UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 1

"Módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión"

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor:

Arlene Padrón Samá

Tutores:

Ing. Serguey González Garay

Ing. Eric Bárbaro Utrera Sust

Ing. Alejandro Jesús Mariño Molerio

La Habana, Junio 2017


"La informática se convertirá en una poderosa fuerza científica, económica y política para Cuba"
Fidel Castro Ruz

Declaración de autoría

Declaro por este medio que yo Arlene Padrón Samá, con carné de identidad 93112706816 soy la autora principal del trabajo de diploma titulado Módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión y autorizo a la Universidad de Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Para que así conste se firma la presente declaración jurada da autoría en La Habana a los días ___ del mes de_____ del año_____.

Arlene Padrón Samá

_____

Firma del Autor

Ing. Serguey González Garay

_____

Firma del Tutor

Ing. Eric Bárbaro Utrera Sust

_____

Firma del Tutor

Ing. Alejandro Jesús Mariño Molerio

_____

Firma del Tutor

Agradecimientos

Muchas han sido las personas que, a lo largo de estos años, han contribuido a que hoy me encuentre aquí, este es el momento para agradecer a cada uno de ellos:

***A mis padres** por siempre brindarme su amor, apoyo, preocupación y sabiduría. Gracias por darme todo en la vida y entre los dos hacerme la persona que soy hoy.*

***A mi hermana** por apoyarme y estar a mi lado en todo momento dándome ánimos para seguir adelante.*

***A mi familia** por siempre confiar en mí y apoyarme en todas mis decisiones.*

***A mi novio** por estar ahí en todo momento ofreciéndome su amor, cariño y apoyo.*

***A la UCI** por las oportunidades ofrecidas y por haber sido mi segundo hogar durante estos años.*

***A mis compañeros de aula** por los buenos momentos que pasamos juntos y los recuerdos que quedaron grabados para siempre, por todo el apoyo y la ayuda brindada.*

***A mi familia de la FEU** por cada experiencia que compartimos juntos, por todo el cariño y la incondicionalidad que me demostraron siempre.*

***A mis tutores** por haber soportado todas mis malacrianzas y así todo confiar en mí.*

***A la dirección de la facultad** quien me ayudó y aconsejó en todo momento.*

A todos les agradezco, por haberme dado la oportunidad de graduarme.

Dedicatoria

A la revolución por haberme dado el privilegio de estudiar en la Universidad de las Ciencias Informáticas

A mis padres y mi hermana por su apoyo incondicional

A mi abuelo Santiago Padrón que, aunque ya no está con nosotros se sentiría orgulloso de ver a su nieta convertida en ingeniera.

A mi abuela Paula Duporté por su entera confianza.

A toda mi familia y amigos.

Resumen

Dado que el motor de búsqueda cubano Orión como sistema de recuperación de información (SRI) no contempla el nivel semántico que es uno de los niveles de análisis que posee la lingüística, surge la necesidad de desarrollar un módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión, con el objetivo de poder interpretar el significado semántico de la información y obtener mejores resultados de búsquedas. Este permitirá realizar técnicas de procesamiento de lenguaje natural a los documentos indexados mejorando así los resultados de las indagaciones realizadas por los usuarios y fortalecer el funcionamiento del mismo. Para la ejecución de este módulo se empleó como herramienta de almacenamiento el servidor indexador Solr. Para el diseño, implementación y documentación del módulo la metodología de desarrollo AUP-UCI, para el modelado de la solución se utilizó el Lenguaje Unificado de Modelado (UML) y la herramienta Visual Paradigm. Se empleó también el Eclipse luna como entorno de desarrollo integrado y como servidor web Apache Tomcat. El leguaje de desarrollo utilizado fue Java. Este módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión permite realizar técnicas de procesamiento de lenguaje natural a través de análisis semántico a cada una de las oraciones de los documentos contribuyendo a un mejor resultado de búsqueda.

Palabras clave: búsqueda, lenguaje natural, motor, semántico, tripleta, web.

