REVISTA VIRTUAL
Universidad Católica del Norte

CATÓLICA DEL NORTE
Fundación Universitaria
Pioneros en educación virtual

http://revistavirtual.ucn.edu.co

No. 46 | Septiembre - Diciembre, 2015

ISSN-0124-5821 (en línea)

Cómo citar el artículo
Guzmán-Luna, J.A., Gómez Arias, S.A. & Vélez-Carvajal, C.A. (2015). Un modelo de procesamiento de lenguaje natural para la detección de errores en requisitos de software. *Revista Virtual Universidad Católica del Norte*, 46, 169-186. Recuperado de http://revistavirtual.ucn.edu.co/index.php/RevistaUCN/article/view/707/1234

# Un modelo de procesamiento de lenguaje natural para la detección de errores en requisitos de software*

## A Natural-Language Processing Model for Detecting Errors in Software Requirements

## Un modèle de traitement du langage naturel pour détecter des erreurs dans spécifications du logiciel

**Jaime Alberto Guzmán-Luna**
Ingeniero Civil
Magister en Ingeniería de Sistemas
Doctor en Ingeniería
Docente Asociado Universidad Nacional de Colombia
Director Grupo Investigación SINTELWEB
jaguzman@unal.edu.co

**Sebastián Alonso Gómez Arias**
Ingeniero de Sistemas
Magíster en Ingeniería de Sistemas
Integrante Grupo de Investigación SINTELWEB
seagomezar@unal.edu.co

**Carlos Andrés Vélez-Carvajal**
Ingeniero de Sistemas
Integrante Grupo de Investigación SINTELWEB
caavelezca@unal.edu.co

Recibido: 22 de enero de 2015
Evaluado: 11 de agosto de 2015
Aprobado: 28 de agosto de 2015
Tipo de artículo: investigación científica y tecnológica

* "Un modelo de procesamiento de lenguaje natural para la detección de errores en requisitos de software" es uno de los resultados del proyecto de investigación "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", financiado por Colciencias mediante la convocatoria Jóvenes Investigadores 2013, Código Hermes 21481, Universidad Nacional de Colombia Sede Medellín.

Indexada Publindex-Colciencias (B), Latindex, EBSCO InformationServices,Redalyc, Dialnet, DOAJ, Actualidad Iberoamericana, Índice de Revistas de Educación Superior e Investigación Educativa (IRESIE) de la Universidad Autónoma de México.

**Resumen**

La ambigüedad semántica polisémica, inherente al lenguaje natural, afecta la interpretación de los requisitos de software, generando errores en su especificación por los múltiples significados que puede tener una palabra. Algunos de los errores generados debido a una mala interpretación de los requisitos de software son: inconsistencia, duplicidad y falta de unicidad. En este artículo, se presenta un modelo de procesamiento de lenguaje natural que permite detectar estos errores automáticamente en requisitos de software desde el idioma español. La metodología usada corresponde a la definición e implementación de un conjunto de reglas que ayudan a detectar dichos errores usando una técnica de desambiguación semántica polisémica llamada filtrado de coeficientes. Se realizaron un conjunto de pruebas sobre siete casos de estudio diferentes para valorar el modelo y se obtuvo, en promedio, una eficiencia en la detección de dichos errores del 85%.

**Palabras clave**

Ambigüedad semántica polisémica, Desambiguación de los sentidos de las palabras, Ingeniería de requisitos, Ingeniería de software.

**Abstract**

The semantic polysemic ambiguity, inherent to natural language affects the interpretation of software requirements, producing errors in its specification because of the multiple meanings a word can have. Some of the errors generated by a bad interpretation of software requirements are: inconsistency, duplicity and lack of uniqueness. This article presents a model for processing natural language that allows automatically detecting this errors in software requirements from Spanish language. The used methodology is related to the definition and implementation of a set of rules that help to detect such errors by using a polysemic semantic disambiguation technique called coefficient filtering. A set of tests were performed with seven case studies in order to assess the model, it was obtained, on average, an 85% efficiency when detecting this errors.

**Keywords**

Polysemic semantic ambiguity, Disambiguation of the meaning of words, Requirement engineering, Software engineering.

**Résumé**

L'ambiguïté sémantique polysémique, qui est inhérent au langage naturel, affecte l'interprétation des spécifications du logiciel, en produisant des erreurs dans les spécifications à cause des multiples significations qui peuvent avoir un mot. Quelques des erreurs produits à cause d'une mauvaise interprétation des spécifications du logiciel sont: inconsistance, duplicité et manque d'unicité. Dans cet article on présente un modèle de traitement du langage naturel qui permet de détecter ces erreurs de manière automatique dans spécifications du logiciel dans l'espagnol. La méthodologie utilisée se base sur la définition et implémentation d'un ensemble de règles qui aident à détecter tels erreurs en utilisant une technique de désambigüation sémantique polysémique appelé filtrage de coefficients. On a réalisé un ensemble de tests avec sept cas d'étude différents pour évaluer le modèle et on a obtenu, comme moyenne, une efficacité dans la détection d'erreurs de 85%.

**Mots-clés**

Ambiguïté sémantique polysémique, Désambigüation des significations des mots, Ingénierie des exigences, Génie logiciel.

# Introducción

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 en relación con el problema o necesidad que se desea computarizar. Esta fase permite a su vez identificar, analizar y validar los requisitos que el futuro software deberá satisfacer. Sin embargo, según Booch *et al.* (1999), se presentan problemas en la comunicación entre el analista y el interesado debido a:

a. La 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.

b. Los tipos de ambigüedades presentes en el discurso, tales como la léxica o morfológica, la sintáctica o estructural y la ambigüedad semántica (polisémica y referencial), que dificulta la comprensión correcta y no ambigua de los requisitos.

c. 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.

d. La poca claridad por parte del interesado al momento de expresar sus necesidades, dando lugar a

más ambigüedades en las especificaciones de requisitos.

