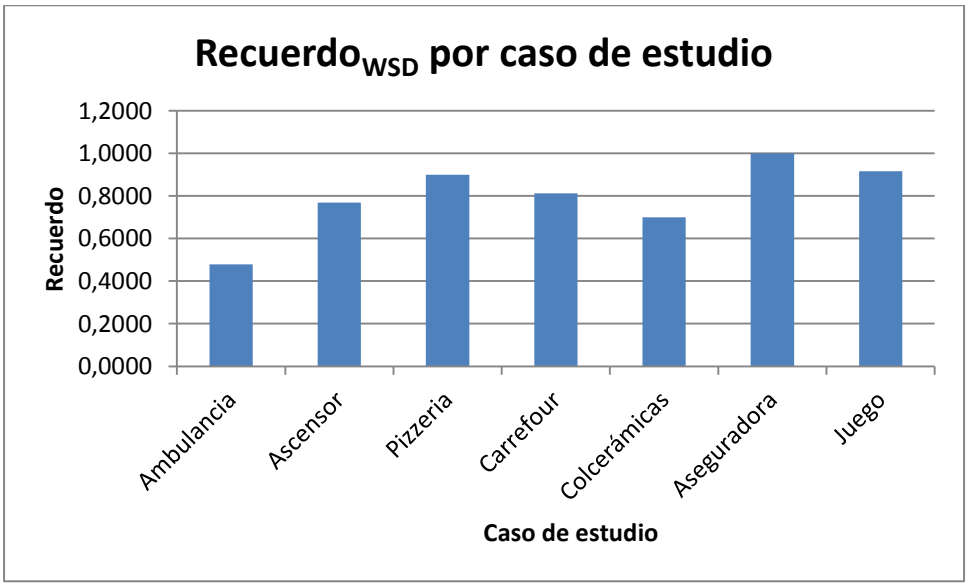


Figura 6.3 *Recuerdo por caso de estudio.*
Fuente: Elaboración propia.

Al verse afectada la cobertura en el caso de la ambulancia el recuerdo se ve afectado en el sentido de que se encuentran pocas respuestas correctas frente al total de elementos constitutivos introducidos, esto se debe a que un elemento constitutivo que no esté completamente desambiguado, no se somete a evaluación de correctitud de la desambiguación, dado esto, el cálculo del recuerdo se afecta ostensiblemente.

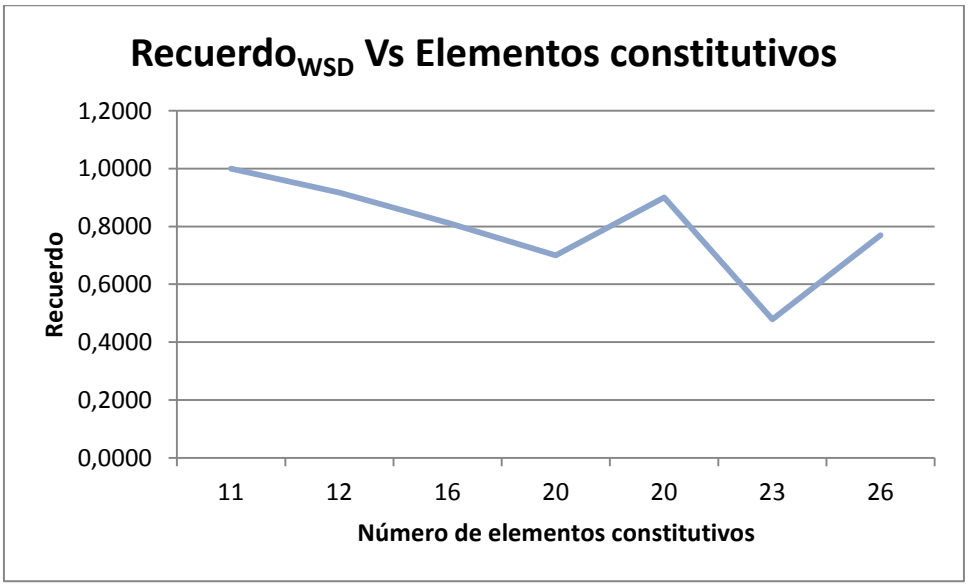


Figura 6.4 *Recuerdo versus elementos constitutivos.*
Fuente: Elaboración propia.

De la figura anterior se puede concluir que el número de elementos constitutivos introducidos afectan notablemente el recuerdo de la desambiguación es decir a mayor número de elementos constitutivos introducidos se genera un menor recuerdo.

El recuerdo promedio se halló en 0.7967 esto significa que aproximadamente en el 80% de los casos el sistema de desambiguación de los sentidos de las palabras dio una respuesta correcta sobre el total de elementos constitutivos introducidos.

6.2.3. Análisis de precisión

En las siguientes gráficas se presenta la gráfica que describe la precisión obtenida por el sistema sobre los casos de estudio presentados en la sección anterior y se presenta la precisión versus el número de elementos constitutivos introducidos.

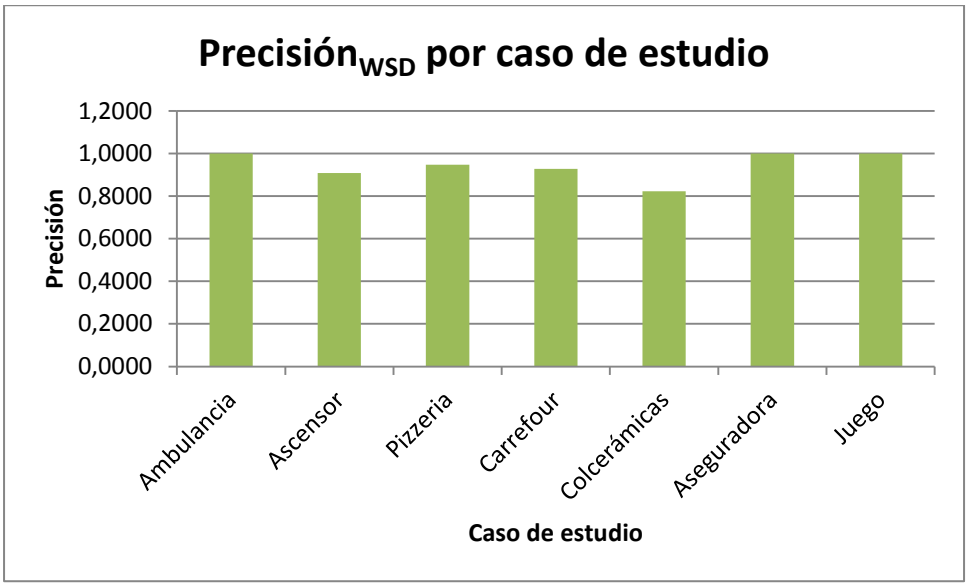


Figura 6.5 *Precisión por caso de estudio.*
Fuente: Elaboración propia.

La precisión en general del sistema de desambiguación fue estable a lo largo de los casos de estudio, no hubo observaciones anormales en la muestra de datos de los casos de estudio. El caso de estudió con más baja precisión fue el caso de estudio de Colcerámicas esto se debió a que encontró problemas al desambiguar adjetivos como “Primera” y sustantivos como “gerente”.

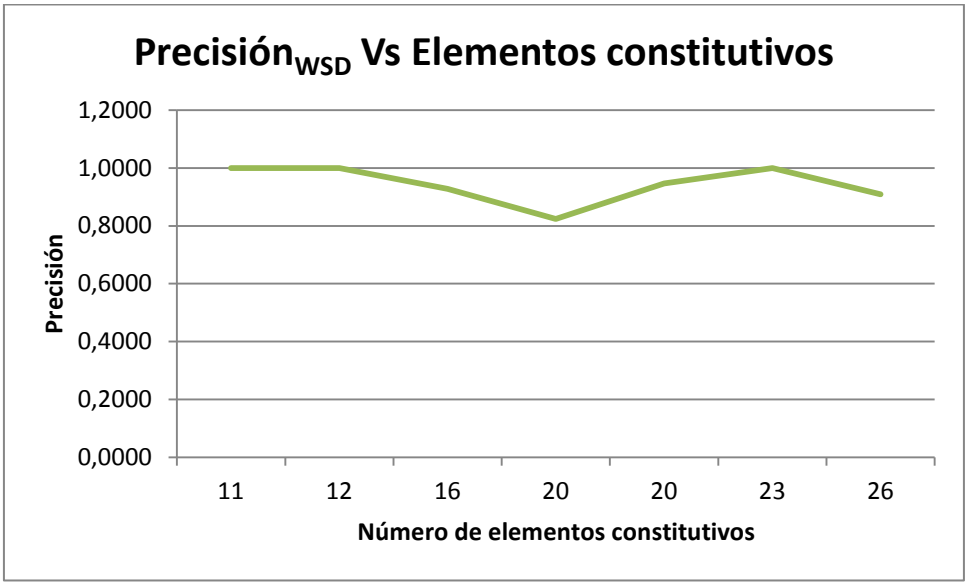


Figura 6.6 *Precisión vs Elementos constitutivos.*
Fuente: Elaboración propia.

De la figura anterior se puede concluir que sin importar el número de elementos constitutivos introducidos la precisión es estable.

La precisión promedio se halló en 0,9441 esto significa que aproximadamente en el 95% de los casos que el sistema de desambiguación de los sentidos de las palabras dio una respuesta, esta respuesta fue correctamente desambiguada.

6.2.4. Análisis de la métrica F1.

En las siguientes gráficas se presenta la gráfica que describe la métrica F1 obtenida por el sistema sobre los casos de estudio presentados en la sección anterior y se presenta la métrica F1 versus el número de elementos constitutivos introducidos. La métrica F1 es la medida que agrupa la precisión y el recuerdo con el fin de determinar en términos generales el desempeño global de la desambiguación del sistema.

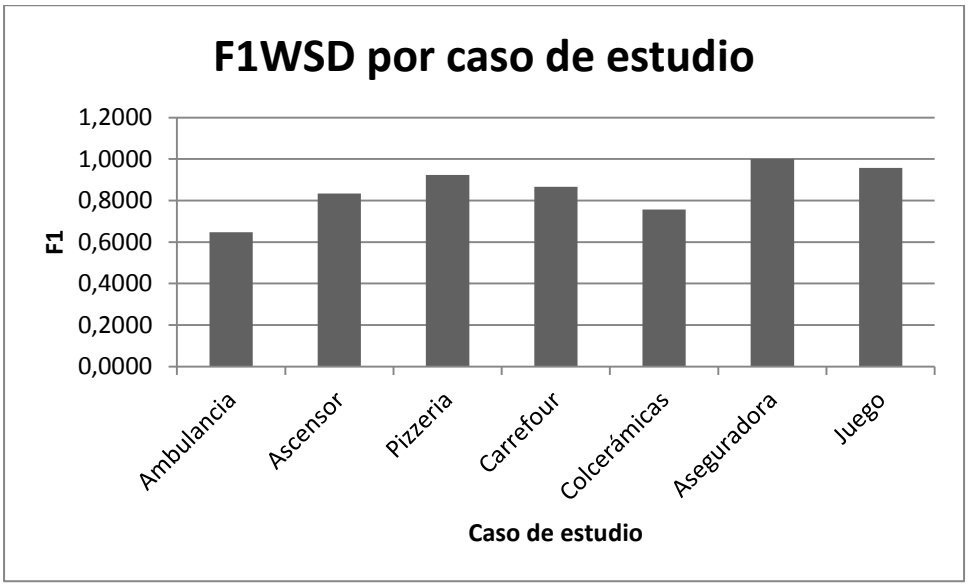


Figura 6.7 *F1 por caso de estudio.*
Fuente: Elaboración propia.

En términos generales se observa que el sistema de desambiguación obtuvo la peor desambiguación para el caso de estudio de la ambulancia es decir con un 63% de efectividad, sin embargo también se tiene que realizo una desambiguación perfecta para el caso de estudió de la aseguradora es decir con un 100% de efectividad. Para los demás casos de estudio se obtuvo una efectividad de la desambiguación entre 75% y 90%.

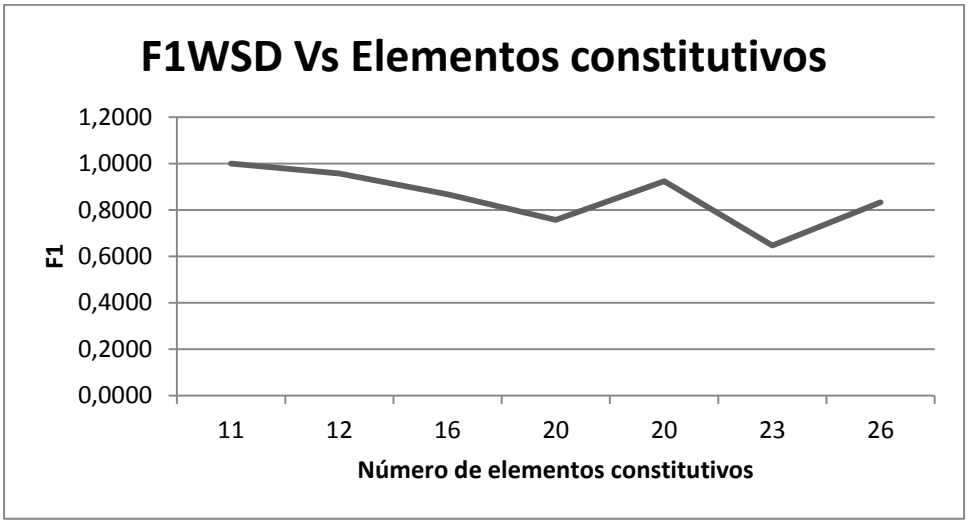


Figura 6.8 *F1 versus Elementos constitutivos.*
Fuente: Elaboración propia.

La métrica F1 promedio se halló en 0,8548 esto significa que el sistema de desambiguación realiza la tarea global de desambiguación de los sentidos de las palabras en elementos constitutivos con una efectividad aproximada del 85%.

6.3. Comparación de la desambiguación de los sentidos de las palabras.

Finalmente se compara la desambiguación del módulo de filtrado de coeficientes versus las técnicas individuales que componen el módulo de filtrado de coeficientes. (Simplified Lesk, Adapted Lesk, MFS, UKB). Esto con el fin de probar en qué medida el sistema de desambiguación propuesto realiza una mejor tarea de desambiguación de los sentidos de las palabras en requisitos de software que los demás métodos. En el anexo A en la sección A.8. Se presenta la tabla detallada con los sentidos de las palabras de cada caso de estudio en esta sección se presenta la tabla que agrupa los datos y la gráfica asociada para realizar el respectivo análisis de los datos.

Tabla 6-16 Resumen comparación del modelo de desambiguación de filtrado de coeficientes vs técnicas de desambiguación que son usadas por el modelo.

Caso Ambulancia					
	Filtrado de Coeficientes	**UKB**	**MFS**	**Alesk**	**Slesk**
Totales	23	23	23	23	23
Desambiguados	11	6	2	1	0
Desambiguados Correctos	11	6	2	1	0
Desambiguados Incorrectos	0	0	0	0	0
No desambiguados	12	17	21	22	23
Cobertura	0,4783	0,2609	0,0870	0,0435	0,0000
Precisión	1,0000	1,0000	1,0000	1,0000	0,0000
Recuerdo	0,4783	0,2609	0,0870	0,0435	0,0000
F1	0,6471	0,4138	0,1600	0,0833	0,0000
Caso Ascensor					
	Filtrado de Coeficientes	**UKB**	**MFS**	**Alesk**	**Slesk**
Totales	26	26	26	26	26
Desambiguados	22	11	7	1	0

Desambiguados Correctos	20	11	7	1	0
Desambiguados Incorrectos	2	0	0	0	0
No desambiguados	2	15	19	25	26
Cobertura	0,8462	0,4231	0,2692	0,0385	0,0000
Precisión	0,9091	1,0000	1,0000	1,0000	0,0000
Recuerdo	0,7692	0,4231	0,2692	0,0385	0,0000
F1	0,8333	0,5946	0,4242	0,0741	0,0000
Caso Carrefour					
	Filtrado de Coeficientes	**UKB**	**MFS**	**Alesk**	**Slesk**
Totales	16	16	16	16	16
Desambiguados	14	10	6	2	1
Desambiguados Correctos	13	9	6	1	1
Desambiguados Incorrectos	1	1	0	1	0
No desambiguados	2	6	10	14	15
Cobertura	0,8750	0,6250	0,3750	0,1250	0,0625
Precisión	0,9286	0,9000	1,0000	0,5000	1,0000
Recuerdo	0,8125	0,5625	0,3750	0,0625	0,0625
F1	0,8667	0,6923	0,5455	0,1111	0,0000

De esta tabla se puede observar que la técnica de desambiguación propuesta en esta tesis realiza posee una mejor cobertura y recuerdo que las demás técnicas de desambiguación en todos los casos de estudio. También se observa que para el caso de estudio del ascensor la técnica de desambiguación UKB presenta levemente una mejor precisión pero un bajo recuerdo, por tanto la métrica F1 del método de desambiguación UKB sigue siendo muy inferior que el filtrado de coeficientes.