Índice

Índice de Tablas

Índice de Figuras

Introducción

En la actualidad el desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC) permite ampliar las posibilidades del progreso de la sociedad y la adquisición de conocimientos útiles, logrando en ocasiones satisfacer las necesidades de los profesionales. El impacto causado por las TIC revolucionó la manera de hacer negocios, la comunicación entre las personas y la interacción entre las empresas. Como parte de estas tecnologías, se encuentra Internet, la cual supuso una revolución sin precedentes en el mundo de la informática y la comunicación. Al cursar el tiempo, este universo digital está evolucionando rápidamente con el incremento de contenido, por lo que se necesitan estrategias para lograr un mejor aprovechamiento de la información que se encuentra disponible en Internet. Para poder facilitar la búsqueda de dicha información contenida en la variedad de sitios existentes, surgieron los buscadores web, que permiten mostrarles a los usuarios los resultados de la indagación de una forma rápida y efectiva (Estigarribia, 2015).

La mayoría de recursos disponibles en la Web están estructurados en base al formato de hipertexto conocido como *Hypertext Markup Language* (HTML). Este es un lenguaje de marcado desarrollado por el creador de la web Tim Berners-Lee y adoptado como estándar por la World Wide Web Consortium (W3C). Los documentos estructurados con este formato (páginas web), pueden ser comprendidos fácilmente por los humanos y los navegadores web, sin embargo, los navegadores y sistemas no entienden que significa lo que está escrito en el contenido por lo que, en ocasiones, resulta imposible extraer su valor semántico automáticamente. Por otro lado, la información en la Web se encuentra dispersa y no existe relación explícita entre los diferentes recursos, provocando ambigüedad en la información (Dadzie, 2011).

Los problemas de formato e integración provocan que la recuperación de información se vea afectada, lo que se evidencia en los motores de búsquedas más utilizados en la actualidad. Los resultados ofrecidos por motores de búsqueda como Google y Yahoo presentan limitaciones y resultan imprecisos (Prasad, 2012). Además, en muchos casos, no satisfacen las necesidades de búsqueda de los usuarios al responder consultas basadas en palabras clave, no siendo capaces de recuperar la información a partir de consultas expresadas en lenguaje natural.

La lingüística computacional es un área de investigación que se basa en el procesamiento de lenguajes

naturales. Se considera una rama de la lingüística[1] que mezcla el análisis sintáctico y morfológico[2] con herramientas informáticas para el estudio de la lengua humana hablada y escrita (Illescas, 2010). Desde el ámbito informático es considerada una subdisciplina de la inteligencia artificial que intenta entender mejor el lenguaje humano, a través de algoritmos de análisis sintáctico y matemáticos basados en probabilidad. El principal problema que presenta la lengua humana es la denominada infinitud discreta por (Chomsky, 1998). La infinitud discreta está muy ligada a la recursión, pues es un elemento necesario para cualquier teoría humana que trata de formalizar oraciones a partir de un número limitado de elementos constituyentes de un lenguaje. Desde hace varias décadas la lingüística computacional está en pleno desarrollo y parece un campo muy interesante donde investigar, pero a su vez presenta un problema esencial muy difícil de resolver, pues la infinitud discreta no es fácil de abordar con las nuevas tecnologías (Sidorov, 2001).

El proceso de análisis de cadenas de símbolos, escritos en lenguaje natural o lenguaje máquina, empleando las reglas de una gramática formal es denominado Análisis sintáctico o *parsing*, el cual consiste en entender el significado riguroso de la frase. Un analizador sintáctico es un software basado en algoritmos de *parsing*, algoritmos probabilísticos y una combinación de ambos, que convierte un texto de entrada en otras estructuras. En otras palabras, es un analizador sintáctico de una lengua natural (Colmenarejo, 2016).

Actualmente los analizadores sintácticos están más avanzados que cuando aparecieron por primera vez, pues el avance de los ordenadores y de la computación en general ha dejado mejorar los rendimientos de los algoritmos y por tanto el análisis se realiza más rápido. También, con el desarrollo de la computación, aparecen nuevas formas de procesar datos permitiendo crear algoritmos combinatorios, así pues, se puede decir que los analizadores sintácticos seguirán evolucionando con la informática manteniéndose actualizados y permitiendo nuevas formas de análisis sintáctico del lenguaje (Periñan-Pascual, 2012).