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 un texto en lenguaje natural que consigna el conocimiento de los interesados (Arango & Zapata, 2006).

Los requisitos de software generalmente son plasmados en diagramas o modelos conceptuales que hacen parte de modelos de educción de requisitos (tales como KAOS, casos de uso, diagramas de actividades y modelos entidad relación, entre otros) y es en estos modelos donde se evidencian las fallas en el proceso de educción de requisitos, debido a que se encuentran requisitos ambiguos, duplicados, inconsistentes y no unificados. Es por esto que aproximadamente el 53% de los proyectos de software que fracasan, lo hacen debido a errores e inconsistencias en la fase de educción de requisitos (Ruiz, 2004).

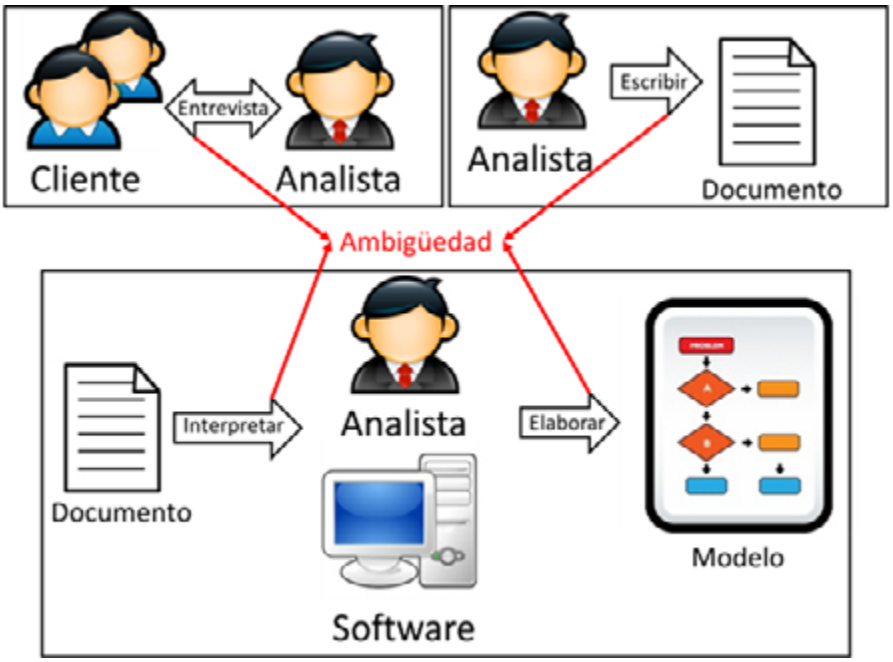


**Figura 1. Problema de ambigüedad en la educción de requisitos de software.**
**Fuente: Elaboración Propia**

Para enfrentar el problema de la ambigüedad semántica polisémica y los errores que esta ambigüedad genera (Nerlich & Domínguez, 1999), en este artículo se propone un modelo para la detección de errores en requisitos de software usando una técnica de desambiguación semántica polisémica.

# Antecedentes

Los trabajos relacionados se revisaron de acuerdo a dos criterios: i) la desambiguación de requisitos de software de manera automática y ii) si intentan solucionar o no algún problema asociado a esta ambigüedad. A continuación, se presentan algunos de estos trabajos:

En Ambriola y Gervasi (2006) se presenta un modelo para la generación automática de modelos de requisitos de software utilizando herramientas morfosintácticas para analizarlos. Sin embargo, el autor no hace referencia a la ambigüedad semántica, dado que simplemente se predefine cada término utilizado en un glosario. 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 Nanduri y Rugaber (1995) 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 el problema de ambigüedad semántica inherente a los requisitos de software extraídos del lenguaje natural.

En Spreeuwenberg y Healy (2010) y Bajwa, Lee y Bordbar (2011), 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. En estos trabajos, se realiza un marcado semántico de los requisitos, usando ontologías de dominios como glosarios, con el fin de tener precargados y predefinidos la semántica y el sentido de cada palabra que será usada en el modelo. No obstante, el autor no realiza una desambiguación del sentido de las palabras ni evalúa la ambigüedad del sentido de las palabras en la especificación.

En Hinge, Ghose y Koliadis (2009), 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, para eliminar las ambigüedades presentes en el lenguaje y procesos de negocio obtenidos. Aun así, el autor no realiza la tarea de desambiguación de manera automática, ni relaciona los modelos de proceso de negocio con los requisitos de software.

Yang et al. (2010) presentan un método para identificar la ambigüedad estructural y sintáctica en los requisitos de software, utilizando una técnica de aprendizaje de máquina. Los autores entrenan directamente diversos corpus con el fin de inducir al algoritmo a que determine las ambigüedades en frases que hacen parte de un requisito de software. Sin embargo, no se menciona la ambigüedad semántica presente en los requisitos de software.

En la siguiente tabla se resumen los trabajos relacionados:

**Tabla 1. Resumen comparación de trabajos relacionados.**

| Enfoque | Usan lenguaje natural | Lenguaje natural controlado | Modelo de educción de requisitos usado | Ambigüedad atacada | Problema enfrentado |
|---|---|---|---|---|---|
| (Ambriola & Gervasi, 2006) | No | Sí | Modelos UML | Morfológica y sintáctica | Unicidad y duplicidad |
| Yang et al., (2010) | Sí | No | Ninguna | Sintáctica, mediante reglas | Duplicidad |
| (Nanduri & Rugaber, 1995) | Sí | No | Modelos orientados a objetos | Sintáctica | Ninguno |
| (Spreeuwenberg & Healy, 2010) Bajwa et al., (2011) | Sí | No | Modelos de procesos de negocio | Sintáctica y semántica | Inconsistencia |
| (Hinge, Ghose, & Koliadis, 2009) | Sí | No | Modelos de procesos de negocio | Semántica | Inconsistencia |

A continuación, se presentan algunas definiciones que serán usadas en el desarrollo del presente artículo.