La siguiente grafica muestra los datos de la tabla anterior utilizando una gráfica de barras donde se comparan los las métricas F1 obtenidas por los distintos métodos de desambiguación en los distintos casos de estudio, con el fin de compararlos gráficamente y demostrar porque la técnica de filtrado de coeficientes propuesta en ésta tesis es superior a los demás métodos de desambiguación.

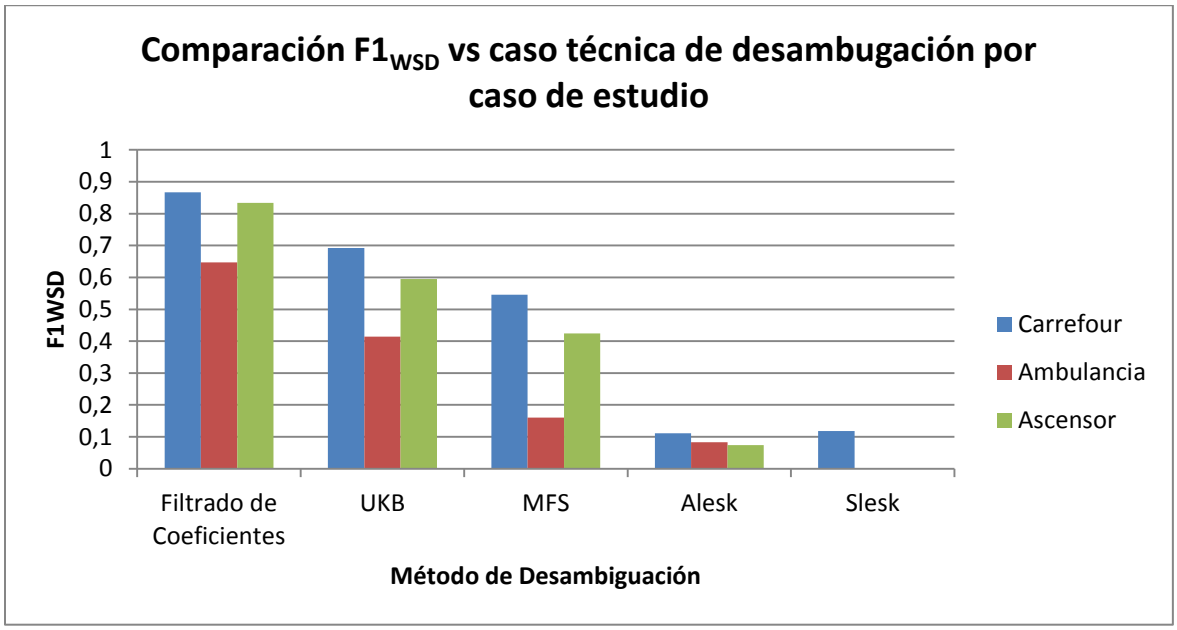


Figura 6.9 *Comparación técnicas de desambiguación por medio de la métrica F1. Fuente: Elaboración propia.*

De la gráfica anterior podemos observar que independientemente del caso de estudio la técnica de desambiguación propuesta en esta tesis (Filtrado de coeficientes), realiza una mejor desambiguación que los métodos individuales basados en conocimiento expuestos en el capítulo 3 esta comparación se hizo basados en la métrica F1.

	Filtrado de Coeficientes	UKB	MFS	Alesk	Slesk
Carrefour	0,866666667	0,69230769	0,54545455	0,11111111	0,11764706
Ambulancia	0,647058824	0,4137931	0,16	0,08333333	0
Ascensor	0,833333333	0,59459459	0,42424242	0,07407407	0
Promedio	0,782352941	0,56689846	0,37656566	0,08950617	0,03921569

De la tabla anterior se evidencia también en que el promedio de la métrica F1 de la técnica de filtrado de coeficientes es 0.78 seguido por UKB con 0.57 posteriormente MFS 0.38 luego AdaptedLesk 0.09 y finalmente Lesk con 0.04, concluyendo el filtrado de coeficientes como la mejor técnica de desambiguación de las técnicas evaluadas.

6.4. Rendimiento del módulo de desambiguación.

El rendimiento permite establecer la velocidad del sistema para realizar la tarea de desambiguación encomendada, es decir determina el tiempo que le toma al sistema realizar la tarea de desambiguación dado un número específico de elementos constitutivos. En la siguiente tabla se presentan los datos tomados para evaluar el rendimiento.

Tabla 6-17 Tiempo tomado por el modelo de desambiguación versus los elementos constitutivos a desambiguar.

Rendimiento por elementos constitutivos	
Tiempo de ejecución (segundos)	**Elementos constitutivos a procesar**
156	11
212	12
227	16
253	20
248	20
266	23
300	26

En la siguiente gráfica se presentan los datos expuestos en la tabla anterior.

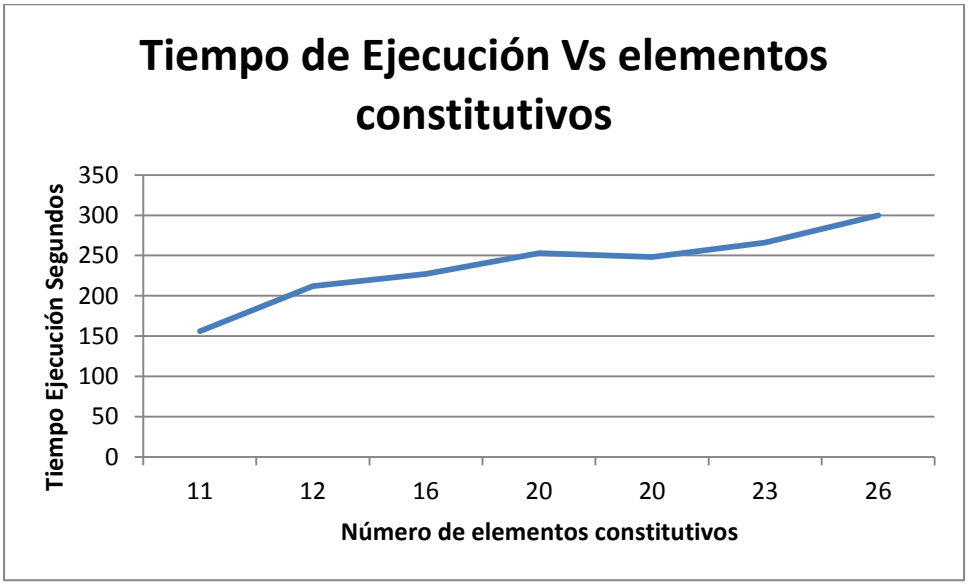


Figura 6.10 *Tiempo de ejecución para realizar las tareas de desambiguación variando el número de elementos constitutivos a procesar.*
Fuente: Elaboración propia.

Finalmente de la gráfica anterior se demuestra que a mayor número de elementos constitutivos a desambiguar mayor es el tiempo de ejecución del método de desambiguación. Sin embargo que evidencia que los tiempos de ejecución son bastante altos ya que para procesar 11 elementos constitutivos el sistema se gasta 2 minutos y 36 segundos, lo cual representa un tiempo de cómputo bastante alto.

6.5. Evaluación del módulo de detección de errores.

Para la evaluación del módulo de detección de errores se usaron igualmente los casos de estudio presentados en la Anexo A. Concretamente para la evaluación del módulo de detección de errores se evaluaron la precisión y el recuerdo en inconsistencias, duplicidades y no unificidades, estas métricas fueron expuestas en la Tabla 1-5.En la siguiente tabla se exponen los datos obtenidos al aplicar el módulo de detección de inconsistencias.

Tabla 6-18 Resumen datos obtenidos sobre los casos de estudio en la detección inconsistencias, duplicidades y no unificidades.

		Encontradas	**Correctas**	**Incorrectas**	**No Encontradas**
Ambulancia	Inconsistencias	0	0	0	0
	Duplicidades	6	4	2	0
	No Unificidades	2	2	0	0
		Encontradas	**Correctas**	**Incorrectas**	**No Encontradas**
Ascensor	Inconsistencias	2	2	0	2
	Duplicidades	11	9	2	2
	No Unificidades	3	3	0	1
		Encontradas	**Correctas**	**Incorrectas**	**No Encontradas**
Pizzería	Inconsistencias	0	0	0	0
	Duplicidades	9	7	2	0
	No Unificidades	0	0	0	0
		Encontradas	**Correctas**	**Incorrectas**	**No Encontradas**
Carrefour	Inconsistencias	6	5	1	2
	Duplicidades	3	3	0	0

	No Unificidades	0	0	0	0
Colcerámicas		**Encontradas**	**Correctas**	**Incorrectas**	**No Encontradas**
	Inconsistencias	2	2	0	2
	Duplicidades	8	8	0	0
	No Unificidades	0	0	0	0
Aseguradora		**Encontradas**	**Correctas**	**Incorrectas**	**No Encontradas**
	Inconsistencias	2	2	0	0
	Duplicidades	5	4	0	0
	No Unificidades	3	2	0	0
Juego		**Encontradas**	**Correctas**	**Incorrectas**	**No Encontradas**
	Inconsistencias	2	2	0	4
	Duplicidades	3	2	0	1
	No Unificidades	3	2	0	0

En la siguiente tabla se presentan los valores de precisión y recuerdo para cada caso de estudio tomando como referencia los valores presentados en la tabla anterior.

Tabla 6-19 Valores obtenidos al aplicar las métricas sobre la tabla anterior

Caso de Estudio	Precisión detección Inconsistencia	Recuerdo detección de Inconsistencia	Precisión detección Duplicidades	Recuerdo detección Duplicidades	Precisión detección no unificidades	Recuerdo detección no unificidades
	PI_A	**RI_A**	**PD_A**	**RD_A**	**PU_A**	**RU_A**
Ambulancia	1,0000	1,0000	0,6667	0,6667	1,0000	1,0000
Ascensor	1,0000	0,6923	0,8182	0,6923	1,0000	0,7500
Pizzería	1,0000	1,0000	0,7778	0,7778	1,0000	1,0000
Carrefour	0,8333	0,6250	1,0000	1,0000	1,0000	1,0000
Colcerámicas	1,0000	0,5000	1,0000	1,0000	1,0000	1,0000
Aseguradora	1,0000	1,0000	0,8000	0,8000	0,6667	0,6667
Juego	1,0000	0,3333	0,6667	0,5000	0,6667	0,6667
Promedio	**0,976190476**	**0,735805861**	**0,818470418**	**0,7766788**	**0,9047619**	**0,8690476**

De la tabla anterior se puede concluir que el sistema detecta inconsistencias con un 98% de precisión, es decir de los elementos constitutivos que muestra como inconsistentes el 98% son verdaderamente inconsistencias, el recuerdo de la detección de inconsistencias

fue de 74% esto indica que de las inconsistencias totales que posee una especificación detectar el 74% de ellas de manera correcta.

En cuanto a la detección de duplicidades se tiene que de los elementos constitutivos que detecta como duplicados el 82% se encuentran realmente duplicados. Mientras que sobre el total de todas las duplicidades que el sistema debería detectar el 78% son correctamente detectadas por el sistema.

Finalmente en cuanto a la detección de no unificidades se tiene que de los elementos constitutivos que detecta como no unificados el 90% están realmente no unificados, mientras que del total de elementos constitutivos incluyendo detectados y no detectados el sistema detecto como duplicados el 86%.

6.6. Conclusiones

Se evaluaron las métricas propuestas en el marco teórico en la sección 1.5 (Cobertura, precisión y recuerdo) para la desambiguación de los sentidos de las palabras, y para la detección de errores se evaluó Precisión y recuerdo. Para tal fin se realizaron los experimentos sobre tres casos de estudio sacados de la literatura y cuatro adicionales tomados de casos reales de la industria.

De los experimentos se concluye como principales fortalezas del modelo propuesto que:

i. El módulo de desambiguación realiza las tareas de desambiguación encomendadas con una efectividad del 86% (esto dado por la métrica F1), comparativamente sobre los casos de estudio presentados, este módulo de desambiguación es superior a las técnicas presentadas en el capítulo 3 (UKB, MFS, Adapted Lesk, Simplified Lesk).

ii. El sistema identifica los errores de manera satisfactoria ya que la precisión de detección de inconsistencias es del 98%, la precisión de la detección de duplicidades es del 82% y la precisión de la identificación de no unificidades es del 90%. Esto indica que en promedio el módulo de detección de errores detecta

con una precisión del 90% los errores presentes en las especificaciones de requisitos de software.

Como principales debilidades se encontró que:

i. Los tiempos de procesamiento del módulo de desambiguación son relativamente altos.
ii. La falta de información en las bases de conocimiento afectan ostensiblemente la calidad de la desambiguación, (como ocurrió para el caso de estudio de la ambulancia).
iii. No se realiza un proceso semántico para la detección de inconsistencias, ya que solo se analiza a nivel de antonimia directa, lo cual puede ser mejorado.

Dicho lo anterior con este capítulo se le pretende dar cumplimiento al quinto objetivo del presente trabajo: “Validar el modelo de resolución de ambigüedad de sentidos de palabras y el modelo de integración de propósito general propuestos mediante la implementación de un ambiente de prueba que permita evaluar el desempeño de una arquitectura existente de educción de requisitos de software con y sin el uso de los modelos propuestos con el fin de validar sus fortalezas y debilidades”.

7. Conclusiones y trabajo futuro.

En este capítulo se presenta el resumen de los resultados, las contribuciones y el trabajo futuro.

7.1. Resumen de Resultados y Contribuciones

En esta tesis de maestría, se propuso un modelo de resolución de ambigüedad de sentidos de palabras para mejorar la calidad de resultados en una arquitectura de educción de requisitos de software. Esta tesis ha sido direccionado para responder algunos de los principales retos de la ingeniería de software y el procesamiento del lenguaje natural, tales como: 1) La desambiguación semántica (polisémica) de requisitos de software, 2) La detección de errores de inconsistencia, duplicidad y no unificidad en elementos constitutivos de manera automática y 3) un modelo de propósito general que permita unir el modelo propuesto con cualquier sistema de educción de requisitos que aporte arquitecturas de educción de requisitos..

Para soportar el modelo completo (Con los problemas citados en los numerales 1), 2) y 3) se definieron algunos trabajos específicos que originaron los siguientes aportes.

En primer lugar lo que constituye el sistema NEONPL es una aplicación orientada a la desambiguación de elementos constitutivos a partir del requisito fuente que originó dicho elemento constitutivo, definiendo para ello un comportamiento funcional consistente de tres módulos i) modulo adaptador para unir el sistema con componentes externos ii) Modulo de desambiguación de los sentidos de las palabras que integró diversas técnicas de desambiguación basada en conocimiento, y iii) Módulo de detección de errores de inconsistencia, duplicidad y no unificidad en elementos constitutivos.

Para lograr la funcionalidad del sistema NEONPL en primer lugar se presentó la base teórica que permite establecer los tipos de ambigüedad presente en el lenguaje natural escrito, y cuáles de estas afectan de manera más directa los requisitos de software. Se

encontraron cuatro tipos de ambigüedades i) Morfológica, ii) Sintáctica, iii) Semántica polisémica y iv) semántica referencial, y se demostró por qué la ambigüedad semántica polisémica afecta con más fuerza los requisitos de software escritos, originando errores de inconsistencia, duplicidad y no unificidad.