En la lingüística existen varios niveles de análisis, aunque no todos los lingüistas están de acuerdo, la mayoría coinciden en al menos los siguientes: fonético[3], fonológico[4], morfológico, sintáctico y semántico. Si bien estos niveles están conectados entre ellos no es necesario que aparezcan todos para hacer un análisis lingüístico correcto (Abela, 2002). En esta investigación se pondrán en práctica los que se explican a continuación:

[1] Lingüística: Es el estudio científico tanto de la estructura de las lenguas naturales.
[2] Morfológico: Morfología, parte de la lingüística que estudia las reglas que rigen la flexión, la composición y la derivación de las palabras.
[3] Fonético: Es el estudio de los sonidos físicos del discurso humano.
[4] Fonológico: Describe el modo en que los sonidos funcionan en una lengua en particular o en las lenguas en general.

- ✓ Sintaxis: es la rama de la lingüística dedicada al análisis sintáctico. Estudia las reglas, principios y normas que rigen la combinatoria de las palabras o secuencias de palabras que forman sintagmas y oraciones.
- ✓ Semántica: Estudia el significado de las expresiones dentro de las oraciones. Se refiere a los aspectos del significado, sentido o interpretación de los signos lingüísticos.

Las oraciones luego de analizarlas sintáctica y semánticamente cambian su estructura y se convierten en un nuevo modelo denominado tripletas (Ceran, 2012). La tripleta está formada por tres componentes: sujeto - relación - predicado, que se puede representar como dos nodos, sujeto y objeto, unidos por una relación. Además, se debe considerar que un sujeto es un recurso, la forma verbal es la relación con el recurso y el predicado es otro recurso o el valor de la propiedad.

En la Universidad de las Ciencias Informáticas (UCI) actualmente se encuentra en desarrollo el motor de búsqueda Orión, cuyo principal objetivo es brindar servicios para la recuperación de información. El buscador es accesible para usuarios de todas las regiones del país y los resultados de las búsquedas mediante el mismo, serán páginas web comprendidas bajo el dominio .cu, por lo que la gran mayoría se encuentra en idioma español. De los niveles de análisis que posee la lingüística, el buscador cubano solo pone en práctica el fonético, fonológico, morfológico y sintáctico, no tiene en cuenta el nivel semántico.
Esto se evidencia cuando un usuario realiza una consulta, debido a que Orión devuelve todos los documentos que contienen términos iguales a los solicitados, sin embargo, el buscador no es capaz de mostrarle al usuario los documentos que contienen exactamente la misma estructura de la consulta realizada, lo que contribuye a que los resultados de búsquedas obtenidos no sean los más indicados. Otras de las limitaciones con que cuenta Orión es que no aprovecha las propiedades de la semántica al no poder interpretar el significado de la información que indexa en su base de datos. Orión reconoce la cantidad de términos relevantes que tienen sus documentos, pero no es capaz de enlazar los términos que responden a un mismo dominio, lo que impide obtener mejores resultados en las búsquedas.

Sobre la base de los elementos expuestos anteriormente se formula el siguiente **problema de la investigación**: ¿Cómo contribuir al procesamiento semántico de los documentos indexados por el motor de búsqueda Orión?

Para la realización de la investigación se define como **objeto de estudio**: El proceso de extracción de

tripletas de documentos; enfocado en el proceso de extracción de tripletas de documentos indexados por el motor de búsqueda Orión como **campo de acción.**

Para dar solución al problema planteado, se define como **objetivo general**: Desarrollar un módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión utilizando técnicas de procesamiento de lenguaje natural para contribuir al procesamiento semántico de los documentos indexados por el motor de búsqueda Orión.

Con el propósito de complementar gradualmente el objetivo general antes mencionado, el mismo se ha desglosado en los siguientes **objetivos específicos**:

- ✓ Caracterizar los fundamentos teóricos relacionados con el módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión.
- ✓ Definir las tecnologías, las herramientas y la metodología para la implementación del módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión.
- ✓ Diseñar el módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión.
- ✓ Implementar el módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión.
- ✓ Validar la solución propuesta.

Teniendo en cuenta lo anterior, se plantea la siguiente **hipótesis de investigación:** El desarrollo de un módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión utilizando técnicas de procesamiento de lenguaje natural contribuirá al procesamiento semántico de los documentos indexados por el motor de búsqueda Orión. Teniendo en cuenta la hipótesis planteada, se define como **variable independiente:** el módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión. Esta consiste en una aplicación informática que a partir de la extracción de tripletas de los documentos permitirá realizar un análisis más profundo y exacto de estos mediante la información extraída. Como **variable dependiente** se especifica: contribuir al procesamiento semántico de los

documentos indexados por el motor de búsqueda Orión.