# Definiciones

## 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.

Una palabra polisémica cumple una estructura computacional 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 lo 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 (Navigli & Vannella, 2013).

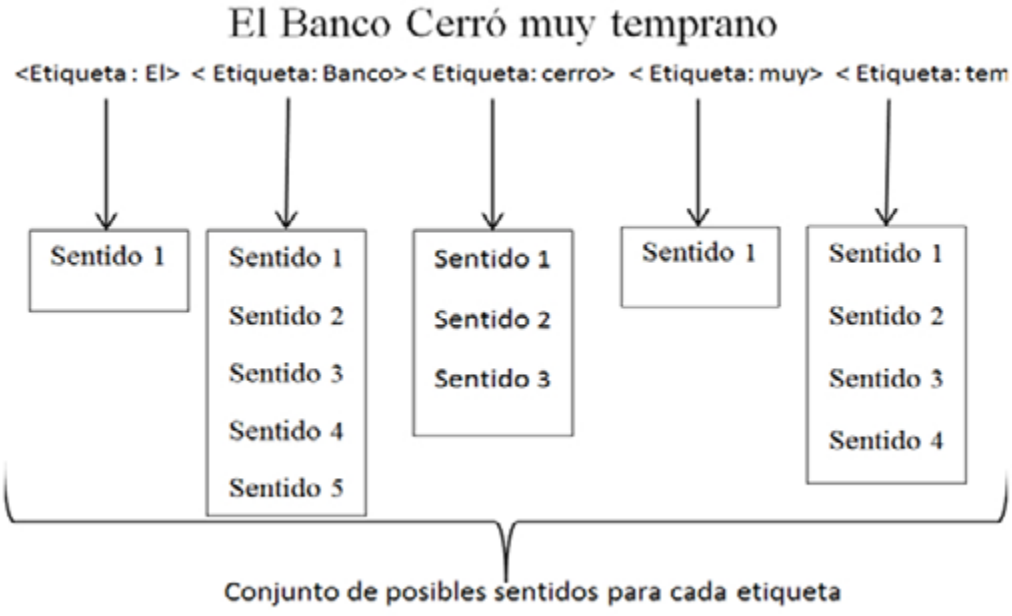


**Figura 2. Ejemplo estructura de una frase con palabras polisémicas**
**Fuente: Elaboración propia**

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 número único conocido y aceptado mundialmente por las agrupaciones académicas que trabajan en el área (Bajwa, Gonzalo, & Sekine, 2007). En la Figura 2 se presenta un ejemplo de una frase con palabreas polisémicas.

**Definición 1.** Los sentidos de las palabras se asocian solo a palabras funcionales. Formalmente, se define una palabra funcional como aquella que aporta contenido semántico a la frase a la que pertenece. Concretamente, bajo el marco de este artículo, las palabras funcionales son Verbos, Sustantivos, Adjetivos y Adverbios. No se consideran palabras funcionales, preposiciones ni determinantes.

# Desambiguación de los sentidos 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 $A(i) \subseteq SentidosD(Wi)$, 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 $|A(i)|=1$.

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 usa principalmente para métodos supervisados, en los cuales 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.

Los métodos de desambiguación de sentidos de las palabras se pueden clasificar en tres tipos, según Navigli (2009):

- 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), y así poder procesar y desambiguar textos nuevos basados en las reglas inferidas.
- 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.
- Basados en conocimiento: estos métodos pretenden explotar el conocimiento (tesauros, ontologías, diccionarios, etc.), con el fin de inferir el sentido de una palabra en un contexto. Hacen uso de WordNet, una base de datos léxica que agrupa palabras en conjuntos de sinónimos llamados synsets, proporcionando definiciones cortas y generales, y almacena las relaciones semánticas entre los conjuntos de sinónimos. Su propósito es doble: producir una combinación de diccionario y tesauro cuyo uso sea más intuitivo, y soportar análisis automático de texto y aplicaciones de inteligencia artificial.

# Errores generados en requisitos de software

A continuación, se presenta el conjunto de errores que pretende enfrentar el modelo propuesto en el presente artículo.

## Ambigüedad semántica  polisémica

Este error es el elemento que se tiene siempre presente ante un requisito de software, sin la definición de cada una de las palabras que componen el requisito de software. Por ejemplo, para un requisito de software "crear pilas", sin una definición asociada a cada palabra, se genera el error de ambigüedad semántica ya que la palabra "pilas" queda ambigua, al no saberse a qué sentido se está haciendo referencia.

## Inconsistencia

La inconsistencia se da cuando un requisito de software es contradictorio u opuesto a otro requisito de software (Hadad, 2011). Esto se determina una vez se soluciona el problema de la ambigüedad semántica y se tiene asociado un sentido para cada palabra que compone los requisitos de software.