En segundo lugar se presentaron las técnicas actuales de desambiguación polisémica, Se distinguieron i) Métodos supervisados, ii) Métodos no supervisados y iii) Métodos basados en conocimiento. Se enfocó en las técnicas basadas en conocimiento debido a su facilidad de uso y a su rendimiento en las tareas de desambiguación. Adicionalmente se realizó una comparativa de estas técnicas versus otras técnicas presentes en el estado del arte.

En tercer lugar se presentó el modelo teórico que presenta el proceso de desambiguación de elementos constitutivos, este modelo agrupa las técnicas basadas en conocimiento mediante un filtrado de coeficientes de mínimos cuadrados y dan lugar a una nueva técnica de desambiguación que aporta resultados superiores a las demás técnicas. También se presentaron las reglas que permiten realizar la detección de errores de inconsistencias, duplicidades y no unificidades sobre elementos constitutivos ya desambiguados.

En cuarto lugar se presentó la arquitectura del sistema que implementa el modelo teórico descrito anteriormente y le agrega el módulo adaptador para la extensión del modelo. Este sistema se desarrolló en el lenguaje JAVA en la parte del backend y en lenguaje PHP en la parte del frontend, este sistema usa bases de datos relacionales mediante el motor de bases de datos MySQL.

En quinto lugar se presentaron las pruebas hechas al sistema NEONPL con el fin de establecer su capacidad de desambiguación, su rendimiento y la capacidad de detección de errores. Las métricas evaluadas son precisión, cobertura y recuerdo. De las pruebas hechas se concluyó que el sistema NEONPL es un sistema muy superior a los demás enfoques en cuanto a la desambiguación y constituye el primer sistema de desambiguación semántica polisémica adaptado al proceso de requisitos de software, ya que tiene bastantes características y porcentajes suficientemente buenos que permiten además de ser un sistema enfocado en la desambiguación de requisitos de software es capaz de detectar con efectividad los errores de inconsistencia, duplicidad y no unificidad

ocasionados por la ambigüedad semántica polisémica en especificaciones de requisitos de software.

7.2. Direcciones para trabajo futuro

La desambiguación semántica polisémica es un problema trabajado a nivel mundial por diferentes organizaciones constituye un campo de investigación relativamente amplio y extenso. Además si este problema es direccionado al proceso de educción de requisitos de software que también es otro paradigma relativamente nuevo y problemático surgen varias preguntas de investigación interesantes que representan el trabajo futuro para esta tesis de maestría. Cabe señalar sin embargo que el marco presentado en esta tesis puede ser visto como un punto de partida para tales trabajos futuros.

En particular se pueden señalar diferentes aspectos en los que el marco de trabajo y sobre todo el prototipo implementado se puede mejorar. En los siguientes apartados se enumeran las principales líneas a seguir a partir del trabajo desarrollado en esta tesis de maestría:

1) Las primeras se relacionan directamente con el módulo de estructuración de los elementos constitutivos implementado en esta tesis:
    a. Mejorar este módulo para realizar la desambiguación morfológica y resolución de ambigüedad semántica referencial, dado que el modelo actual supone que las entradas está libre de esta ambigüedad.
    b. Mejorar el proceso de resolución de ambigüedad estructural, debido a que esta tesis se sirve del proceso implementado por Freeling para la resolución de este tipo de ambigüedad, sería posible añadirle un módulo de desambiguación estructural que sea un poco más adaptado al proceso de educción de requisitos de software.

2) Las segundas líneas de investigación a desarrollar se relacionan con las tareas de desambiguación semántica polisémica,

a. Crear o adaptar nuevos mecanismos de desambiguación más efectivos, (Que mejoren los porcentajes de desambiguación obtenidos en esta tesis, o consuman menos tiempo).
b. Comparar la técnica propuesta en esta tesis con otros enfoques de desambiguación semántica en requisitos de software.

3) Las terceras líneas de investigación a desarrollar se relacionan con proceso de detección de errores de inconsistencia, duplicidad y no unificidad.
    a. Utilizar un proceso semántico para la detección de inconsistencias, ya que el modelo actual solo soporta como inconsistentes aquellos elementos constitutivos que contiene antónimos explícitos en su definición.
    b. Utilizar ontologías para realizar un mejor razonamiento sobre las inconsistencias detectadas, así se evitaría en gran medida la intervención del usuario para solucionar dichas inconsistencias.
    c. Incrementar las reglas propuestas en la presente tesis con el fin de detectar otros tipos de características presentes en los requisitos de software, por ejemplo necesario, conciso. Verificable, completo entre otros.
    d. Crear métricas de validación semántica que permitan determinar la correctitud semántica de los elementos constitutivos desambiguados.
    e. Educir otros elementos constitutivos a partir de los elementos constitutivos desambiguados.

A. Anexo Casos de estudio detallados.

A continuación se presentan los casos de estudio que se utilizaron para realizar la validación del modelo propuesto, estos casos de estudio abarcan diferentes temas propios de los estudios de educción y validación de requisitos, por tanto se encuentra que estos casos de estudio han sido particularmente adaptados en diversas áreas y son aceptados mundialmente como casos de estudio válidos para los problemas de educción de requisitos [79] .

A.1. Caso de estudio “Ambulancia de Londres”

El sistema de la ambulancia de Londres fue tratado y mencionado por [80] con el fin de explicar el proceso de educción de requisitos de software orientado a objetivos, por lo cual se convierte en un caso de estudio ejemplar a tratar. A Continuación se presenta la entrada al sistema NEONPL.

Tabla A-1 XML Entrada Ambulancia

```
<?xml version="1.0" encoding="UTF-8"?>
<KAOS>
    <Objetivos>
        <objetivo id="1">
            <definicion> Buscar que los auxiliares de la ambulancia
lleguen al lugar del incidente en el menor tiempo posible</definicion>
            <fuente>Se busca que los auxiliares de la ambulancia lleguen
al lugar del incidente en el menor tiempo posible.</fuente>
        </objetivo>
        <objetivo id="2">
            <definicion> Atender las llamadas de emergencia</definicion>
            <fuente>El asistente de control debe atender las llamadas de
emergencia.</fuente>
        </objetivo>
        <objetivo id="3">
            <definicion> Rellenar el formulario del
incidente</definicion>
            <fuente>El asistente de control tiene que rellenar el
formulario del incidente.</fuente>
        </objetivo>
        <objetivo id="4">
```

```
            <definicion> Indicar en el formulario la localización del
incidente y la hora en la que la llamada fue hecha</definicion>
            <fuente>Es necesario que el asistente de control indique en
el formulario la localización del incidente y la hora en la que la
llamada fue hecha.</fuente>
        </objetivo>
        <objetivo id="5">
            <definicion> Examinar el formulario del
incidente</definicion>
            <fuente>el operario de localización de recursos debe
examinar el formulario del incidente</fuente>
        </objetivo>
        <objetivo id="6">
            <definicion> Identificar a la ambulancia más
cercana</definicion>
            <fuente>El operario de localización de recursos tiene que
identificar a la ambulancia más cercana.</fuente>
        </objetivo>
        <objetivo id="7">
            <definicion> Seleccionar la ambulancia más cercana al
incidente</definicion>
            <fuente>el operario de localización de recursos debe
seleccionar la ambulancia más cercana al incidente</fuente>
        </objetivo>
        <objetivo id="8">
            <definicion> Proponer al operario de localización de
recursos la ambulancia más apropiada</definicion>
            <fuente>el sistema CAD debe proponer al operario de
localización de recursos la ambulancia más apropiada</fuente>
        </objetivo>
        <objetivo id="9">
            <definicion>Aceptar o rechazar propuesta del sistema
CAD</definicion>
            <fuente>El operario de localización de recursos puede
aceptar o rechazar propuesta del sistema CAD.</fuente>
        </objetivo>
        <objetivo id="10">
            <definicion>Proponer otra ambulancia , en caso de que el
operario de localización de recursos rechace la propuesta inicial
</definicion>
            <fuente>El sistema CAD puede proponer otra ambulancia, en
caso de que el operario de localización de recursos rechace la propuesta
inicial</fuente>
        </objetivo>
        <objetivo id="11">
            <definicion>Llamar a los auxiliares de la
ambulancia</definicion>
            <fuente>Es necesario que el operador logístico llame a los
auxiliares de la ambulancia</fuente>
        </objetivo>
        <objetivo id="12">
            <definicion>Dar los datos al auxiliar de la ambulancia para
llegar al lugar del incidente </definicion>
            <fuente>El operador logístico debe darle los datos al
auxiliar de la ambulancia para llegar al lugar del incidente</fuente>
```

```
        </objetivo>
        <objetivo id="13">
            <definicion>Guiar al personal de ambulancia hacia el lugar
del incidente</definicion>
            <fuente>Es menester que el operador de radio guíe al
personal de ambulancia hacia el lugar del incidente</fuente>
        </objetivo>
        <objetivo id="14">
            <definicion>Garantizar que el operador de radio reciba la
información de la ambulancia seleccionada</definicion>
            <fuente>Se debe garantizar que el operador de radio reciba
la información de la ambulancia seleccionada</fuente>
        </objetivo>
        <objetivo id="15">
            <definicion>Dar instrucciones de localización al auxiliar de
la ambulancia constantemente</definicion>
            <fuente>El operador de radio debe dar instrucciones de
localización al auxiliar de la ambulancia constantemente</fuente>
        </objetivo>
        <objetivo id="16">
            <definicion>Reportar el estado de la operación al operador
de radio</definicion>
            <fuente>El auxiliar de la ambulancia tiene que reportar el
estado de la operación al operador de radio.</fuente>
        </objetivo>
        <objetivo id="17">
            <definicion>Avisar el estado de la operación al operador de
radio</definicion>
            <fuente>El auxiliar de la ambulancia tiene que avisar el
estado de la operación al operador de radio.</fuente>
        </objetivo>
    </Objetivos>
    <Agentes>
        <agente id="18">
            <definicion>El auxiliar de la ambulancia</definicion>
            <fuente>El auxiliar de la ambulancia tiene que avisar el
estado de la operación al operador de radio.</fuente>
        </agente>
        <agente id="19">
            <definicion> El asistente de control</definicion>
            <fuente>El asistente de control debe atender las llamadas de
emergencia.</fuente>
        </agente>
        <agente id="20">
            <definicion> El asistente de control</definicion>
            <fuente>El asistente de control tiene que rellenar el
formulario del incidente.</fuente>
        </agente>
        <agente id="21">
            <definicion>El auxiliar de la ambulancia</definicion>
            <fuente>El auxiliar de la ambulancia tiene que reportar el
estado de la operación al operador de radio.</fuente>
        </agente>
    </Agentes>
    <Entidades>
        <entidad id="22">
```

```
            <definicion>radio</definicion>
            <fuente>El auxiliar de la ambulancia tiene que avisar el
estado de la operación al operador de radio.</fuente>
        </entidad>
        <entidad id="23">
            <definicion>ambulancia</definicion>
            <fuente>El operador de radio debe dar instrucciones de
localización al auxiliar de la ambulancia constantemente</fuente>
        </entidad>
    </Entidades>
</KAOS>
```

Tabla A-2 Resultados obtenidos caso de estudio ambulancia

# Elemento constitutivo	Tipo	Desambiguado	Correctamente Desambiguado	Incorrectamente Desambiguado	No desambiguado
1	Objetivo	SI	SI	NO	NO
2	Objetivo	SI	SI	NO	NO
3	Objetivo	SI	SI	NO	NO
4	Objetivo	SI	SI	NO	NO
5	Objetivo	SI	SI	NO	NO
6	Objetivo	NO	NA	NA	SI
7	Objetivo	NO	NA	NA	SI
8	Objetivo	NO	NA	NA	SI
9	Objetivo	SI	SI	NO	NO
10	Objetivo	NO	NA	NA	SI
11	Objetivo	NO	NA	NA	SI
12	Objetivo	NO	NA	NA	SI
13	Objetivo	NO	NA	NA	SI
14	Objetivo	NO	NA	NA	SI
15	Objetivo	NO	NA	NA	SI
16	Objetivo	SI	SI	NO	NO
17	Objetivo	SI	SI	NO	NO
18	Agente	NO	NA	NA	SI
19	Agente	SI	SI	NO	NO
20	Agente	SI	SI	NA	NO
21	Agente	NO	NA	NA	SI
22	Entidad	SI	SI	NO	NO
23	Entidad	NO	NA	NA	SI

	Encontradas	Correctas	Incorrectas	No

				Encontradas
Inconsistencias Detectadas	0	0	0	0
Duplicidades Detectadas	6	4	2	0
No Unificidades Detectadas	2	2	0	0

CoberturaWSD	0,47826087	Totales	23
PrecisiónWSD	1,00	Desambiguados	11
RecuerdoWSD	0,48	Desambiguados Correctos	11
F1WSD	0,647058824	Desambiguados Incorrectos	0
		No desambiguados	12

Presición detección Inconsistencia	1,00
Presición deteccion Duplicidad	0,67
Presición deteccion No Unificidad	1,00
Recuerdo deteccion Inconsistencia	1,00
Recuerdo deteccion Duplicidad	0,67
Recuerdo detección No Unificidad	1,00

A.2. Caso de estudio sistema de Ascensor

El sistema de ascensores fue tratado y mencionado por [16] con el fin de explicar el proceso de educción de requisitos de software orientado a objetivos, por lo cual se convierte en otro caso de estudio ejemplar a tratar. A Continuación se presenta la entrada al sistema NEONPL.

Tabla A-3 XML entrada caso estudio ascensor

```
<?xml version="1.0" encoding="UTF-8" ?>
<KAOS>
    <Objetivos>
        <objetivo id="1"><definicion>Garantizar el funcionamiento
correcto de el ascensor</definicion>
            <fuente>La compañía encargada del ascensor debe garantizar
el funcionamiento correcto del ascensor</fuente>
        </objetivo>
        <objetivo id="2"><definicion>Asegurar controles de
emergencia</definicion>
        <fuente>la compañía tiene que asegurar controles de
emergencia.</fuente>
        </objetivo>
        <objetivo id="3"><definicion>Realizar el montaje del
ascensor</definicion>
            <fuente>La compañía tiene que realizar el  montaje del
ascensor</fuente>
        </objetivo>
```

```
        <objetivo id="4"><definicion>Diseñar la estructura del
ascensor</definicion>
            <fuente>Es necesario que el ingeniero civil diseñe la
estructura del ascensor.</fuente>
        </objetivo>
        <objetivo id="5"><definicion>Garantizar una forma de escape de
el ascensor</definicion>
<fuente> El ingeniero civil debe garantizar una forma de escape del
ascensor</fuente>
        </objetivo>
        <objetivo id="6"><definicion>evitar que las puertas se abran
mientras esté en movimiento</definicion>
<fuente> El sistema de control debe evitar que las puertas se abran
mientras esté en movimiento</fuente>
</objetivo>
<objetivo id="7"><definicion>permitir que las puertas se abran mientras
esté en movimiento</definicion>
<fuente> El sistema de control debe permitir que las puertas se abran
mientras esté en movimiento</fuente>
</objetivo>
        <objetivo id="8"><definicion>Detener el elevador siempre que
haya una falla de energía</definicion>
        <fuente>Es importante que el sistema de control detenga el
elevador siempre que haya una falla de energía</fuente>
        </objetivo>
        <objetivo id="9"><definicion>Desear que el sistema de control
informe a los pasajeros del estado de su petición</definicion>
<fuente>Se desea que el sistema de control informe a los pasajeros del
estado de su petición</fuente>
        </objetivo>
        <objetivo id="10"><definicion>Verificar el funcionamiento del
botón de emergencia</definicion>
<fuente>El ingeniero de control tiene que verificar el funcionamiento
del botón de emergencia</fuente>
        </objetivo>
        <objetivo id="11"><definicion>Realizar una interfaz usable por
el usuario</definicion>
<fuente>La compañía tiene que realizar una interfaz usable por el
usuario</fuente>
        </objetivo>
        <objetivo id="12"><definicion>Crear una interfaz basada en
botones negros</definicion>
            <fuente>el ingeniero de control debe crear una interfaz
basada en botones negros</fuente>
        </objetivo>
        <objetivo id="13"><definicion>Crear un panel basado en botones
blancos</definicion>
            <fuente>el ingeniero de control debe crear una interfaz
basada en botones blancos</fuente>
        </objetivo>
        <objetivo id="14"><definicion>Desarrollar el software para el
funcionamiento de los ascensores</definicion>
            <fuente>El técnico de sistemas tiene que desarrollar el
software para el funcionamiento de los ascensores</fuente>
        </objetivo>
```

```
    </Objetivos>
    <Agentes>
        <agente id="15"><definicion>La compañía</definicion>
            <fuente>La compañía encargada del ascensor debe garantizar
el funcionamiento correcto del ascensor</fuente>
        </agente>
        <agente id="16"><definicion>La comapñia</definicion>
        <fuente>la compañía tiene que asegurar controles de
emergencia.</fuente>
            </agente>
        <agente id="17"><definicion>La compañía</definicion>
            <fuente>La compañía tiene que realizar el  montaje del
ascensor</fuente>
        </agente>
        <agente id="18"><definicion>El ingeniero civil</definicion>
            <fuente>Es necesario que el ingeniero civil diseñe la
estructura del ascensor.</fuente>
        </agente>
        <agente id="19"><definicion>El ingeniero civil</definicion>
<fuente> El ingeniero civil debe garantizar una forma de escape del
ascensor</fuente>
        </agente>
        <agente id="20"><definicion>El sistema de control</definicion>
<fuente> El sistema de control debe evitar que las puertas se abran
mientras esté en movimiento</fuente>
</agente>
        <agente id="21"><definicion>el sistema de control</definicion>
        <fuente>Es importante que el sistema de control detenga el
elevador siempre que haya una falla de energía</fuente>
        </agente>
        <agente id="22"><definicion>el sistema de control</definicion>
<fuente>Se desea que el sistema de control informe a los pasajeros del
estado de su petición</fuente>
        </agente>
        <agente id="23"><definicion>El ingeniero de control</definicion>
<fuente>El ingeniero de control tiene que verificar el funcionamiento
del botón de emergencia</fuente>
        </agente>
        <agente id="24"><definicion>La compañía</definicion>
<fuente>La compañía tiene que realizar una interfaz usable por el
usuario</fuente>
        </agente>
        <agente id="25"><definicion>el ingeniero de control</definicion>
            <fuente>el ingeniero de control debe crear una interfaz
basada en botones</fuente>
        </agente>
        <agente id="26"><definicion>El técnico de sistemas</definicion>
            <fuente>El técnico de sistemas tiene que desarrollar el
software para el funcionamiento de los ascensores</fuente>
        </agente>
    </Agentes>
</KAOS>
```