Tabla 1 Operacionalización de las variables

Variables	Descripción	Dimensiones	Indicadores	Unidad de Medida
Variable independiente	Módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión.	Extracción de conocimientos	Extracción de tripletas de documentos indexados por el motor de búsqueda Orión.	• Totalmente de acuerdo • De acuerdo • Ni de acuerdo ni en desacuerdo • En desacuerdo • Totalmente en Desacuerdo
Variable dependiente	Contribuir al procesamiento semántico de los documentos indexados por el motor de búsqueda Orión.	Calidad de la tripleta	Criterio de expertos	

Para dar cumplimiento a los objetivos específicos se definieron las siguientes **tareas de investigación**:

- ✓ Realización de un estudio sobre las tendencias de las herramientas que realizan una extracción de tripletas de documentos.
- ✓ Selección de las tecnologías, herramientas y estándares que se necesitan para implementar la propuesta de solución.
- ✓ Selección de la metodología de desarrollo de software.
- ✓ Elaboración de los artefactos requeridos por la metodología de desarrollo seleccionada.
- ✓ Implementación de la propuesta de solución.
- ✓ Documentación de las pruebas realizadas.

Para el cumplimiento de los objetivos propuestos se utilizarán una serie de métodos teóricos:

- ✓ **Analítico-Sintético** Se aplicará en el análisis de las herramientas y tecnologías de procesamiento de lenguaje natural posibilitando identificar aquellas que puedan ser aplicadas en el desarrollo del trabajo, así como para el análisis de la información estudiada.

- ✓ **Histórico-Lógico**: Se utilizará en el estudio de la evolución y desarrollo de la extracción de tripletas de documentos, lo que permitirá que se pueda conocer las tendencias actuales en el desarrollo de estos sistemas.

- ✓ **Inducción-Deducción**: Se usará en el análisis de las características del comportamiento de las tendencias de las herramientas de extracción de tripletas de documentos.

El Trabajo de Diploma está estructurado en tres capítulos como se describe a continuación:

CAPÍTULO 1: **Fundamentación teórica asociada al procesamiento de extracción de tripletas de documentos indexados por le motor de búsqueda Orión**: Se expondrán un conjunto de conceptos y criterios fundamentales asociados al objeto de estudio de la investigación. Además, se estudiarán las principales herramientas para el procesamiento de textos en lenguaje natural con la finalidad de brindar una solución a la problemática planteada. Finalmente, se expondrán las distintas tecnologías a utilizar en el desarrollo de la herramienta, así como la metodología de desarrollo de software a utilizar.

CAPÍTULO 2: Análisis y diseño del módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión: Se exponen las características del módulo a desarrollar, incluyendo los requisitos funcionales y no funcionales, patrones de diseño y arquitectura utilizados; además algunos de los artefactos que requiere la metodología de desarrollo utilizada.

CAPÍTULO 3: Implementación y validación del módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión: Se exponen aspectos asociados con la implementación de la solución informática, así como los componentes que la integran. Además, se presentan los diseños de casos de prueba a utilizar en la validación del módulo y se analizan los resultados de las pruebas realizadas que permiten evaluar la calidad de la propuesta de solución.

Capítulo 1: Fundamentación teórica asociada al proceso de extracción de tripletas de documentos indexados por el motor de búsqueda Orión.

1.1 Introducción

En este capítulo, se presentan los fundamentos básicos para la extracción de tripletas de documentos, destacando conceptos y definiciones fundamentales asociados a la investigación. Además, se realiza una breve descripción de las herramientas, tecnologías, lenguajes de programación, marcos de trabajo y otros elementos utilizados en el desarrollo de la propuesta de solución.

1.2 Fundamentos teóricos asociados al dominio del problema

A lo largo de la existencia del hombre, la información ha adquirido diversos matices en diferentes campos del saber y en distintas actividades sociales. La información como punto de partida y parte de cada proceso universal constituye hoy la base del desarrollo de las esferas de una sociedad. Según la Real Academia Española la información es: "**Comunicación o adquisición de conocimientos que permiten ampliar o precisar los que se posee sobre una materia determinada**" (Real Academia Española, 2014).