Este error se identifica en el presente artículo mediante la siguiente regla: si existen sentidos opuestos en las palabras que componen los requisitos de software y si comparten al menos un sintagma nominal en común, entonces los requisitos involucrados son inconsistentes. En la Figura 3 se ilustra un ejemplo de este error.

| | Requisito de software 1 | | | Requisito de software 2 | | |
|---|---|---|---|---|---|---|
| | **Crear** | **Los** | **Carros** | **Destruir** | **Los** | **Carros** |
| **Categoría Gramatical** | Verbo | Artículo | Sustantivo | Verbo | Artículo | 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 | acabar con, causar la destrucción o perdición de | | Vehículo Automotor de 4 ruedas |

**Figura 3. Ejemplificación de problema de inconsistencia entre requisitos de software**

## Duplicidad

La duplicidad se da cuando dos requisitos de software significan lo mismo, así las palabras que los constituyen sean diferentes. Es decir si se encuentran dos requisitos de software cuyas palabras que componen su definición tienen definidos sentidos iguales, entonces se da duplicidad entre estos dos requisitos de software.  En la Figura 4 se presenta un ejemplo de duplicidad en requisitos de software.

| | Requisito de software 1 | | | Requisito de software 2 | | |
|---|---|---|---|---|---|---|
| **Palabra** | **Crear** | **Los** | **Carros** | **Fabricar** | **Los** | **automóviles** |
| **Categoría Gramatical** | Verbo | Artículo | Sustantivo | Verbo | Artículo | 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 |

**Figura 4. Ejemplo de duplicidad en requisitos de software**

## No unicidad

El problema de no unicidad se da cuando se presentan requisitos de software que tienen sentidos iguales, pero las palabras utilizadas para la definición son diferentes en los requisitos de software. En la Figura 5, se presenta un ejemplo.

| | Requisito de software 1 | | | | | | |
|---|---|---|---|---|---|---|---|
| Palabra | Crear | Los | Carros | | | | |
| Categoría Gramatical | Verbo | Artículo | 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 | | | | |
| | Requisito de software 2 | | | | | | |
| Palabra | Elaborar | Las | llantas | de | los | Automóviles | |
| Categoría Gramatical | Verbo | Artículo | Sustantivo | | Artículo | 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 | |
| | Requisito de software 3 | | | | | | |
| Palabra | Minimizar | Los | Daños | sufridos | en | las | ruedas |
| Categoría Gramatical | Verbo | Artículo | Sustantivo | verbo | Preposición | Artículo | 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 |
| | Requisito de software 4 | | | | | | |
| Palabra | Generar | Los | Planos | del | Automotor | | |
| Categoría Gramatical | Verbo | Artículo | Sustantivo | Artículo | 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 5. Ejemplo de requisitos de software no unificados**

El resto del artículo se encuentra organizado así: en la siguiente sección se presenta la metodología que expone el modelo propuesto de tres capas; posteriormente, se presentan los resultados, la discusión y, finalmente, las conclusiones.

# Metodología

El modelo propuesto recibe requisitos de software a modo de frase. Luego, se realiza un análisis para segmentación del requisito en palabras y un proceso morfosintáctico para el etiquetado de cada una de ellas. Después, se realiza una desambiguación usando la técnica de filtrado de coeficientes propuesta, de manera que sea posible determinar el mejor sentido para cada palabra que compone el requisito de software.

De esta forma, se entrega un requisito de software desambiguado (véase definición 3). Posteriormente, el modelo determina los errores de inconsistencia, duplicidad, no unicidad y ambigüedad encontrados en los requisitos de software. En la Figura 6 se presenta la estructura por capas del modelo propuesto.

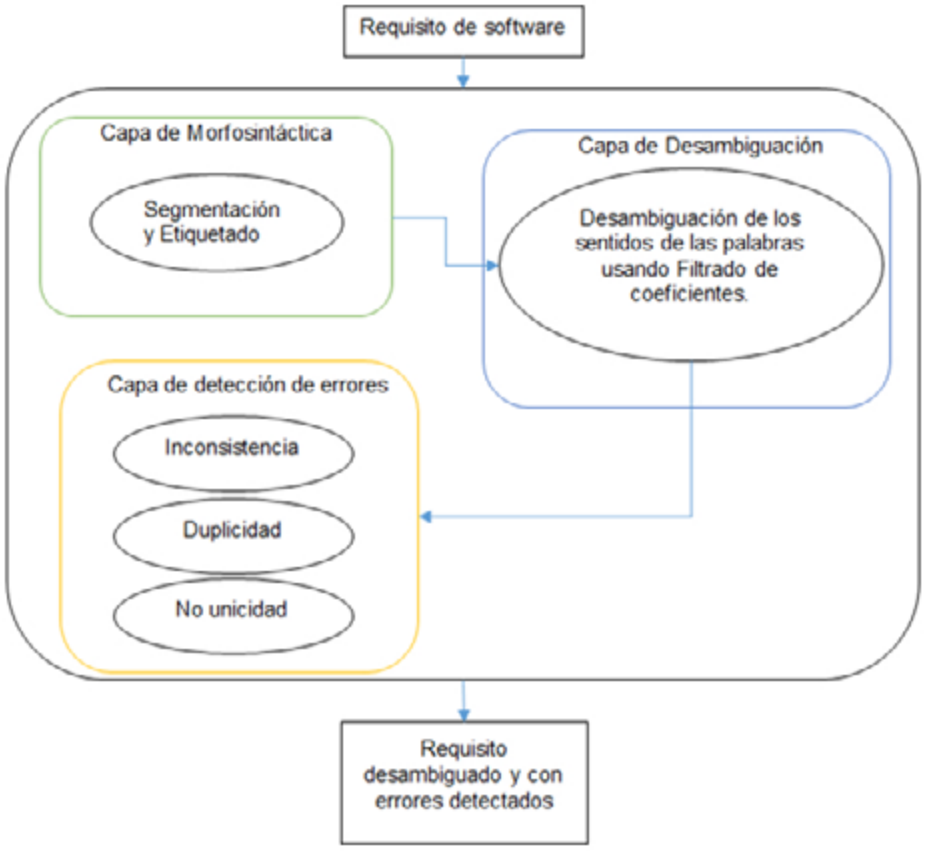


**Figura 6. Modelo automático de desambiguación semántica polisémica para la detección de errores en requisitos de software.**

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

## Proceso morfosintáctico

El modelo propuesto, una vez recibe el requisito de software a modo de frase, realiza el proceso morfosintáctico de segmentación de cada palabra que compone la frase del requisito. Esto convierte esta fuente del requisito en un arreglo de palabras. Finalmente, el algoritmo retorna el arreglo de palabras que componen la frase fuente del requisito 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 requisito se presenta en la Figura 7.