Tabla A-4 Resultados obtenidos caso de estudio ascensor

# Elemento constitutivo	Tipo	Desambiguado	Correctamente Desambiguado	Incorrectamente Desambiguado	No desambiguado
1	Objetivo	SI	SI	NO	NO
2	Objetivo	SI	NO	SI	NO
3	Objetivo	SI	SI	NO	NO
4	Objetivo	SI	SI	NO	NO
5	Objetivo	SI	SI	NO	NO
6	Objetivo	SI	SI	NO	NO
7	Objetivo	SI	SI	NO	NO
8	Objetivo	SI	NO	SI	NO
9	Objetivo	SI	SI	NO	NO
10	Objetivo	SI	SI	NO	NO
11	Objetivo	NO	NA	NA	SI
12	Objetivo	NO	NA	NA	SI
13	Objetivo	NO	NA	NA	NO
14	Objetivo	SI	SI	NO	NO
15	Agente	SI	SI	NO	NO
16	Agente	SI	SI	NO	NO
17	Agente	SI	SI	NO	NO
18	Agente	SI	SI	NO	NO
19	Agente	SI	SI	NO	NO
20	Agente	SI	SI	NO	NO
21	Agente	SI	SI	NO	NO
22	Agente	SI	SI	NO	NO
23	Agente	SI	SI	NO	NO
24	Agente	SI	SI	NO	NO
26	Agente	SI	SI	NO	NO

	Encontradas	Correctas	Incorrectas	No Encontradas
Inconsistencias Detectadas	2	2	0	2
Duplicidades Detectadas	11	9	2	2
No Unificidades Detectadas	3	3	0	1

CoberturaWSD	0,846153846
PrecisiónWSD	0,91
RecuerdoWSD	0,77
F1WSD	0,833333333

Totales	26
Desambiguados	22
Desambiguados Correctos	20
Desambiguados Incorrectos	2
No desambiguados	2

Presición detección Inconsistencia	1,00
Presición deteccion Duplicidad	0,82
Presición deteccion No Unificidad	1,00
Recuerdo deteccion Inconsistencia	0,69
Recuerdo deteccion Duplicidad	0,69
Recuerdo detección No Unificidad	0,75

A.3. Caso de estudio Pizzería

El caso de estudio de la pizzería fue tratado y mencionado por [81] con el fin de explicar el proceso de educción de requisitos de software orientado a objetivos, por lo cual se convierte en otro caso de estudio ejemplar a tratar. A Continuación se presenta la entrada del sistema.

Tabla A-5 XML entrada caso de estudio pizzería

```
<?xml version="1.0" encoding="UTF-8" ?>
<KAOS>
<Objetivos>
    <objetivo id="1"><definicion>Aumentar la rentabilidad del
negocio</definicion>
    <fuente>El administrador de la pizzería desea aumentar la
rentabilidad del negocio</fuente>
     </objetivo>
    <objetivo id="2"><definicion>Ofrecer a los clientes diversos tipos
de pizza</definicion>
    <fuente>El administrador de la pizzería debe ofrecer a los clientes
diversos tipos de pizza</fuente>
     </objetivo>
    <objetivo id="3"><definicion>Ofrecer a los clientes la posibilidad
```

```
de ordenar aditivos para la pizza</definicion>
    <fuente>el administrador de la pizzería quiere ofrecer a los
clientes la posibilidad de ordenar aditivos para la pizza</fuente>
     </objetivo>
    <objetivo id="4"><definicion>Entregar el pedido al cliente en menos
de 30 minutos; de lo contrario, será gratis</definicion>
    <fuente>la pizzería debe entregar el pedido al cliente en menos de
30 minutos</fuente>
     </objetivo>
    <objetivo id="5"><definicion>Evitar entregar pedidos
retardados</definicion>
    <fuente>los repartidores deben evitar entregar pedidos
retardados</fuente>
    </objetivo>
    <objetivo id="6"><definicion>Ingresar el pedido en el
sistema</definicion>
    <fuente>El operador logístico debe ingresar el pedido en el
sistema</fuente>
     </objetivo>
    <objetivo id="7"><definicion>Enviar la orden al
cocinero</definicion>
    <fuente>El operador logístico debe enviar la orden al
cocinero</fuente>
     </objetivo>
    <objetivo id="8"><definicion>Preparar la masa, los ingredientes y el
horno</definicion>
    <fuente>El cocinero tiene que preparar la masa, los ingredientes y
el horno</fuente>
    </objetivo>
    <objetivo id="9"><definicion>Despachar la orden</definicion>
    <fuente>el cocinero debe despachar la orden</fuente>
     </objetivo>
    <objetivo id="10"><definicion>Evitar la pérdida de
dinero</definicion>
    <fuente>el administrador de la pizzería evita la pérdida de
dinero</fuente>
    </objetivo>
</Objetivos>
<Agentes>
    <agente id="11"><definicion>El administrador</definicion>
    <fuente>El administrador de la pizzería desea aumentar la
rentabilidad del negocio</fuente>
     </agente>
    <agente id="12"><definicion>el administrador</definicion>
    <fuente>El administrador de la pizzería debe ofrecer a los clientes
diversos tipos de pizza</fuente>
     </agente>
    <agente id="13"><definicion>el administrador</definicion>
    <fuente>el administrador de la pizzería quiere ofrecer a los
clientes la posibilidad de ordenar aditivos para la pizza</fuente>
     </agente>
    <agente id="14"><definicion>la pizzería</definicion>
    <fuente>la pizzería debe entregar el pedido al cliente en menos de
30 minutos</fuente>
     </agente>
```

```
    <agente id="15"><definicion>los repartidores</definicion>
    <fuente>los repartidores deben evitar entregar pedidos
retardados</fuente>
    </agente>
    <agente id="16"><definicion>el operador logistico</definicion>
    <fuente>El operador logístico debe ingresar el pedido en el
sistema</fuente>
     </agente>
    <agente id="17"><definicion>Enviar la orden al cocinero</definicion>
    <fuente>El operador logístico debe enviar la orden al
cocinero</fuente>
     </agente>
    <agente id="18"><definicion>el cocinero</definicion>
    <fuente>El cocinero tiene que preparar la masa, los ingredientes y
el horno</fuente>
    </agente>
    <agente id="19"><definicion>el cocinero</definicion>
    <fuente>el cocinero debe despachar la orden</fuente>
     </agente>
    <agente id="20"><definicion>el administrador</definicion>
    <fuente>el administrador de la pizzería evita la pérdida de
dinero</fuente>
    </agente>
</Agentes>
</KAOS>
```

Tabla A-6 Resultados obtenidos caso de estudio pizzería

# Elemento constitutivo	Tipo	Desambiguado	Correctamente Desambiguado	Incorrectamente Desambiguado	No desambiguado
1	Objetivo	SI	SI	NO	NO
2	Objetivo	SI	SI	NO	NO
3	Objetivo	SI	SI	NO	NO
4	Objetivo	SI	SI	NO	NO
5	Objetivo	SI	SI	NO	NO
6	Objetivo	SI	SI	NO	NO
7	Objetivo	SI	SI	NO	NO
8	Objetivo	SI	NO	SI	NO
9	Objetivo	SI	SI	NO	NO
10	Objetivo	SI	SI	NO	NO
11	Agente	SI	SI	NA	SI
12	Agente	SI	SI	NA	SI
13	Agente	SI	SI	NA	NO
14	Agente	NO	NA	NA	SI
15	Agente	SI	SI	NO	NO
16	Agente	SI	SI	NO	NO
17	Agente	SI	SI	NO	NO

18	Agente	SI	SI	NO	NO
19	Agente	SI	SI	NO	NO
20	Agente	SI	SI	NO	NO

			Encontradas	Correctas	Incorrectas	No Encontradas
Inconsistencias Detectadas			0	0	0	0
Duplicidades Detectadas			9	7	2	0
No Unificidades Detectadas			0	0	0	0

CoberturaWSD	0,95
PrecisiónWSD	0,95
RecuerdoWSD	0,90
F1WSD	0,92307692

Totales	20
Desambiguados	19
Desambiguados Correctos	18
Desambiguados Incorrectos	1
No desambiguados	3

Presición detección Inconsistencia	1,00
Presición deteccion Duplicidad	0,78
Presición deteccion No Unificidad	1,00
Recuerdo deteccion Inconsistencia	1,00
Recuerdo deteccion Duplicidad	0,78
Recuerdo detección No Unificidad	1,00

A.4. Caso de estudio Carrefour

Tabla A-7 XML entrada caso de estudio carrefour

```
<?xml version="1.0" encoding="UTF-8" ?>
<KAOS>
    <Objetivos>
        <objetivo id="1"><definicion>ser líder de los supermercados de cadena</definicion>
        <fuente>El principal objetivo de esta gran empresa es ser líder de los supermercados de cadena</fuente>
         </objetivo>
        <objetivo id="2"><definicion>buscar alternativas para mejorar</definicion>
```

```
        <fuente> Debido a la gran competencia internacional debe buscar
alternativas para mejorar</fuente>
        </objetivo>
        <objetivo id="3"><definicion>vender productos de buena
calidad</definicion>
        <fuente>Esto conlleva a que Carrefour venda productos de buena
calidad</fuente> </objetivo>
        <objetivo id="4"><definicion>vender productos de mala
calidad</definicion>
                <fuente>Esto conlleva a que Carrefour venda productos de
mala calidad</fuente> </objetivo>
        <objetivo id="5"><definicion>buscar excelentes
proveedores</definicion>
        <fuente>el jefe de producción debe buscar excelentes
proveedores</fuente>
         </objetivo>
        <objetivo id="6"><definicion>existir en la empresa un control de
calidad</definicion>
        <fuente>para cumplir con la calidad, debe existir en la empresa
un control de calidad</fuente>
         </objetivo>
        <objetivo id="7"><definicion>vigilar el control de
calidad</definicion>
        <fuente>El jefe de producción tiene que vigilar el control de
calidad</fuente>
        </objetivo>
        <objetivo id="8"><definicion>brindar una buena atención a el
cliente</definicion>
        <fuente>Otro aspecto importante para buscar el liderazgo es
brindar una buena atención al cliente</fuente>
         </objetivo>
        <objetivo id="9"><definicion>contar con personal de buenas
competencias</definicion>
        <fuente>Carrefour necesita contar con personal de buenas
competencias</fuente>
         </objetivo>
        <objetivo id="10"><definicion>buscar las mejores alternativas de
tecnología limpia</definicion>
        <fuente>el jefe de personal debe buscar las mejores alternativas
de tecnología limpia</fuente>
         </objetivo>
        <objetivo id="11"><definicion>buscar las mejores alternativas de
tecnología sucia</definicion>
        <fuente>el jefe de personal debe buscar las mejores alternativas
de tecnología sucia</fuente>
         </objetivo>
        <objetivo id="12"><definicion>mejorar la atención a el
usuario</definicion>
        <fuente>al presidente le interesa mejorar la atención al
usuario</fuente>  </objetivo>
    </Objetivos>
    <Agentes>
        <agente id="13"><definicion>jefe de producción</definicion>
        <fuente>el jefe de producción debe buscar excelentes
proveedores</fuente>
         </agente>
```

```
        <agente id="14"><definicion>jefe de producción</definicion>
        <fuente>El jefe de producción tiene que vigilar el control de
calidad</fuente>
        </agente>
        <agente id="15"><definicion>Carrefour</definicion>
        <fuente>Carrefour necesita contar con personal de buenas
competencias</fuente>
         </agente>
        <agente id="16"><definicion>el jefe de personal</definicion>
        <fuente>el jefe de personal debe buscar las mejores alternativas
de tecnología</fuente>
         </agente>
    </Agentes>
</KAOS>
```

Tabla A-8 Resultados obtenidos caso de estudio carrefour

# Elemento constitutivo	Tipo	Desambiguado	Correctamente Desambiguado	Incorrectamente Desambiguado	No desambiguado
1	Objetivo	NO	NA	NA	SI
2	Objetivo	SI	SI	NO	NO
3	Objetivo	SI	SI	NO	NO
4	Objetivo	SI	SI	NO	NO
5	Objetivo	SI	SI	NO	NO
6	Objetivo	SI	SI	NO	NO
7	Objetivo	SI	SI	NO	NO
8	Objetivo	SI	NO	SI	NO
9	Objetivo	SI	SI	NO	NO
10	Objetivo	SI	SI	NO	NO
11	Objetivo	SI	SI	NO	NO
12	Agente	SI	SI	NO	NO
13	Agente	SI	SI	NA	NO
14	Agente	SI	SI	NO	NO
15	Agente	NO	NA	NA	SI
16	Agente	SI	SI	NO	NO

		Encontradas	Correctas	Incorrectas	No Encontradas
Inconsistencias Detectadas		6	5	1	2
Duplicidades		3	3	0	0

Detectadas					
No Unificidades Detectadas		0	0	0	0

CoberturaWSD	0,875
PrecisiónWSD	0,93
RecuerdoWSD	0,81
F1WSD	0,86666667

Totales	16
Desambiguados	14
Desambiguados Correctos	13
Desambiguados Incorrectos	1
No desambiguados	2

Presición detección Inconsistencia	0,83
Presición deteccion Duplicidad	1,00
Presición deteccion No Unificidad	1,00
Recuerdo deteccion Inconsistencia	0,63
Recuerdo deteccion Duplicidad	1,00
Recuerdo detección No Unificidad	1,00