Para un correcto uso de la información esta debe gestionarse de manera efectiva. La gestión de la información es un proceso que incluye operaciones como extracción, manipulación, tratamiento, depuración, conservación, acceso y/o colaboración de la información adquirida por una organización a través de diferentes fuentes. Esta información gestiona el acceso y los derechos de los usuarios sobre la misma (Batanero, 2013). A continuación, se muestran los principales conceptos que se abordan en esta investigación.

1.2.1 Recuperación de información

A lo largo de los años el concepto de Recuperación de Información (RI), ha sido analizado por varios autores, entre ellos ((Lado, 2010); (Baeza-Yates, 1999)), plantean que "la Recuperación de Información trata con la representación, el almacenamiento, la organización y el acceso a ítems de información". Sin embargo, un año después, (Pérez, 2000) indica que "la Recuperación de Información es traer documentos relevantes desde un gran archivo en respuesta a una pregunta formulada".

Aunque se evidencia con claridad que existe determinada variación en las definiciones respecto a la evolución del término RI en el tiempo, se puede constatar que existe gran similitud con la definición que plantea años más tarde (Bordignon, 2008) en su libro Introducción a la Recuperación de Información: "la RI consiste en encontrar documentos relevantes que satisfagan la necesidad de información de un usuario,

expresada en un determinado lenguaje de consulta".

De acuerdo con la definición anterior, la autora del presente trabajo considera que la Recuperación de Información consiste en la presentación de información relevante desde una colección de documentos, a un usuario que hace una petición sobre la base del lenguaje natural, mediante la utilización de métodos, técnicas y sistemas de recuperación de información que logren satisfacer sus necesidades de información.

1.2.2 Sistema de recuperación de Información

Los Sistemas de Recuperación de Información (SRI) son los encargados de buscar información dentro de un grupo de documentos que han sido indexados previamente. Estos sistemas no se limitan a buscar dentro de documentos solamente, sino que pueden ampliar su dominio de búsqueda a bases de datos y también a Repositorios Institucionales, además permiten localizar y procesar cualquier contenido existente, tales como textos, imágenes, videos y archivos de sonido (Martínez,2004).

Los SRI "deben de alguna manera interpretar el contenido de la información dentro de una colección de documentos y establecer con ellos, un orden de acuerdo al grado de relevancia que estos posean para las consultas de los usuarios" (Baeza–Yates, 2009).

En la presente investigación, el sistema de recuperación de información cuenta con una arquitectura como se muestra en la Figura 1. Para que los SRI entiendan la información de los documentos que se almacenan en su base de datos es necesario trabajar con web semántica.

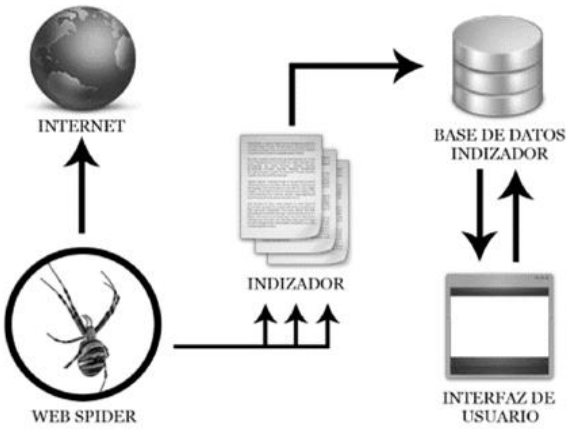

Figura 1 Arquitectura de un buscador (Kuna, 2016)

1.2.5 Web semántica

La web semántica propone superar las limitaciones de la web actual mediante la introducción de descripciones explícitas del significado, la estructura interna y la estructura global de los contenidos y servicios disponibles en la WWW. Frente a la semántica implícita, el crecimiento caótico de recursos y la ausencia de una organización clara de la web actual, la web semántica aboga por clasificar, dotar de estructura y anotar los recursos con semántica explícita procesable por máquinas. La Figura 2 ilustra esta propuesta. En la web semántica cada nodo (recurso) tiene un tipo (profesor, tienda, pintor, libro) y los arcos representan relaciones explícitamente diferenciadas (pintor – obra, profesor – departamento, libro – editorial) (Bernes-Lee, 2001).

La web semántica mantiene los principios que han hecho un éxito de la web actual, como son los principios de descentralización, compartición, compatibilidad, máxima facilidad de acceso y contribución, o la apertura al crecimiento y uso no previstos de antemano. En este contexto un problema clave es alcanzar un entendimiento entre las partes que han de intervenir en la construcción y explotación de la web: usuarios, desarrolladores y programas de muy diverso perfil. La web semántica rescata la noción de ontología del campo de la Inteligencia Artificial como vehículo para cumplir este objetivo (Hernández, 2013).