Luego de este proceso morfosintáctico, se procede a realizar la desambiguación de cada palabra mediante la técnica de filtrado de coeficientes. Antes de ello se presentan algunas definiciones.

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

**Figura 7. Algoritmo de segmentar y etiquetar.**

**Definición 2.** Una palabra desambiguada se define mediante la tupla:

<*palabra*, *etiqueta*, *lema*, *dependencia_sintáctica*, *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_sintáctica* 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 con 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 3.** Un requisito de software no ambiguo se define mediante la tupla.

<definición,  palabras_definicion>

- *definición* representa las palabras que componen el requisito de software.
- *palabras_definicion* es un arreglo que contiene las palabras desambiguadas (véase definición 2) que

componen el requisito de software.

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

## Filtrado de coeficientes de sentidos de palabras mediante la técnica de mínimos cuadrados para la desambiguación semántica polisémica en requisitos de software

La técnica de desambiguación usando filtrado de coeficientes fue propuesta por Guzmán-Luna y Arias (2014). Se compone de un conjunto de técnicas de desambiguación basadas en conocimiento (Adapted Lesk, Simplified Lesk, UKB, Most Frequent Sense), este conjunto de algoritmos arrojan una lista de sentidos con su respectivo *score* para cada una de las palabras en el arreglo *palabras_fuente*.

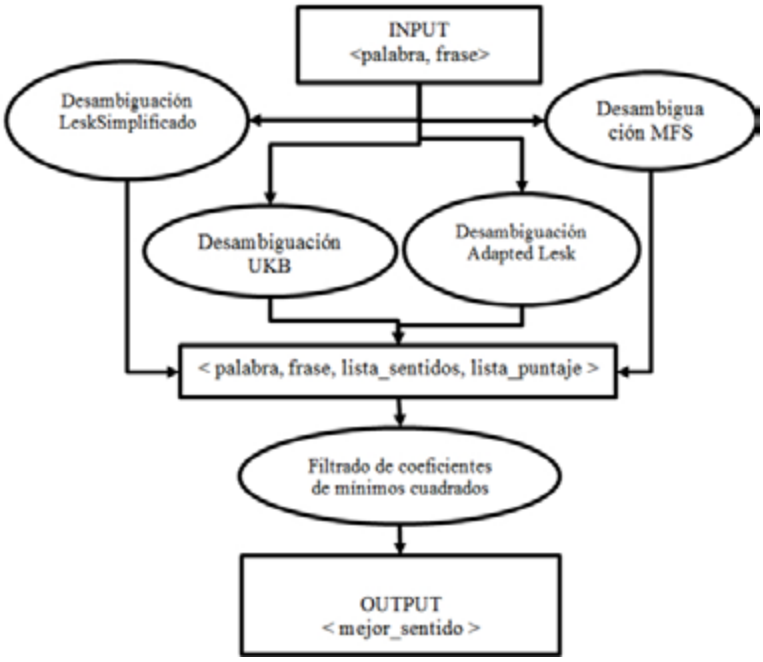


**Figura 8. Proceso por etapas del modelo de desambiguación.**
**Fuente: tomada de Guzmán-Luna y Arias (2014)**

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, y así se obtendrá un score definitivo que garantice un mejor sentido al unir las cuatro técnicas usadas. En la Figura 8 se detalla este proceso.

Un vez que se tiene el mejor sentido para cada palabra que compone la definición del requisito de software, se procede a la capa de detección de errores sobre los requisitos de software. Esta capa se expone a continuación.

# Detección de errores en requisitos de software

La capa de detección de errores analiza los requisitos de software. Una vez son desambiguados por la capa de desambiguación, su función consiste en aplicar un conjunto de reglas semánticas para detectar los requisitos inconsistentes, duplicados y no unificados. A continuación, se exponen las reglas formales para identificar cada uno de los errores.

## Detección de inconsistencias en requisitos de software

Para identificar inconsistencias entre requisitos de software se debe seguir la siguiente regla:

- Dos requisitos de software 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 requisito es contradictoria a otra de la siguiente manera:

$$\exists\, Palabra_i \;en\; Requisito1.definicion \wedge Palabra_j \;en\; Requisito2.definicion \;\therefore$$
$$\left(Palabra_i.sentido = Palabra_j.sentido\right) \vee$$
$$(Palabra_i.sentido \; esAntonimoDe \; sinonimos(Palabra_j.sentido) \rightarrow$$
$$Requisito1.inconsistente = true \wedge Requisito2.inconsistente = true$$

**Figura 9. Detección formal de inconsistencias en requisitos de software.**

## Identificación de duplicidad en requisitos de software

Para la identificación de duplicidades en requisitos de software se sigue la siguiente regla:

- Dos requisitos de software son duplicados si para cada sentido de las palabras que componen el requisito 1, se encuentran iguales sentidos en las palabras que componen el requisito 2.

La regla anterior se define formalmente como:

$$\forall\, Palabra_i \;en\; Requisito1.definicion \wedge Palabra_j \;en\; Requisito2.definicion \;\therefore$$
$$\left(Palabra_i.sentido = Palabra_j.sentido\right) \vee$$
$$\rightarrow Requisito1.duplicado = true \wedge Requisito2.duplicado = true$$

**Figura 10. Regla formal para la detección de requisitos de software duplicados.**

## Identificación de no unicidad en requisitos de software

Para la identificación de no unicidad en requisitos de software se sigue la siguiente regla:

Dos requisitos de software están no unificados si existen palabras diferentes en la definición de los requisitos de software, pero con un mismo sentido en ambos requisitos.