A.5. Caso de estudio Colcerámicas

Tabla A-9 XML entrada caso de estudio colcerámicas.

```
<?xml version="1.0"?>
<KAOS>
    <objetivos>
        <objetivo id="4">
            <definicion>ganar daños a terceros</definicion>
            <fuente>asegurar que el sistema gane los daños a
terceros.</fuente>
        </objetivo>
        <objetivo id="5">
            <definicion>registrar los automóviles</definicion>
            <fuente>asegurar que el sistema registre los
automóviles.</fuente>
        </objetivo>
        <objetivo id="6">
            <definicion>registrar los vehículos</definicion>
            <fuente>asegurar que el sistema registre los
vehículos.</fuente>
        </objetivo>
        <objetivo id="7">
            <definicion>registrar los autos</definicion>
            <fuente>asegurar que el sistema registre los autos.</fuente>
        </objetivo>
    </objetivos>
    <agentes>
        <agente id="8">
            <definicion>el sistema negro</definicion>
```

```
            <fuente>el sistema negro debe registrar los
clientes.</fuente>
        </agente>
    <agente id="9">
        <definicion>el sistema blanco</definicion>
        <fuente>el sistema blanco debe registrar los carros.</fuente>
    </agente>
    <agente id="10">
        <definicion>el sistema</definicion>
        <fuente>el sistema negro debe registrar los carros.</fuente>
    </agente>
    <agente id="11">
            <definicion>el sistema</definicion>
            <fuente>el sistema negro debe registrar los carros.</fuente>
    </agente>
</agentes>
</KAOS>
```

Tabla A-10 Resultados obtenidos caso de estudio colcerámicas

# Elemento constitutivo	Tipo	Desambiguado	Correctamente Desambiguado	Incorrectamente Desambiguado	No desambiguado	
1	Objetivo	SI	SI	NO	NO	
2	Objetivo	SI	SI	NO	NO	
3	Objetivo	NO	NA	NA	SI	
4	Objetivo	NO	NA	NA	SI	
5	Objetivo	SI	SI	NO	NO	
6	Objetivo	SI	SI	NO	NO	
7	Objetivo	SI	SI	NO	NO	
8	Objetivo	SI	NO	SI	NO	
9	Objetivo	SI	SI	NO	NO	
10	Objetivo	SI	SI	NO	NO	
11	Objetivo	SI	SI	NO	NO	
12	Objetivo	SI	SI	NO	NO	
13	Agente	SI	SI	NA	NO	

14	Agente	SI	SI	NO	NO	
15	Agente	NO	NA	NA	SI	
16	Agente	SI	SI	NO	NO	
17	Agente	SI	NO	SI	NO	
18	Agente	SI	NO	SI	NO	
19	Agente	SI	SI	NO	NO	
20	Agente	SI	SI	NO	NO	
			Encontradas	**Correctas**	**Incorrectas**	**No Encontradas**
Inconsistencias Detectadas			2	2	0	2
Duplicidades Detectadas			8	8	0	0
No Unificidades Detectadas			0	0	0	0
CoberturaWSD		0,85		Totales	20	
PrecisiónWSD		0,82		Desambiguados	17	
RecuerdoWSD		0,70		Desambiguados Correctos	14	
F1WSD		0,756756757		Desambiguados Incorrectos	3	
				No desambiguados	3	
Presición detección Inconsistencia			1,00			
Presición deteccion Duplicidad			1,00			
Presición deteccion No Unificidad			1,00			
Recuerdo deteccion Inconsistencia			0,50			
Recuerdo deteccion Duplicidad			1,00			
Recuerdo detección No Unificidad			1,00			

A.6. Caso de estudio Aseguradora

Tabla A-11 XML entrada caso de estudio aseguradora

```
<?xml version="1.0"?>
<KAOS>
    <objetivos>
        <objetivo id="4">
            <definicion>ganar daños a terceros</definicion>
            <fuente>asegurar que el sistema gane los daños a
terceros.</fuente>
        </objetivo>
```

```
        <objetivo id="5">
            <definicion>registrar los automóviles</definicion>
            <fuente>asegurar que el sistema registre los
automóviles.</fuente>
        </objetivo>
        <objetivo id="6">
            <definicion>registrar los vehículos</definicion>
            <fuente>asegurar que el sistema registre los
vehículos.</fuente>
        </objetivo>
        <objetivo id="7">
            <definicion>registrar los autos</definicion>
            <fuente>asegurar que el sistema registre los autos.</fuente>
        </objetivo>
    </objetivos>
    <agentes>
        <agente id="8">
            <definicion>el sistema negro</definicion>
            <fuente>el sistema negro debe registrar los
clientes.</fuente>
        </agente>
    <agente id="9">
        <definicion>el sistema blanco</definicion>
        <fuente>el sistema blanco debe registrar los carros.</fuente>
    </agente>
    <agente id="10">
        <definicion>el sistema</definicion>
        <fuente>el sistema negro debe registrar los carros.</fuente>
    </agente>
    <agente id="11">
            <definicion>el sistema</definicion>
            <fuente>el sistema negro debe registrar los carros.</fuente>
    </agente>
</agentes>
</KAOS>
```

Tabla A-12 Resultados obtenidos caso de estudio aseguradora

# Elemento constitutivo	Tipo	Desambiguado	Correctamente Desambiguado	Incorrectamente Desambiguado	No desambiguado
1	Objetivo	SI	SI	NO	NO
2	Objetivo	SI	SI	NO	NO
3	Objetivo	SI	SI	NO	NO
4	Objetivo	SI	SI	NO	NO
5	Objetivo	SI	SI	NO	NO
6	Objetivo	SI	SI	NO	NO
7	Objetivo	SI	SI	NO	NO
8	Objetivo	SI	SI	NO	NO

9	Objetivo	SI	SI	NO	NO
10	Objetivo	SI	SI	NO	NO
11	Agente	SI	SI	NO	NO

	Encontradas	Correctas	Incorrectas	No Encontradas
Inconsistencias Detectadas	2	2	0	0
Duplicidades Detectadas	5	4	0	0
No Unificidades Detectadas	3	2	0	0

CoberturaWSD	1
PrecisiónWSD	1,00
RecuerdoWSD	1,00
F1WSD	1

Totales	11
Desambiguados	11
Desambiguados Correctos	11
Desambiguados Incorrectos	0
No desambiguados	0

Presición detección Inconsistencia	1,00
Presición deteccion Duplicidad	0,80
Presición deteccion No Unificidad	0,67
Recuerdo deteccion Inconsistencia	1,00
Recuerdo deteccion Duplicidad	0,80
Recuerdo detección No Unificidad	0,67

A.7. Caso de estudio Sistema de juegos

Tabla A-13 XML entrada caso de estudio sistema de juegos

```
<?xml version="1.0"?>
<KAOS>
    <objetivos>
        <objetivo id="1">
            <definicion>perder el juego</definicion>
            <fuente>el sistema debe perder el juego.</fuente>
        </objetivo>
        <objetivo id="2">
            <definicion>recuperar el juego</definicion>
            <fuente>El sistema debe recuperar el juego.</fuente>
        </objetivo>
        <objetivo id="3">
            <definicion>encontrar el juego</definicion>
            <fuente>el sistema debe encontrar el juego.</fuente>
```

```
        </objetivo>
        <objetivo id="4">
            <definicion>construir el juego</definicion>
            <fuente>El sistema debe crear el juego.</fuente>
        </objetivo>
        <objetivo id="5">
            <definicion>destruir el juego</definicion>
            <fuente>el sistema debe destruir el juego.</fuente>
        </objetivo>
        <objetivo id="6">
            <definicion>recuperar el juego</definicion>
            <fuente>El sistema debe recuperar el juego.</fuente>
        </objetivo>
        <objetivo id="7">
            <definicion>olvidar el juego</definicion>
            <fuente>el sistema debe olvidar el juego.</fuente>
        </objetivo>
        <objetivo id="8">
            <definicion>recordar el juego</definicion>
            <fuente>El sistema debe recordar el juego.</fuente>
        </objetivo>
        <objetivo id="9">
            <definicion>pintar el juego blanco</definicion>
            <fuente>el sistema debe pintar el juego blanco.</fuente>
        </objetivo>
        <objetivo id="10">
            <definicion>despintar el juego negro</definicion>
            <fuente>El sistema debe despintar el juego negro.</fuente>
        </objetivo>
        <objetivo id="11">
            <definicion>pintar el juego rápido</definicion>
            <fuente>el sistema debe pintar el juego rápido.</fuente>
        </objetivo>
        <objetivo id="12">
            <definicion>despintar el juego lento</definicion>
            <fuente>El sistema debe despintar el juego lento.</fuente>
        </objetivo>
    </objetivos>
</KAOS>
```

Tabla A-4 Resultados obtenidos caso de estudio sistema de juegos.

# Elemento constitutivo	Tipo	Desambiguado	Correctamente Desambiguado	Incorrectamente Desambiguado	No desambiguado
1	Objetivo	SI	SI	NO	NO
2	Objetivo	SI	SI	NO	NO
3	Objetivo	SI	SI	NO	NO
4	Objetivo	NO	NA	NA	SI
5	Objetivo	SI	SI	NO	NO

6	Objetivo	SI	SI	NO	NO
7	Objetivo	SI	SI	NO	NO
8	Objetivo	SI	SI	NO	NO
9	Objetivo	SI	SI	NO	NO
10	Objetivo	SI	SI	NO	NO
11	Objetivo	SI	SI	NO	NO
12	Objetivo	SI	SI	NO	NO

	Encontradas	**Correctas**	**Incorrectas**	**No Encontradas**
Inconsistencias Detectadas	2	2	0	4
Duplicidades Detectadas	3	2	0	1
No Unificidades Detectadas	3	2	0	0

CoberturaWSD	0,916666667
PrecisiónWSD	1,00
RecuerdoWSD	0,92
F1WSD	0,956521739

Totales	12
Desambiguados	11
Desambiguados Correctos	11
Desambiguados Incorrectos	0
No desambiguados	1

Presición detección Inconsistencia	1,00
Presición deteccion Duplicidad	0,67
Presición deteccion No Unificidad	0,67
Recuerdo deteccion Inconsistencia	0,33
Recuerdo deteccion Duplicidad	0,50
Recuerdo detección No Unificidad	0,67

A.8. Tabla comparativa técnicas de desambiguación vs Filtrado de coeficientes.

A continuación se presenta la tabla que agrupa las técnicas de desambiguación evaluados sobre los casos de uso utilizados, esta tabla presenta en detalle los datos de la comparación.

				Filtrado			MFS			UKB			Alesk			Slesk	
Caso	Artefacto	Correctos	Incorrectos	Vacios	Correctos	Incorrectos	Vacios	Correctos	Incorrectos	Vacios	Correctos	Incorrectos	Vacios	Correctos	Incorrectos	Vacios	Correctos	Incorrec
Ambula	1	8	0	1	6	2	1	3	5	1	0	4	5	0	3	6		
Ambula	2	3	0	0	2	1	0	2	1	0	0	1	2	0	0	3		
Ambula	3	3	0	0	1	2	0	3	0	0	0	1	2	0	0	3		
Ambula	4	8	0	0	4	4	0	7	1	0	0	6	2	0	3	5		
Ambula	5	3	0	0	1	2	0	3	0	0	0	1	2	0	0	3		
Ambula	6	2	0	1	1	1	1	1	1	1	0	1	2	0	1	2		
Ambula	7	3	0	1	2	1	1	2	1	1	0	1	3	0	0	3		
Ambula	8	5	0	1	1	4	1	5	0	1	0	3	3	0	2	4		
Ambula	9	5	0	0	2	3	0	2	3	0	2	3	0	0	0	5		
Ambula	10	7	0	1	1	6	1	6	1	1	0	5	3	0	3	5		
Ambula	11	2	0	1	1	1	1	2	0	1	0	1	2	1	0	2		
Ambula	12	6	0	1	2	4	1	5	1	1	0	4	3	3	4	0		
Ambula	13	4	0	1	1	3	1	4	0	1	0	3	2	0	2	3		
Ambula	14	6	0	1	1	5	1	5	1	1	0	3	4	0	1	6		
Ambula	15	4	0	1	1	3	1	3	1	1	0	1	4	0	0	5		
Ambula	16	5	0	0	2	3	0	5	0	0	0	2	3	0	1	4		
Ambula	17	5	0	0	1	4	0	5	0	0	0	3	2	0	1	4		
Ambula	18	1	0	1	1	0	1	1	0	0	0	0	2	0	0	2		
Ambula	19	2	0	0	0	2	0	2	0	0	0	1	1	0	0	2		
Ambula	20	2	0	0	0	2	0	2	0	0	0	1	1	0	0	2		
Ambula	21	2	0	0	1	0	1	1	0	1	0	0	2	0	0	2		
Ambula	22	1	0	0	1	0	0	1	0	0	0	0	1	0	0	1		
Asensc	1	4	0	0	2	2	0	4	0	0	0	1	3	0	1	3		
Asensc	2	3	0	0	1	2	0	3	0	0	0	1	2	0	1	3		
Asensc	3	3	0	0	1	2	0	3	0	0	0	0	3	0	0	3		
Asensc	4	3	0	0	1	2	0	3	0	0	0	3	0	0	1	2		
Asensc	5	4	0	0	1	3	0	3	0	0	0	1	3	0	1	3		
Asensc	6	5	0	0	1	4	0	4	5	0	0	4	0	1	3	2		
Asensc	7	5	0	0	1	4	0	5	0	0	4	0	1	0	3	2		
Asensc	8	5	0	0	1	4	0	5	0	0	0	1	4	0	1	4		
Asensc	9	7	0	0	4	2	1	4	3	0	0	5	2	0	4	3		
Asensc	10	4	0	0	3	0	1	3	1	0	0	1	3	1	0	3		
Asensc	11	2	0	2	1	1	2	2	0	2	0	0	4	0	0	4		
Asensc	12	3	2	1	2	1	2	3	0	2	0	1	4	0	0	5		
Asensc	13	4	0	0	2	1	2	3	0	2	0	1	4	0	1	4		
Asensc	14	4	0	0	3	1	0	3	1	0	0	0	4	0	0	4		
Asensc	15	1	0	0	1	0	0	1	0	0	0	1	0	0	0	1		
Asensc	16	1	0	0	0	0	1	0	0	1	0	0	1	0	0	1		

Asensc	17	1	0	0	1	0	0	1	0	0	0	1	0	0	0	1
Asensc	18	2	0	0	1	1	0	2	0	0	1	1	0	1	0	0
Asensc	19	2	0	0	1	1	0	2	0	0	1	1	0	1	0	1
Asensc	20	2	0	0	2	0	0	0	2	0	0	1	1	0	0	2
Asensc	21	2	0	0	2	0	0	0	2	0	0	1	1	0	0	2
Asensc	22	2	0	0	2	0	0	0	2	0	0	1	1	0	0	2
Asensc	23	2	0	0	2	0	0	1	1	0	0	1	1	0	0	2
Asensc	24	1	0	0	1	1	0	0	1	0	0	0	1	0	0	1
Asensc	25	2	0	0	2	0	0	1	0	0	0	1	0	0	0	2
Asensc	26	2	0	0	2	0	0	1	1	0	0	2	1	0	0	2
Carrefo	1	3	0	0	0	2	1	2	0	1	0	1	2	0	0	3
Carrefo	2	5	0	0	0	4	1	3	1	1	0	2	3	0	1	4
Carrefo	3	4	0	2	1	3	2	3	1	2	0	3	3	0	2	4
Carrefo	4	4	0	0	1	3	0	2	2	0	0	3	1	0	2	2
Carrefo	5	4	0	0	2	1	1	2	1	1	0	2	2	0	2	2
Carrefo	6	3	0	0	2	1	0	3	0	0	0	1	2	0	0	3
Carrefo	7	3	0	0	2	1	0	2	1	0	0	2	1	0	1	2
Carrefo	8	4	0	0	1	2	1	2	1	1	3	0	1	0	1	3
Carrefo	9	2	0	0	2	0	0	2	0	0	0	1	1	0	0	2
Carrefo	10	3	0	0	0	3	0	3	0	0	0	1	2	0	1	2
Carrefo	11	1	0	0	1	0	0	1	0	0	0	0	1	0	0	1
Carrefo	12	1	0	0	1	0	0	1	0	0	0	0	1	0	0	1
Carrefo	13	1	0	0	1	0	0	1	0	0	0	0	1	0	0	1
Carrefo	14	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1
Carrefo	15	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1
Carrefo	16	1	1	0	0	1	1	1	0	1	0	1	1	0	0	2