Esta regla se define formalmente como:

$$\exists\, Palabra_i \text{ en } Requisito1.definicion \wedge Palabra_j \text{ en } Requisito2.definicion \;\therefore$$
$$\left(Palabra_i.sentido = Palabra_j.sentido\right) \wedge Palabra_i.palabra = Palabra_j.palabra \rightarrow$$
$$Requisito1.unificado = false \wedge Requisito2.unificado = false$$

**Figura 11. Regla formal para la identificación de requisitos no unificados.**

# Resultados

El sistema prototipo que se implementó para probar el modelo propuesto 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*.

Los resultados que se muestran en esta sección 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.

Las métricas usadas para evaluar la presente propuesta se presentan en la Tabla 2.

**Tabla 2. Métricas usadas para la evaluación de la presente propuesta.**

| Métricas | Ecuación | Definición |
|---|---|---|
| Cobertura ($C_{wsd}$) | $C = \frac{U}{T}$ | La cobertura es el porcentaje de requisitos de software 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 requisitos de software en el conjunto de prueba) |
| Recuerdo ($R_{wsd}$) | $V = \frac{V}{T}$ | El recuerdo es el porcentaje de requisitos de software que han sido correctamente desambiguados dentro del conjunto de todos los requisitos de software 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 requisitos de software 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. |

| | | |
|---|---|---|
| | | modelo. |
| Precisión de la detección de inconsistencias | $PI_A = \frac{A_I}{A_T}$ | La precisión es el porcentaje de requisitos de software correctamente identificados como inconsistentes por el sistema $A_I$, versus el número total de requisitos de software 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 requisitos de software correctamente identificados como inconsistentes por el sistema $A_I$, versus el número total de requisitos de software inconsistentes hallados por el sistema $A_T$ más el número total de requisitos de software 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 requisitos de software correctamente identificados, como duplicados por el sistema $A_D$, versus el número total de requisitos de software 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 requisitos de software correctamente identificados, como duplicados por el sistema $A_D$, versus el número total de requisitos de software duplicado, hallados por el sistema $A_{TD}$, más el número total de requisitos de software duplicados no hallados por el sistema $A_{ND}$. |
| Precisión de la detección de no unicidades | $PU_A = \frac{A_U}{A_{TU}}$ | La precisión es el porcentaje de requisitos de software correctamente identificados como no unificados por el sistema $A_U$, versus el número total de requisitos de software no unificados hallados por el sistema $A_{TU}$. |
| Recuerdo de la detección de no unicidades | $RU_A = \frac{A_U}{A_{TU} + A_{NU}}$ | El recuerdo es el porcentaje de requisitos de software correctamente identificados como no unificados por el sistema $A_U$, versus el número total de requisitos de software inconsistentes hallados por el sistema $A_{TU}$, más el número total de requisitos de software inconsistentes no hallados por el sistema $A_{NU}$. |

Concretamente, se evaluó el modelo propuesto con siete casos de estudio, tres del estado del arte y cuatro propios (Ambulancia (Letier, 2001), Ascensor (A KAOS Tutorial, 2007), Pizzería (Zapata, Villegas & Arango, 2012), Carrefour, Colcerámicas, Aseguradora, Juego) bajo las métricas anteriormente descritas. En la Tabla 3 se presentan los resultados de la desambiguación del filtrado de coeficientes en requisitos de software, y en la Tabla 5 se presentan los resultados del proceso de detección de errores en requisitos de software. A continuación, se presentan algunas definiciones usadas para realizar las pruebas.

**Definición 4.** Un requisito 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 5.** Un requisito de software se considera correctamente desambiguado si todos los sentidos de las palabras funcionales que lo componen son correctos según el usuario.

**Definición 6.** Un requisito de software 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 7.** Un requisito de software se considera no desambiguado si al menos uno de los sentidos de las palabras funcionales que lo componen no posee un sentido asociado.

**Tabla 3. Resumen resultados capa de filtrado de coeficientes**

| | 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 |
| Requisitos | 23 | 26 | 20 | 16 | 20 | 11 | 12 | |

Para la evaluación del módulo de detección de errores se usaron los mismos casos de estudio. Específicamente, para la evaluación del módulo de detección de errores se evaluaron la precisión y el recuerdo en inconsistencias, duplicidades y no unicidades. En la Tabla 4 se presentan los valores de precisión y recuerdo para cada caso de estudio, tomando como referencia los valores presentados en la Tabla 3.

**Tabla 4. Valores obtenidos al aplicar las métricas sobre los resultados de cada caso de estudio.**

| Caso de estudio | Precisión en detección de inconsistencia | Recuerdo en detección de inconsistencia | Precisión en detección de duplicidades | Recuerdo en detección de duplicidades | Precisión en detección de no unicidades | Recuerdo en detección de no unicidades |
|---|---|---|---|---|---|---|
| | $PI_A$ | $RI_A$ | $PD_A$ | $RD_A$ | $PU_A$ | $RU_A$ |
| 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 |
| Colcerámica | 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 | 0,97619 | 0,735805 | 0,81847 | 0,77667 | 0,90476 | 0,86904 |

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 requisitos de software. En la siguiente tabla se presentan los datos tomados para evaluar el rendimiento.

**Tabla 5. Tiempo tomado por el modelo de desambiguación versus los requisitos de software a procesar.**

| Rendimiento por requisitos de software | |
|---|---|
| Tiempo de ejecución (segundos) | Requisitos de Software a procesar |
| 156 | 11 |
| 212 | 12 |
| 227 | 16 |
| 253 | 20 |
| 248 | 20 |
| 266 | 23 |
| 300 | 26 |

Finalmente, en la Figura 12 se demuestra que a mayor número de requisitos de software a desambiguar mayor es el tiempo de ejecución del método de desambiguación. Sin embargo, también se evidencia que los tiempos de ejecución son bastante altos, ya que para procesar 11 requisitos de software el sistema se gasta 2 minutos y 36 segundos, lo cual representa un tiempo de cómputo muy alto.

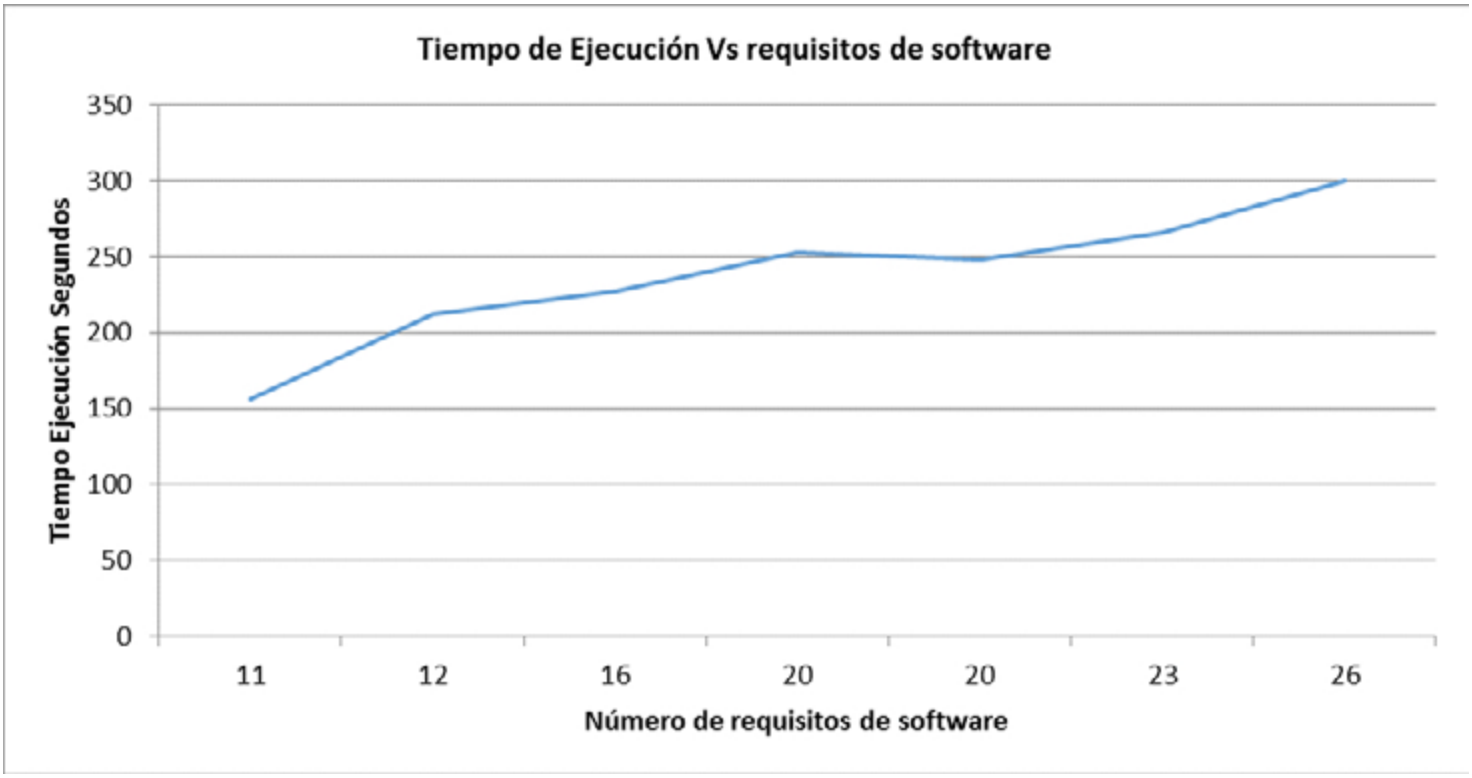


**Figura 12. Tiempo de ejecución para realizar las tareas de desambiguación variando el número de requisitos de software a procesar.**

# Discusión

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.

El recuerdo promedio se halló en 0.7967. Así, 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 requisitos de software introducidos.

La precisión promedio se halló en 0.9441, lo que significa que, aproximadamente, en el 95% de los casos en los que el sistema dio una respuesta, esta fue correctamente desambiguada.

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, realizó una desambiguación perfecta para el caso de estudio de la aseguradora, esto es, arrojó un 100% de efectividad. Para los demás casos de estudio se obtuvo una efectividad de la desambiguación entre 75% y 90%.

Para la capa de detección de errores de la Tabla 5, se puede decir que el sistema detecta inconsistencias con un 98% de precisión. De esta forma, de la totalidad de los requisitos de software que muestra como inconsistentes, el 98% son verdaderamente inconsistencias. Por su parte, el recuerdo de la detección de inconsistencias fue de 74%, lo que indica que, de las inconsistencias totales que posee una especificación, se detectaron el 74% de ellas de manera correcta.

En cuanto a la detección de duplicidades, se tiene que, de los requisitos de software 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.

Finalmente, en cuanto a la detección de no unicidades se tiene que, de los requisitos que identifica como no unificados, el 90% están realmente no unificados, mientras que, del total de requisitos de software, incluyendo detectados y no detectados, el sistema detectó como duplicados el 86%.