B. Anexo Algoritmos y Clases implementadas del sistema NEONPL.

Clase principal Teb:

```
package tseb;

import connSQL.ConnSQL;

import edu.upc.freeling.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.net.MalformedURLException;
import java.sql.SQLException;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.Map;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.io.BufferedReader;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.net.URL;

public class TSeb {

    private static final String DATA = "/usr/local/share/freeling/";
    private static final String LANG = "es";
    private static final ArrayList<String> stopwords = new ArrayList<String>();//Lista
predefinida de stopwords español
    private ArrayList<WordS> wxora;//palabras por oración
    private Map<Integer, ArrayList<WordS>> hm = new HashMap<Integer, ArrayList<WordS>>();
    private PrintWriter writer;
    private Map<String, String> palabrasdesamb = new HashMap<String, String>();
    private ArrayList<Elemento constitutivo> elementos constitutivos = new
ArrayList<Elemento constitutivo>();
    private ArrayList<Regla> reglas = new ArrayList<Regla>();
    private boolean urls = false;
    private boolean adaptedLesk = false;
```

```
    private boolean luis = false;//para comparar con el proyecto viejo
    private ConnSQL mn;
    private Map<String, ArrayList<String>> defs;
    private StringBuilder sb;
    private StringBuilder onto = new StringBuilder();
    Tokenizer tk;
    Splitter sp;
    Maco mf;
    HmmTagger tg;
    ChartParser parser;
    DepTxala dep;
    Nec neclass;
    UkbWrap dis;
    Senses sen;

    public TSeb() throws IOException, NoSuchFieldException, IllegalArgumentException,
IllegalAccessException {
        System.out.println("Cargando freeling...");
        File file = new File("sw");
        try {
            Scanner scanner = new Scanner(file);
            while (scanner.hasNextLine()) {
                stopwords.add(scanner.nextLine());
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        System.setProperty("java.library.path", "/home/personal/Descargas/freeling-
3.0/APIs/java");
        java.lang.reflect.Field fieldSysPath =
ClassLoader.class.getDeclaredField("sys_paths");
        fieldSysPath.setAccessible(true);
        fieldSysPath.set(null, null);
        System.loadLibrary("freeling javaAPI");

        Util.initLocale("default");

        // Create options set for maco analyzer.
        // Default values are Ok, except for data files.
        MacoOptions op = new MacoOptions(LANG);

        op.setActiveModules(
                false, true, true, true, true, true, true, true, true, true, false);

        op.setDataFiles(
                "",
                DATA + LANG + "/locucions.dat",
                DATA + LANG + "/quantities.dat",
                DATA + LANG + "/afixos.dat",
                DATA + LANG + "/probabilitats.dat",
                DATA + LANG + "/dicc.src",
                DATA + LANG + "/np.dat",
                DATA + "common/punct.dat",
                DATA + LANG + "/corrector/corrector.dat");

        // Create analyzers.
        tk = new Tokenizer(DATA + LANG + "/tokenizer.dat");
        sp = new Splitter(DATA + LANG + "/splitter.dat");
        mf = new Maco(op);

        tg = new HmmTagger(LANG, DATA + LANG + "/tagger.dat", true, 2);
        parser = new ChartParser(
                DATA + LANG + "/chunker/grammar-chunk.dat");
        dep = new DepTxala(DATA + LANG + "/dep/dependences.dat",
                parser.getStartSymbol());
        neclass = new Nec(DATA + LANG + "/nec/nec-ab.dat");

        //UkbWrap dis;
        dis = new UkbWrap(DATA + LANG + "/ukb.dat");
```

```
        // Instead of "UkbWrap", you can use a "Senses" object, that simply
        // gives all possible WN senses, sorted by frequency.
        sen = new Senses(DATA + LANG + "/senses.dat");

        mn = new ConnSQL();
        defs = new HashMap<String, ArrayList<String>>();

        System.out.println("Freeling cargado.");

    }
    public static void main(String[] args) throws Exception {
       TSeb ts=new TSeb();
       ts.crearXML("http://localhost/neonpl/inputs/juego.xml");
    }
    public void crearXML(String url) throws IOException
    {
        try{
            URL url2=new URL(url);
            BufferedReader reader = new BufferedReader(new
InputStreamReader(url2.openStream(), "UTF-8"));
            String content="";
            for(String line; (line = reader.readLine()) !=null;){
                content+=line;
            }
            File file = new File("temp.xml");
            FileOutputStream fop = new FileOutputStream(file);
            fop.write(content.getBytes());
            fop.flush();
            fop.close();
            cargarXml(file);
            procesar();
        } catch (Exception ex) {
            System.out.println("Error en archivo xml");
        }
    }
    public void cargarXml(File xmlFile) {
        sb = new StringBuilder();
        elementos constitutivos.clear();
        //Se crea un SAXBuilder para poder parsear el archivo
        SAXBuilder builder = new SAXBuilder();
        try {
            //Se crea el documento a traves del archivo
            org.jdom2.Document document = (org.jdom2.Document) builder.build(xmlFile);

            //Se obtiene la raiz 'documento'
            Element rootNode = document.getRootElement();

            //Se obtiene la lista de hijos de la raiz 'documento'
            List list = rootNode.getChildren();

            //Se recorre la lista de hijos de 'documento': objetivos, casos de uso,
agentes....
            for (int i = 0; i < list.size(); i++) {
                //Se obtiene el elemento 'objetivos', o 'caso de usos' o 'agentes' o...
                Element elemento = (Element) list.get(i);

                List listElements = elemento.getChildren();
                for (int j = 0; j < listElements.size(); j++) {
                    Element elemento_act = (Element) listElements.get(j);
                    //Se obtiene el atributo 'id' que está en el tag recién leído
                    String id = elemento_act.getAttributeValue("id");
                    String tipo = elemento_act.getName();
                    String def = elemento_act.getChildTextTrim("definicion");
                    String fuente = elemento_act.getChildTextTrim("fuente");
                    elementos constitutivos.add(new Elemento constitutivo(id, def, fuente,
tipo));

                }
```

```
            }

        } catch (IOException io) {
            System.out.println(io.getMessage());
        } catch (JDOMException jdomex) {
            System.out.println(jdomex.getMessage());
        }
    }

    public void procesar() {
        try {
            mn.conectarDime();
            setReglas("seagomezar");
            mn.desconectar();

            mn.conectar();

            for (int i = 0; i < elementos constitutivos.size(); i++) {

                // Extract the tokens from the line of text.
                ListWord l = tk.tokenize(elementos constitutivos.get(i).getDefinicion()+
".");

                // Split the tokens into distinct sentences.
                ListSentence ls = sp.split(l, false);

                // Perform morphological analysis
                mf.analyze(ls);

                // Perform part-of-speech tagging.
                tg.analyze(ls);

                // Perform named entity (NE) classificiation.
                neclass.analyze(ls);

                //MFS
                sen.analyze(ls);
                //UKB
                // dis.analyze(ls);elementos constitutivos.get(1).ver elemento
constitutivo();
                printResults(ls, "tagged");

                // Chunk parser
                parser.analyze(ls);
                printResults(ls, "parsed");

                // Dependency parser
                dep.analyze(ls);
                printResults(ls, "dep");

                disambiguateLesk(ls);
                for (Map.Entry entry : hm.entrySet()) {
                    ArrayList<WordS> dd = (ArrayList<WordS>) entry.getValue();
                    elementos constitutivos.get(i).setPalabrasDefinicion(dd);
                }

            }

            for (int i = 0; i < elementos constitutivos.size(); i++) {

                // Extract the tokens from the line of text.
                ListWord l = tk.tokenize(elementos constitutivos.get(i).getFuente() +
".");

                // Split the tokens into distinct sentences.
                ListSentence ls = sp.split(l, false);

                // Perform morphological analysis
```

```
                mf.analyze(ls);

                // Perform part-of-speech tagging.
                tg.analyze(ls);

                // Perform named entity (NE) classificiation.
                neclass.analyze(ls);

                //MFS
                sen.analyze(ls);
                //UKB
                // dis.analyze(ls);elementos constitutivos.get(1).ver elemento
constitutivo();
                printResults(ls, "tagged");

                // Chunk parser
                parser.analyze(ls);
                printResults(ls, "parsed");

                // Dependency parser
                dep.analyze(ls);
                printResults(ls, "dep");

                disambiguateLesk(ls);
                for (Map.Entry entry : hm.entrySet()) {
                    ArrayList<WordS> dd = (ArrayList<WordS>) entry.getValue();
                    elementos constitutivos.get(i).setPalabrasFuente(dd);
                    elementos constitutivos.get(i).filtrar_coeficientes();
                }
                setElementos constitutivos(ls, i);//Aca se hace la verificación de las
reglas


            }
            for (int i = 0; i < elementos constitutivos.size(); i++) {
            elementos constitutivos.get(i).setInconsistente(elementos constitutivos,i);
            }
            //Aca es donde se guarda.
            mn.conectarDime();
            mn.enterElementos constitutivos(elementos constitutivos);
            mn.desconectar();
        } catch (Exception ex) {
            System.out.println("errore en procesar:" + ex);
        }
    }
    public void setElementos constitutivos(ListSentence ls, int elemento constitutivoid) {
        for (int i = 0; i < ls.size(); i++) {//Para cada oracion
            Sentence s = ls.get(i);
            for (int k = 0; k < reglas.size(); k++) {
                ArrayList<Proposicion> antecedente = reglas.get(k).getAntecedente();
                int contBuenos = 0;//Cuenta cuantos elementos cumplen la regla
                for (int l = 0; l < antecedente.size(); l++) {
                    for (int j = l; j < s.size(); j++) {//Para cada palabra
                        if (!antecedente.get(l).isPalabra()) {//Si es un tag
                            if (s.get(j).getTag().startsWith(antecedente.get(l).getTag()))
{//Va cumpiendo la regla
                                contBuenos++;
                                j = (int) s.size();//Para salir del ciclo
                            } else {//Ya no cumple la regla
                                j = (int) s.size();//Para salir del ciclo interno
                                l = (int) antecedente.size();//Para salir del ciclo
externo
                            }
                        } else {//Si no es tag, es decir, si es una palabra
                            if (antecedente.get(l).getTag().equals("F") ||
antecedente.get(l).getTag().equals(s.get(j).getLemma())) {//Va cumpiendo la regla
                                contBuenos++;
                                j = (int) s.size();//Para salir del ciclo
```

```
                            } else {//Ya no cumple la regla
                                j = (int) s.size();//Para salir del ciclo interno
                                l = (int) antecedente.size();//Para salir del ciclo
externo
                            }
                        }
                        //System.out.println("contBuenos = " + contBuenos);
                    }
                }
                if (contBuenos == antecedente.size()) {//Si se cumplió la regla

                    ArrayList<Integer> consecuente = reglas.get(k).getConsecuente();
                    String texto = "";
                    for (int j = 0; j < consecuente.size(); j++) {
                        try {
                            texto += s.get(consecuente.get(j)).getLemma() + " ";
                        } catch (Exception e) {
                            System.out.println("Error");
                        }
                    }
                    ArrayList<WordS> palabras_definicion=elementos
constitutivos.get(elemento constitutivoid).getPalabras_definicion();
                    String def total="";
                    for (int i2 = 0; i2 < palabras definicion.size(); i2++) {
                        def_total+=palabras_definicion.get(i2).getLemma()+" ";
                    }
                    def_total = def_total.substring(0, def_total.length() - 3);

                    texto = texto.substring(0, texto.length() - 1);
                    if(texto.equals(def total)){
                        elementos constitutivos.get(elemento
constitutivoid).setReglas(true);
                    }
                    else
                    {
                        elementos constitutivos.get(elemento
constitutivoid).setReglas(false);
                    }
                }
            }
        }
    }

    public void setReglas(String userId) {

        mn = new ConnSQL();

        try {
            mn.conectarDime();
            this.reglas = mn.reglas(userId);
            int count = 1;
//            for (Regla temp : reglas) {
//                System.out.println("Regla " + (count++));
//                for (Proposicion temp2 : temp.getAntecedente()) {
//                    System.out.println("A.Proposición: " + temp2.getTag() + " " +
temp2.isPalabra());
//                }
//                for (Integer temp2 : temp.getConsecuente()) {
//                    System.out.println("Consecuente: " + temp2);
//                }
//            }
            mn.desconectar();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void addWord(int i, WordS w) {
        ArrayList<WordS> ws = new ArrayList<WordS>(elementos
```

```
constitutivos.get(i).palabras definicion());
        boolean flag = false;
        for (int j = 0; j < ws.size(); j++) {
            if (ws.get(j).getLemma().equals(w.getLemma())) {
                ws.get(j).setSentido(w.getSentido());
                ws.get(j).setTag(w.getTag());
                ws.get(j).setTipo(w.getTipo());
                flag = true;
                break;
            }
        }
        if (!flag) {
            ws.add(w);
        }
       // elementos constitutivos.get(i).setPalabrasFuente(ws);
    }

    private WordS putMFS(Word w) {
        String ss = w.getSensesString();
        String[] sens = ss.split("/");

        ArrayList<Sensess> sentidos = new ArrayList<Sensess>();
        WordS ww = null;
        for (int i = 0; i < sens.length; i++) {
            String[] subsens = sens[i].split("\\:");
            if (subsens.length > 1) {
                String spa="spa-30-" + subsens[0];
                try {
                    sentidos.add( new Sensess(Double.parseDouble(subsens[1]), spa,
mn.gloss(spa)));
                } catch (SQLException ex) {
                    Logger.getLogger(TSeb.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        }
        ww = new WordS(w.getForm(), w.getLemma(), w.getTag());
        ww.setMfs(sentidos);

        // The senses for a FreeLing word are a list of
        // pair<string,double> (sense and page rank). From java, we
        // have to get them as a string with format
        // sense:rank/sense:rank/sense:rank
        // which will have to be splitted to obtain the info.
        //
        // Here, we just output it:
//        System.out.print(" " + ss);
        return ww;
    }

    private WordS putUKB(WordS ww, Word w) {

        String ss = w.getSensesString();
        String[] sens = ss.split("/");
        ArrayList<Sensess> sentidos = new ArrayList<Sensess>();
        for (int i = 0; i < sens.length; i++) {
            String[] subsens = sens[i].split("\\:");
            if (subsens.length > 1) {
                String spa="spa-30-" + subsens[0];
                try {
                    sentidos.add( new Sensess(Double.parseDouble(subsens[1]), spa,
mn.gloss(spa)));
                } catch (SQLException ex) {
                    Logger.getLogger(TSeb.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        }
        ww.setUkb(sentidos);
        return ww;
```

```
    }

    private void printResults(ListSentence ls, String format) {
        if (format == "parsed") {
//            System.out.println("-------- CHUNKER results -----------");

            for (int i = 0; i < ls.size(); i++) {
                TreeNode tree = ls.get(i).getParseTree();
                printParseTree(0, tree);
            }
        } else if (format == "dep") {
//            System.out.println("-------- DEPENDENCY PARSER results -----------");

            for (int i = 0; i < ls.size(); i++) {
                TreeDepnode deptree = ls.get(i).getDepTree();
                printDepTree(0, deptree);
            }
        } else {
//            System.out.println("-------- TAGGER results -----------");
            //MFS
            //AQUÍ llegan con MFS
            // get the analyzed words out of ls.
            for (int i = 0; i < ls.size(); i++) {
                Sentence s = ls.get(i);
                wxora = new ArrayList<WordS>();
                for (int j = 0; j < s.size(); j++) {
                    Word w = s.get(j);
                    wxora.add(putMFS(w));
                }
                hm.put(i, wxora);
            }

            //UKB
            UkbWrap dis;
            dis = new UkbWrap(DATA + LANG + "/ukb.dat");
            dis.analyze(ls); //ahora obtiene el UKB

            for (int i = 0; i < ls.size(); i++) {
                Sentence s = ls.get(i);
                wxora = hm.get(i);
                for (int j = 0; j < wxora.size(); j++) {
                    wxora.set(j, putUKB(wxora.get(j), s.get(j)));
                }
                hm.put(i, wxora);
            }
        }
    }

    private static void printParseTree(int depth, TreeNode tr) {
        Word w;
        TreeNode child;
        long nch;

        // Indentation
        for (int i = 0; i < depth; i++) {
//            System.out.print("  ");
        }

        nch = tr.numChildren();

        if (nch == 0) {
            // The node represents a leaf
            if (tr.getInfo().isHead()) {
//                System.out.print("+");
            }
            w = tr.getInfo().getWord();
//            System.out.print(
//                    "(" + w.getForm() + " " + w.getLemma() + " " + w.getTag());
```

```
          //  putMFS(w);
//            System.out.println(")");
        } else {
            // The node probably represents a tree
            if (tr.getInfo().isHead()) {
//                System.out.print("+");
            }

//            System.out.println(tr.getInfo().getLabel() + " [");
            for (int i = 0; i < nch; i++) {
                child = tr.nthChildRef(i);

                if (child != null) {
                    printParseTree(depth + 1, child);
                } else {
                    System.err.println("ERROR: Unexpected NULL child.");
                }
            }
            for (int i = 0; i < depth; i++) {
//                System.out.print("  ");
            }

//            System.out.println("]");
        }
    }

    private static void printDepTree(int depth, TreeDepnode tr) {
        TreeDepnode child = null;
        TreeDepnode fchild = null;
        Depnode childnode;
        long nch;
        int last, min;
        Boolean trob;

        for (int i = 0; i < depth; i++) {
//            System.out.print("  ");
        }

//        System.out.print(
//                tr.getInfo().getLinkRef().getInfo().getLabel() + "/"
//                + tr.getInfo().getLabel() + "/");
        Word w = tr.getInfo().getWord();

//        System.out.print(
//                "(" + w.getForm() + " " + w.getLemma() + " " + w.getTag());
   //     putMFS(w);
//        System.out.print(")");

        nch = tr.numChildren();

        if (nch > 0) {
//            System.out.println(" [");

            for (int i = 0; i < nch; i++) {
                child = tr.nthChildRef(i);

                if (child != null) {
                    if (!child.getInfo().isChunk()) {
                        printDepTree(depth + 1, child);
                    }
                } else {
                    System.err.println("ERROR: Unexpected NULL child.");
                }
            }

            // Print chunks (in order)
            last = 0;
            trob = true;
```

```
            // While an unprinted chunk is found, look for the one with lower
            // chunk_ord value.
            while (trob) {
                trob = false;
                min = 9999;

                for (int i = 0; i < nch; i++) {
                    child = tr.nthChildRef(i);
                    childnode = child.getInfo();

                    if (childnode.isChunk()) {
                        if ((childnode.getChunkOrd() > last)
                                && (childnode.getChunkOrd() < min)) {
                            min = childnode.getChunkOrd();
                            fchild = child;
                            trob = true;
                        }
                    }
                }
                if (trob && (child != null)) {
                    printDepTree(depth + 1, fchild);
                }

                last = min;
            }

            for (int i = 0; i < depth; i++) {
//                System.out.print("  ");
            }

//            System.out.print("]");
        }

//        System.out.println("");
    }

    /**
     *
     * @param ls
     * @param adapted=1 si es adapted, 0 si es simplified
     */
    public void disambiguateLesk(ListSentence ls) throws MalformedURLException,
IOException, SQLException {

        for (int i = 0; i < ls.size(); i++) {//Para cada oracion
            Sentence s = ls.get(i);
            wxora = hm.get(i);

            for (int j = 0; j < s.size(); j++) {//Para cada palabra
                if (!stopwords.contains(s.get(j).getLemma())) {
                    if (!s.get(j).getTag().startsWith("F") &&
!s.get(j).getTag().equals("NP00000") && !s.get(j).getTag().startsWith("Z")) {
                        //si no es stopword:

                        wxora.set(j, overlapSimplified(s.get(j).getLemma(), s,
s.get(j).getShortTag(), i, j, s.get(j).getSensesString().split(":")[0]));
                        wxora.set(j, overlapAdapted(s.get(j).getLemma(), s,
s.get(j).getShortTag(), i, j, s.get(j).getSensesString().split(":")[0]));

                    }
                }

                hm.put(i, wxora);
            }

        }
    }
```

```
    public WordS overlapSimplified(String lemma, Sentence s, String tipo, int numOracion,
int numPalabra, String w) throws MalformedURLException, IOException, SQLException {
        try {
            //defs: lista de todos los sentidos y definiciones para la palabra que entra
(lemma)
            ArrayList<String> defsact = new ArrayList<String>();
            try {

                //SimplifiedLesk
                defsact = mn.senseDefinicion(lemma);

//                if (defsact.isEmpty()) {
//                    throw new Exception();
//                }
                int cont;

                Map<Integer, Integer> cDefs = new HashMap<Integer, Integer>();
                for (int i = 0; i < defsact.size(); i++) {

                    cont = 0;
                    String def[] = defsact.get(i).split(" ");
                    for (int k = 0; k < def.length; k++) {
                        for (int j = 0; j < s.size(); j++) {
                            if (def[k].equals(s.get(j).getLemma()) &&
!stopwords.contains(s.get(j).getLemma())) {
                                cont++;
                            }
                        }
                    }
                    cDefs.put(i, cont);
                }

                Map<Integer, Integer> sortedMap = sortByComparator(cDefs);
                ArrayList<Sensess> spas = new ArrayList<Sensess>();
                for (Map.Entry entry : sortedMap.entrySet()) {
                    if ((Integer) entry.getValue() > 0) {
                        String spa = defsact.get((Integer) entry.getKey()).split(" ")[0];
                        spas.add(new Sensess((((int)entry.getValue())/s.size()), spa,
mn.gloss(spa)));
                    }
                }

                WordS wsd = hm.get(numOracion).get(numPalabra);

                wsd.setSimlesk(spas);
                return wsd;

            } catch (Exception e) {
                System.out.println("e2:" + e);
            }

        } catch (IndexOutOfBoundsException e) {
            System.out.println("Error " + e.getLocalizedMessage() + e.getMessage());

        }
        return null;
    }

    public WordS overlapAdapted(String lemma, Sentence s, String tipo, int numOracion, int
numPalabra, String w) throws MalformedURLException, IOException, SQLException {
        try {
            //defs: lista de todos los sentidos y definiciones para la palabra que entra
(lemma)
            ArrayList<String> defsact = new ArrayList<String>();

            try {
                //AdapatedLesk
                defsact = mn.senseAdaptedDefinicion(lemma);
```

```
//                if (defsact.isEmpty()) {
//                    throw new Exception();
//                }
                int cont;

                Map<Integer, Integer> cDefs = new HashMap<Integer, Integer>();
                for (int i = 0; i < defsact.size(); i++) {

                    cont = 0;
                    String def[] = defsact.get(i).split(" ");
                    for (int k = 0; k < def.length; k++) {
                        for (int j = 0; j < s.size(); j++) {
                            if (def[k].equals(s.get(j).getLemma()) &&
!stopwords.contains(s.get(j).getLemma())) {
                                cont++;

                            }
                        }
                    }
                    cDefs.put(i, cont);
                }

                Map<Integer, Integer> sortedMap = sortByComparator(cDefs);
                ArrayList<Sensess> spas = new ArrayList<Sensess>();
                for (Map.Entry entry : sortedMap.entrySet()) {
                    if ((Integer) entry.getValue() > 0) {
                        String spa = defsact.get((Integer) entry.getKey()).split(" ")[0];
                        spas.add(new Sensess((((int)entry.getValue())/s.size()), spa,
mn.gloss(spa)));
                    }
                }
                WordS wsd = hm.get(numOracion).get(numPalabra);
                wsd.setAdaplesk(spas);
                return wsd;
            } catch (Exception e) {
                System.out.println("e4:" + e);
            }

        } catch (IndexOutOfBoundsException e) {
            System.out.println("Error " + e.getLocalizedMessage() + e.getMessage());

        }
        return null;
    }

    private static Map sortByComparator(Map unsortMap) {

        List list = new LinkedList(unsortMap.entrySet());

        // sort list based on comparator
        Collections.sort(list, new Comparator() {
            public int compare(Object o1, Object o2) {
                return ((Comparable) ((Map.Entry) (o1)).getValue())
                        .compareTo(((Map.Entry) (o2)).getValue());
            }
        });

        // put sorted list into map again
        //LinkedHashMap make sure order in which keys were inserted
        Map sortedMap = new LinkedHashMap();
        for (Iterator it = list.iterator(); it.hasNext();) {
            Map.Entry entry = (Map.Entry) it.next();
            sortedMap.put(entry.getKey(), entry.getValue());
        }
        return sortedMap;
    }

    public static ArrayList<String> getStopwords() {
```

```
        return stopwords;
    }

}
```