La métrica F1 promedio se halló en 0.8548, de manera que el sistema de desambiguación realiza la tarea global de desambiguación de los sentidos de las palabras en requisitos de software con una efectividad aproximada del 85%.

# Conclusiones

Se evaluaron las métricas cobertura, precisión y recuerdo para la desambiguación de los sentidos de las palabras; para la detección de errores, se evaluó precisión y recuerdo. Para tal fin, se realizaron los experimentos sobre tres casos de estudio tomados 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:

a. El módulo de desambiguación realiza las tareas de desambiguación encomendadas con una efectividad del 86% (dado por la métrica F1), siendo este un porcentaje relativamente alto y bueno según Navigli (2009).

b. El modelo 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 unicidades 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:

a. Los tiempos de procesamiento del módulo de desambiguación son relativamente altos.

b. 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).

c. 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.

# Referencias


A KAOS Tutorial, C. (2007). Recuperado el 03 agosto de 2012 de http://www. objectiver. com/fileadmin/downlo ad/documents. *KaosTutorial. pdf*.

Ambriola, V., & Gervasi, V. (2006). On the Systematic Analysis of Natural Language Requirements with CIRCE. *Automated Software Engineering*, *13*(1), 107-167. http://doi.org/10.1007/s10515-006-5468-2

Artiles, J., Gonzalo, J., & Sekine, S. (2007). The semeval-2007 weps evaluation: Establishing a benchmark for the web people search task. En *Proceedings of the 4th International Workshop on Semantic Evaluations* (pp. 64–69). Recuperado de http://dl.acm.org/citation.cfm?id=1621486

Bajwa, I. S., Lee, M. G., & Bordbar, B. (2011). SBVR Business Rules Generation from Natural Language Specification. En *AAAI*

*Spring Symposium: AI for Business Agility*. Recuperado de http://www.aaai.org/ocs/index.php/SSS/SSS11/paper/viewPDFInterstitial/2378/2918

Booch, G., Rumbaugh, J., Jacobson, I., Martínez, J. S., & Molina, J. J. G. (1999). *El lenguaje unificado de modelado* (Vol. 1). Addison-Wesley. Recuperado de http://files.cecap49.webnode.es/200000016-6cd116dccb/3E-UML.pdf

Guzmán-Luna, J. A., & Arias, S. A. G. (2014). Filtering Word Senses Using the Least Squares Technique for Word Sense Disambiguation. *International Journal of Engineering and Technology*, *6*(6), 2617-2628.

Hadad, G. D. (2011). Panorama de la Ingeniería de Requisitos: sus fundamentos y avances. Recuperado de http://184.168.109.199:8080/xmlui/handle/123456789/3056

Hinge, K., Ghose, A., & Koliadis, G. (2009). Process seer: A tool for semantic effect annotation of business process models. En *Enterprise Distributed Object Computing Conference, 2009. EDOC'09. IEEE International* (pp. 54–63). IEEE. Recuperado de http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5277722

Letier, E. (2001). *Reasoning about agents in goal-oriented requirements engineering*. PhD thesis, Université catholique de Louvain. Recuperado de http://www0.cs.ucl.ac.uk/staff/e.letier/publications/letier-thesis.pdf

Nanduri, S., & Rugaber, S. (1995). Requirements validation via automated natural language parsing. En *System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on* (Vol. 3, pp. 362–368). IEEE. Recuperado de http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=375617

Navigli, R. (2009). Word sense disambiguation: A survey. *ACM Computing Surveys (CSUR)*, *41*(2), 10.

Navigli, R., & Vannella, D. (2013). 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*.

Nerlich, B., & Domínguez, P. J. C. (1999). Cómo hacer cosas con palabras polisémicas: El uso de la ambigüedad en el lenguaje ordinario. *Contrastes: revista internacional de filosofía*, (4), 77–96.

Osborne, M., & MacNish, C. K. (1996). Processing natural language software requirement specifications. En *Requirements Engineering, 1996., Proceedings of the Second International Conference on* (pp. 229–236). IEEE. Recuperado de http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=491451

Ruiz, J. J. M. Z. (2004). ¿Por qué fracasan los proyectos de software? Un Enfoque Organizacional. En *Congreso Nacional de Software Libre*. Recuperado de http://ftp.rabade.net/pub/cdrom_consol_2004/consol/2004/comas/general/material/63/jzavalar_por-que-fallan-los-proyectos-de-software-CONSOL2004.pdf

Spreeuwenberg, S., & Healy, K. A. (2010). SBVR's approach to controlled natural language. En *Controlled Natural Language* (pp. 155–169). Springer. Recuperado de http://link.springer.com/chapter/10.1007/978-3-642-14418-9_10

Videira, C., & Da Silva, A. R. (2005). Patterns and metamodel for a natural-language-based requirements specification language. En *CAiSE Short Paper Proceedings*. Citeseer. Recuperado de http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.89.8443&rep=rep1&type=pdf

Videira, C., Ferreira, D., & Da Silva, A. R. (2006). A linguistic patterns approach for requirements specification. En *Software Engineering and Advanced Applications, 2006. SEAA'06. 32nd EUROMICRO Conference on* (pp. 302–309). IEEE. Recuperado de http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1690153

Yang, H., Willis, A., De Roeck, A., & Nuseibeh, B. (2010). Automatic detection of nocuous coordination ambiguities in natural language requirements. En *Proceedings of the IEEE/ACM international conference on Automated software engineering* (pp. 53–62). ACM. Recuperado de http://dl.acm.org/citation.cfm?id=1859007

Zapata, C. M., Villegas, S. M., & Arango, F. (2012). Reglas de consistencia entre modelos de requisitos de UN-Metodo. *Revista Universidad Eafit*, *42*(141), 40–59.