Clase elemento constitutivo Teb:

```
package tseb;

import connSQL.ConnSQL;
import java.util.ArrayList;

/**
 *
 * @author personal
 */
public class Elemento constitutivo {

    private String id;
    private String definicion;
    private String fuente;
    private String tipo;
    private boolean reglas;
    private ConnSQL mn = new ConnSQL();
    private boolean inconsistente;
    private boolean duplicado;
    private boolean unificado;
    private String inconsistente__con;
    private String duplicado__con;
    private String no_unificado__con;
    private ArrayList<WordS> palabras_definicion = new ArrayList<WordS>();
    private ArrayList<WordS> palabras_fuente = new ArrayList<WordS>();

    public String getInconsistente__con() {
        return inconsistente__con;
    }

    public void setInconsistente__con(String inconsistente__con) {
        this.inconsistente__con = inconsistente__con;
    }

    public String getDuplicado__con() {
        return duplicado__con;
    }

    public void setDuplicado__con(String duplicado__con) {
        this.duplicado__con = duplicado__con;
    }

    public String getNo_unificado__con() {
        return no_unificado__con;
    }

    public void setNo_unificado__con(String no_unificado__con) {
        this.no_unificado__con = no_unificado__con;
    }

    public int isReglas() {
        if (reglas) {
            return 1;
        } else {
            return 0;
        }
    }
```

```
    public void setReglas(boolean reglas) {
        this.reglas = reglas;
    }

    public void setInconsistente(ArrayList<Elemento constitutivo> elementos constitutivos,
int elemento constitutivoId) {
        this.inconsistente = false;
        for (int i = 0; i < elementos constitutivos.size(); i++) {
            if (elemento constitutivoId != i) {
                for (int j = 0; j < elementos
constitutivos.get(i).getPalabras_definicion().size(); j++) {
                    for (int k = 0; k < this.palabras_definicion.size(); k++) {
                        if (this.esOpuesto(this.palabras_definicion.get(k).getSentido(),
elementos constitutivos.get(i).getPalabras_definicion().get(j).getSentido())) {
                            this.inconsistente = true;
                            j = elementos
constitutivos.get(i).getPalabras_definicion().size();
                            k = this.palabras_definicion.size();
                            this.inconsistente__con += "," + elementos
constitutivos.get(i).getId();
                        }
                    }
                }
            }
        }
    }

    public boolean esOpuesto(String spa1, String spa2) {
        try {
            mn.conectar();
            ArrayList<String> antonimos = mn.antonims(spa1);
            for (int i = 0; i < antonimos.size(); i++) {
                if (spa2.equals(antonimos.get(i))) {
                    return true;
                }
            }
            mn.desconectar();
        } catch (Exception ex) {

        }
        return false;
    }

    public void setInconsistente(boolean inconsistente) {
        this.inconsistente = inconsistente;
    }

    public boolean isDuplicado() {
        return duplicado;
    }

    public void setDuplicado(boolean duplicado) {
        this.duplicado = duplicado;
    }

    public boolean isUnificado() {
        return unificado;
    }

    public void setUnificado(boolean unificado) {
        this.unificado = unificado;
    }

    public ArrayList<WordS> getPalabras_definicion() {
        return palabras_definicion;
    }

    public void setPalabras_definicion(ArrayList<WordS> palabras_definicion) {
```

```
        this.palabras_definicion = palabras_definicion;
    }

    public Elemento constitutivo(String id, String def, String fuente, String tipo) {
        this.id = id;
        definicion = def;
        this.tipo = tipo;
        this.fuente = fuente;
    }

    public Elemento constitutivo(String id, String def, String fuente, String tipo,
ArrayList<WordS> palabras_definicion) {
        this.id = id;
        definicion = def;
        this.tipo = tipo;
        this.fuente = fuente;
        this.palabras_definicion = palabras_definicion;
    }

    public String getDefinicion() {
        return definicion;
    }

    public void setDefinicion(String definicion) {
        this.definicion = definicion;
    }

    public String getTipo() {
        return tipo;
    }

    public String getFuente() {
        return fuente;
    }

    public String getId() {
        return id;
    }

    public void setFuente(String fuente) {
        this.fuente = fuente;
    }

    public void setTipo(String tipo) {
        this.tipo = tipo;
    }

    public ArrayList<WordS> palabras_definicion() {
        return palabras_definicion;
    }

    public void setPalabrasDefinicion(ArrayList<WordS> palabras_definicion) {
        this.palabras_definicion = new ArrayList<WordS>(palabras_definicion);
    }

    public void addPalabraDefnicion(WordS palabra) {
        palabras_definicion.add(palabra);
    }

    public ArrayList<WordS> palabras_fuente() {
        return palabras_definicion;
    }

    public void setPalabrasFuente(ArrayList<WordS> palabras_fuente) {
        this.palabras_fuente = new ArrayList<WordS>(palabras_fuente);
    }

    public void addPalabraFuente(WordS palabra) {
```

```
        palabras_fuente.add(palabra);
    }

    public void filtrar_coeficientes() {


        this.normalizarCoeficientes(); //Normalizamos todos los coeficientes para que
queden valores entre 0 y 1 con 3 cifras decimales.
        int count = 0;
        for (int i = 0; i < palabras_fuente.size(); i++) {

            double score = 0;
            double max_filtro = 0;
            String best_sense = "";
            String spa0 = "Vacio";
            String def = "Vacio";
            String definicion_sentido = "Vacio";
            for (int j1 = 0; j1 < palabras_fuente.get(i).getMfs().size(); j1++) {
                double sumatoriaMinimos = 0;
                double minimo1 = 0, minimo2 = 0, minimo3 = 0, minimo4 = 0;
                score = palabras_fuente.get(i).getMfs().get(j1).rank;
                minimo1 = palabras_fuente.get(i).getMfs().get(j1).rank;
                spa0 = palabras_fuente.get(i).getMfs().get(j1).spa;
                def = palabras_fuente.get(i).getMfs().get(j1).definicion;
                for (int j2 = 0; j2 < palabras_fuente.get(i).getUkb().size(); j2++) {
                    if (palabras_fuente.get(i).getUkb().get(j2).spa.equals(spa0)) {
                        score += palabras_fuente.get(i).getUkb().get(j2).rank;
                        minimo2 = palabras_fuente.get(i).getUkb().get(j2).rank;
                    }
                }
                for (int j3 = 0; j3 < palabras_fuente.get(i).getSimlesk().size(); j3++) {
                    if (palabras_fuente.get(i).getSimlesk().get(j3).spa.equals(spa0)) {
                        score += palabras_fuente.get(i).getSimlesk().get(j3).rank;
                        minimo3 = palabras_fuente.get(i).getSimlesk().get(j3).rank;
                    }
                }
                for (int j4 = 0; j4 < palabras_fuente.get(i).getAdaplesk().size(); j4++) {
                    if (palabras_fuente.get(i).getAdaplesk().get(j4).spa.equals(spa0)) {
                        score += palabras_fuente.get(i).getAdaplesk().get(j4).rank;
                        minimo4 = palabras_fuente.get(i).getAdaplesk().get(j4).rank;
                    }
                }
                double promedio = score / 4;
                sumatoriaMinimos = Math.pow((minimo1 - promedio), 2) + Math.pow((minimo2 -
promedio), 2) + Math.pow((minimo3 - promedio), 2) + Math.pow((minimo4 - promedio), 2);
                double filtro = promedio * (sumatoriaMinimos / 4);
                if (filtro > max_filtro) {
                    max_filtro = filtro;
                    best_sense = spa0;
                    definicion_sentido = def;
                    palabras_fuente.get(i).setDef_sentido(definicion_sentido);
                    palabras_fuente.get(i).setSentido(best_sense);
                }
            }
        }
        int max = palabras_definicion.size();
        count = 0;
        for (int i2 = 0; i2 < max; i2++) {
            for (int i = count; i < palabras_fuente.size(); i++) {
                if
(palabras_definicion.get(i2).getLemma().equals(palabras_fuente.get(i).getLemma())) {

palabras_definicion.get(i2).setLemma(palabras_fuente.get(i).getLemma());

palabras_definicion.get(i2).setSentido(palabras_fuente.get(i).getSentido());
                    palabras_definicion.get(i2).setTag(palabras_fuente.get(i).getTag());

palabras_definicion.get(i2).setDef_sentido(palabras_fuente.get(i).getDef_sentido());
                    count = i;
```

```
                    i = palabras_fuente.size();
                }
            }

        }

    }

    public void normalizarCoeficientes() {
        for (int i = 0; i < palabras_fuente.size(); i++) {
            for (int j1 = 0; j1 < palabras_fuente.get(i).getMfs().size(); j1++) {
                palabras_fuente.get(i).getMfs().get(j1).rank = (double) (((double)
palabras_fuente.get(i).getMfs().size() - (double) j1) / (double)
palabras_fuente.get(i).getMfs().size() / 1000);
            }
            for (int j1 = 0; j1 < palabras_fuente.get(i).getAdaplesk().size(); j1++) {
                palabras_fuente.get(i).getAdaplesk().get(j1).rank = (double) (((double)
palabras_fuente.get(i).getAdaplesk().size() - (double) j1) / (double)
palabras_fuente.get(i).getAdaplesk().size() / 1000);
            }
            for (int j1 = 0; j1 < palabras_fuente.get(i).getSimlesk().size(); j1++) {
                palabras_fuente.get(i).getSimlesk().get(j1).rank = (double) (((double)
palabras_fuente.get(i).getSimlesk().size() - (double) j1) / (double)
palabras_fuente.get(i).getSimlesk().size() / 1000);
            }
        }
    }
}
```

Bibliografía

[1] J. Zavala Ruiz, «?` Por qué fracasan los proyectos software?. Un enfoque organizacional», en *Congreso Nacional de Software Libre*, 2004.

[2] R. Pressman, *Ingeniería del software. Un enfoque práctico. Sexta edición. Editoria l McGraw-Hill*. Interamericana Editores, SA de CV México, 2005.

[3] I. Sommerville, *Ingeniería del software*. Pearson Educación, 2005.

[4] D. Lowe y W. Hall, *Hypermedia & the Web*. Wiley Chichester, 1999.

[5] J. M. Quinteiro-González, E. Martel-Jordán, P. Hernández-Morera, J. A. Ligero-Fleitas, y A. López-Rodriguez, «Clasificación de textos en lenguaje natural usando la Wikipedia.», *RISTI Iber. J. Inf. Syst. Technol. Ibérica Sist. E Tecnol. Informação*, n.º 8, 2011.

[6] J. A. M. González, *El contenido de los documentos textuales: su análisis y representación mediante el lenguaje natural*. Trea, 2004.

[7] G. Booch, J. Rumbaugh, I. Jacobson, J. S. Martínez, y J. J. G. Molina, *El lenguaje unificado de modelado*, vol. 1. Addison-Wesley, 1999.

[8] K. Ryan, «The role of natural language in requirements engineering», en *Requirements Engineering, 1993., Proceedings of IEEE International Symposium on*, 1993, pp. 240–242.

[9] P. Loucopoulos y V. Karakostas, *System requirements engineering*. McGraw-Hill, Inc., 1995.

[10] V. A. de S. Júnior y N. L. Vijaykumar, «Generating model-based test cases from natural language requirements for space application software», *Softw. Qual. J.*, vol. 20, n.º 1, pp. 77-143, mar. 2012.

[11] V. Ambriola y V. Gervasi, «On the Systematic Analysis of Natural Language Requirements with CIRCE», *Autom. Softw. Eng.*, vol. 13, n.º 1, pp. 107-167, ene. 2006.

[12] R. Peña, «Diseño de programas. Formalismo y abstracción», *Prentice Hall*, 1998.

[13] C. M. Z. Jaramillo, A. Gelbukh, y F. A. Isaza, «Pre-conceptual schema: A conceptual-graph-like knowledge representation for requirements elicitation», en *MICAI 2006: Advances in Artificial Intelligence*, Springer, 2006, pp. 27–37.

[14] Luis Alfonso Lezcano Rodriguez, «Un modelo de procesamiento terminológico para la obtención y validación de requisitos de software basado en el diagrama de objetivos de KAOS.», Tesis de Doctorado, Universidad Nacional De Colombia Sede Medellin, Medellín, Antioquia, 2014.

[15] P. More y R. Phalnikar, «Generating UML Diagrams from Natural Language Specifications», *Int. J. Appl. Inf. Syst.*, vol. 1, n.º 8, pp. 19–23, 2012.

[16] C. A KAOS Tutorial, «Recuperado el 03 agosto de 2012 de http://www. objectiver. com/fileadmin/downlo ad/documents», *KaosTutorial Pdf*, 2007.

[17] F. Almisned y J. Keppens, «Requirements Analysis: Evaluating KAOS Models.», *J. Softw. Eng. Appl.*, vol. 3, n.º 9, 2010.

[18] E. Letier y A. Van Lamsweerde, «Reasoning about partial goal satisfaction for requirements and design engineering», en *ACM SIGSOFT Software Engineering Notes*, 2004, vol. 29, pp. 53–62.
[19] A. Lapouchnian, «Goal-oriented requirements engineering: An overview of the current research», *Univ. Tor.*, 2005.
[20] R. Matulevi\vcius, P. Heymans, y A. L. Opdahl, «Ontological analysis of KAOS using separation of reference», en *Proc. EMMSAD*, 2006, vol. 6.
[21] A. S. Cueto, «Resolución de la Ambigüedad Semántica de las palabras mediante Modelos de Probabilidad de Máxima Entropıa», 2004.
[22] A. M. Guijarro, «Resolución de la ambigüedad semántica mediante métodos basados en conocimiento y su aportación a tareas de PLN», 2009.
[23] S. TORRES RAMOS, «APRENDIZAJE SUPERVISADO DE COLOCACIONES PARA LA RESOLUCION DE LA AMBIGUEDAD SINTACTICA», 2008.
[24] R. Guastini, «Problemas de interpretación», *Isonomía Rev. Teoría Filos. Derecho*, n.º 7, pp. 121–131, 1997.
[25] R. Krovetz y W. B. Croft, «Lexical ambiguity and information retrieval», *ACM Trans. Inf. Syst. TOIS*, vol. 10, n.º 2, pp. 115–141, 1992.
[26] A. Suárez, «Resolución de la Ambigüedad Semántica de las palabras mediante Modelos de Probabilidad de Máxima Entropía.», *Proces. Leng. Nat.*, vol. 34, 2005.
[27] E. Agirre y G. Rigau, «Word sense disambiguation using conceptual density», en *Proceedings of the 16th conference on Computational linguistics-Volume 1*, 1996, pp. 16–22.
[28] G. Nenadić, I. Spasić, y S. Ananiadou, «Morpho-syntactic clues for terminological processing in Serbian», en *Proceedings of the 2003 EACL Workshop on Morphological Processing of Slavic Languages*, 2003, pp. 79–86.
[29] R. Navigli, «Word sense disambiguation: A survey», *ACM Comput. Surv. CSUR*, vol. 41, n.º 2, p. 10, 2009.
[30] R. Navigli y D. Vannella, «SemEval-2013 task 11: Evaluating word sense induction & disambiguation within an end-user application», en *Proceedings of the 7th International Workshop on Semantic Evaluation (SemEval 2013), in conjunction with the Second Joint Conference on Lexical and Computational Semantcis (* SEM 2013), Atlanta, USA*, 2013.
[31] J. Artiles, J. Gonzalo, y S. Sekine, «The semeval-2007 weps evaluation: Establishing a benchmark for the web people search task», en *Proceedings of the 4th International Workshop on Semantic Evaluations*, 2007, pp. 64–69.
[32] G. Francopoulo, N. Bel, M. George, N. Calzolari, M. Monachini, M. Pet, y C. Soria, «Lexical markup framework: ISO standard for semantic information in NLP lexicons», en *Proceedings of the Workshop of the GLDV Working Group on Lexicography at the Biennial Spring Conference of the GLDV*, 2007.
[33] R. Navigli, «Word sense disambiguation: A survey», *ACM Comput. Surv. CSUR*, vol. 41, n.º 2, p. 10, 2009.
[34] R. L. Rivest, «Learning decision lists», *Mach. Learn.*, vol. 2, n.º 3, pp. 229–246, 1987.
[35] J. R. Quinlan, «Induction of decision trees», *Mach. Learn.*, vol. 1, n.º 1, pp. 81–106, 1986.
[36] G. Escudero, L. Màrquez, G. Rigau, y J. G. Salgado, «On the portability and tuning of supervised word sense disambiguation systems», 2000.
[37] G. Cottrell y J. Allen, «A connectionist approach to word sense disambiguation», 1985.

[38] H. T. Ng, «Getting serious about word sense disambiguation», en *Proceedings of the ACL SIGLEX Workshop on Tagging Text with Lexical Semantics: Why, What, and How*, 1997, pp. 1–7.
[39] B. E. Boser, I. M. Guyon, y V. N. Vapnik, «A training algorithm for optimal margin classifiers», en *Proceedings of the fifth annual workshop on Computational learning theory*, 1992, pp. 144–152.
[40] H. Schutze, «Dimensions of meaning», en *Supercomputing'92., Proceedings*, 1992, pp. 787–796.
[41] D. Lin, «Automatic retrieval and clustering of similar words», en *Proceedings of the 17th international conference on Computational linguistics-Volume 2*, 1998, pp. 768–774.
[42] D. Widdows y B. Dorow, «A graph model for unsupervised lexical acquisition», en *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, 2002, pp. 1–7.
[43] M. Lesk, «Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone», en *Proceedings of the 5th annual international conference on Systems documentation*, 1986, pp. 24–26.
[44] S. Banerjee y T. Pedersen, «An adapted Lesk algorithm for word sense disambiguation using WordNet», en *Computational linguistics and intelligent text processing*, Springer, 2002, pp. 136–145.
[45] J. Preiss, J. Dehdari, J. King, y D. Mehay, «Refining the most frequent sense baseline», en *Proceedings of the NAACL HLT Workshop on Semantic Evaluations: Recent Achievements and Future Directions*, 2009, n.º June, pp. 10-18.
[46] A. Gonzalez-Agirre, E. Laparra, y G. Rigau, «Multilingual Central Repository version 3.0», *Proc. 8th Int. Conf. Lang. Resour. Eval.*, pp. 2525-2529, 2012.
[47] L. Padró, S. Reese, E. Agirre, y A. Soroa, «Semantic services in freeling 2.1: Wordnet and ukb», 2010.
[48] E. Agirre y A. Soroa, «Using the Multilingual Central Repository for Graph-Based Word Sense Disambiguation.», en *LREC*, 2008.
[49] J. M. G. Hidalgo, A. D. Esteban, L. A. Ureña, y M. García, «Utilización y evaluación de la desambiguación en tareas de clasificación de texto».
[50] A. Kilgarri y M. Palmer, «Introduction to the Special Issue on SENSEVAL».
[51] K. C. Litkowski, «SENSEVAL Word-Sense Disambiguation Using a Different Sense Inventory and Mapping to WordNet».
[52] P. Edmonds y S. Cotton, «senseval-2: Overview», en *The Proceedings of the Second International Workshop on Evaluating Word Sense Disambiguation Systems*, 2001, pp. 1–5.
[53] H. Yang, A. Willis, A. De Roeck, y B. Nuseibeh, «Automatic detection of nocuous coordination ambiguities in natural language requirements», en *Proceedings of the IEEE/ACM international conference on Automated software engineering*, 2010, pp. 53–62.
[54] S. Nanduri y S. Rugaber, «Requirements validation via automated natural language parsing», en *System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on*, 1995, vol. 3, pp. 362–368.
[55] S. Spreeuwenberg y K. A. Healy, «SBVR's approach to controlled natural language», en *Controlled Natural Language*, Springer, 2010, pp. 155–169.
[56] I. S. Bajwa, M. G. Lee, y B. Bordbar, «SBVR Business Rules Generation from Natural Language Specification.», en *AAAI Spring Symposium: AI for Business Agility*, 2011.

[57] C. Videira, D. Ferreira, y A. R. Da Silva, «A linguistic patterns approach for requirements specification», en *Software Engineering and Advanced Applications, 2006. SEAA'06. 32nd EUROMICRO Conference on*, 2006, pp. 302–309.
[58] C. Videira y A. R. Da Silva, «Patterns and metamodel for a natural-language-based requirements specification language.», en *CAiSE Short Paper Proceedings*, 2005.
[59] K. Hinge, A. Ghose, y G. Koliadis, «Process seer: A tool for semantic effect annotation of business process models», en *Enterprise Distributed Object Computing Conference, 2009. EDOC'09. IEEE International*, 2009, pp. 54–63.
[60] M. Osborne y C. K. MacNish, «Processing natural language software requirement specifications», en *Requirements Engineering, 1996., Proceedings of the Second International Conference on*, 1996, pp. 229–236.
[61] A. R. Luria, «The development of writing in the child», *J. Russ. East Eur. Psychol.*, vol. 16, n.º 2, pp. 65–114, 1977.
[62] A. R. Luria y L. V. Majovski, «Basic approaches used in American and Soviet clinical neuropsychology.», *Am. Psychol.*, vol. 32, n.º 11, p. 959, 1977.
[63] N. Chomsky, «Recensión crítica de" Berval Behavior" de BF Skinner», *Convivium*, n.º 38, pp. 65–105, 1959.
[64] N. Chomsky y C. P. Otero, *Aspectos de la teoría de la sintaxis*. Aguilar, 1976.
[65] N. Chomsky, *The minimalist program*, vol. 28. Cambridge Univ Press, 1995.
[66] M. Attia, «An Ambiguity-Controlled Morphological Analyzer for Modern Standard Arabic Modelling Finite State Networks», en *Challenges of Arabic for NLP/MT Conference, The British Computer Society, London, UK*, 2006.
[67] D. Z. Hakkani-Tür, K. Oflazer, y G. Tür, «Statistical morphological disambiguation for agglutinative languages», *Comput. Humanit.*, vol. 36, n.º 4, pp. 381–410, 2002.
[68] Y.-G. Hwang, H.-Y. Lee, W.-J. Bae, y Y.-S. Lee, «Resolution strategy of morphological ambiguity for Korean parsing», *Int. J. Comput. Process. Orient. Lang.*, vol. 13, n.º 02, pp. 97–112, 2000.
[69] L. M. Elissalt, «De la ambigüedad sintáctica», *Rev. Esp. Lingüíst.*, vol. 2, n.º 2, pp. 237–248, 1972.
[70] R. López Sánchez, *Resolución de la ambigüedad sintáctica basada en las expectativas léxico-semánticas*. Universidad Complutense de Madrid, Servicio de Publicaciones, 2002.
[71] C. M. Zapata Jaramillo, K. C. Palomino Zuluaga, y R. Rosero, «Un método para la desambiguación sintáctica de tipo coordinativo y preposicional», *Dyna*, vol. 75, n.º 156, pp. 29–42, 2008.
[72] G. D. Hadad, «Panorama de la Ingeniería de Requisitos: sus fundamentos y avances», 2011.
[73] S. Patwardhan, S. Banerjee, y T. Pedersen, «Using measures of semantic relatedness for word sense disambiguation», en *Computational linguistics and intelligent text processing*, Springer, 2003, pp. 241–257.
[74] E. Agirre y A. Soroa, «Personalizing pagerank for word sense disambiguation», en *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, 2009, pp. 33–41.
[75] M. Rios y A. Gelbukh, «Recognizing Textual Entailment with a Semantic Edit Distance Metric», 2012, pp. 15-20.
[76] M. Zampieri, «Evaluating knowledge-poor and knowledge-rich features in automatic classification: A case study in WSD», en *Computational Intelligence and Informatics (CINTI), 2012 IEEE 13th International Symposium on*, 2012, pp. 359–363.
[77] C. L. Lawson y R. J. Hanson, *Solving least squares problems*, vol. 161. SIAM, 1974.

[78] L. Padró y E. Stanilovsky, «Freeling 3.0: Towards wider multilinguality», 2012.
[79] A. Van Lamsweerde y others, «Requirements engineering: from system goals to UML models to software specifications», 2009.
[80] E. Letier, «Reasoning about agents in goal-oriented requirements engineering», PhD thesis, Université catholique de Louvain, 2001.
[81] C. M. Zapata, S. M. Villegas, y F. Arango, «Reglas de consistencia entre modelos de requisitos de UN-Metodo», *Rev. Univ. Eafit*, vol. 42, n.º 141, pp. 40–59, 2012.