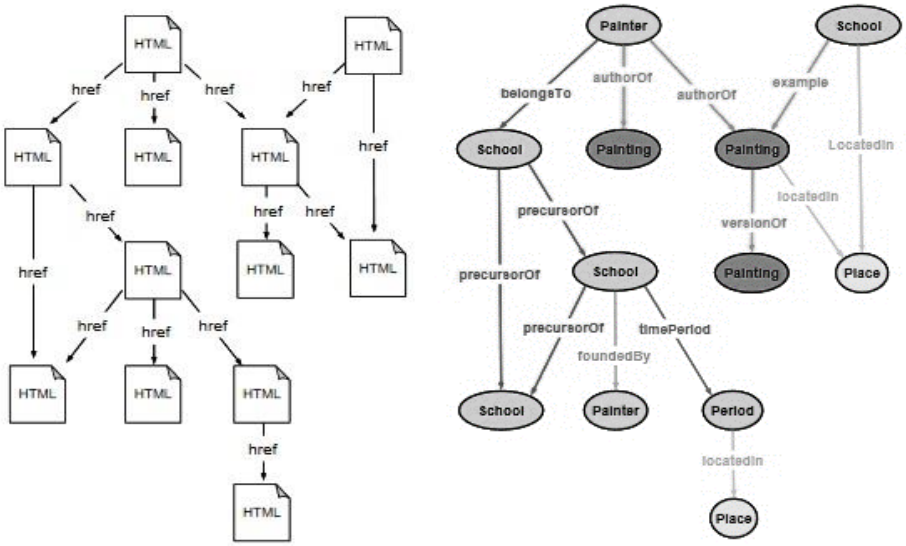

Figura 2 Comparación entre la web actual y la web semántica (Bernes-Lee, 2001).

1.2.6 Tripletas

Para lograr una mejor organización de los documentos a los que se les realiza el análisis semántico se utilizan las tripletas. Según la Real Academia española una tripleta es un conjunto de tres personas, animales o cosas.

En la rama de la lingüística una tripleta es la manera de representar la información de una oración de texto (Ceran, 2012). En este caso la estructura de la tripleta es sujeto-relación-predicado como se muestra en la Figura 3, donde la relación representa la forma verbal y el sujeto y el predicado pertenecen a una oración determinada. Para la realización del análisis sintáctico de un documento, donde se incluye la extracción de tripletas, surgieron los analizadores sintácticos o *Parser*.

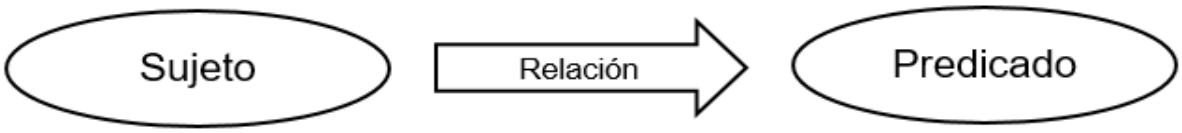

Figura 3 Estructura de una tripleta (Ceran, 2012).

1.2.7 Analizador Sintáctico (*Parser*)

El analizador es la fase que se encarga de chequear el texto de entrada en base a una gramática dada. Y en caso de que el programa de entrada sea válido, suministra el árbol sintáctico que lo reconoce. En teoría,

se supone que la salida del analizador sintáctico es alguna representación del árbol sintáctico que reconoce la secuencia de tokens suministrada por el analizador léxico (Martínez, 2008).
En la práctica, el analizador sintáctico también realiza las siguientes acciones:

- ✓ Acceder a la tabla de símbolos (para hacer parte del trabajo del analizador semántico).
- ✓ Chequeo de tipos (del analizador semántico).
- ✓ Generar código intermedio.
- ✓ Generar errores cuando se producen.

En definitiva, realiza casi todas las operaciones de la compilación. Este método de trabajo da lugar a los métodos de compilación dirigidos por sintaxis.

1.3 Sistemas Homólogos

A nivel mundial se han desarrollado disímiles técnicas que permiten el estudio del lenguaje natural tanto de textos y documentos como de páginas web. Aunque no se puede afirmar que han alcanzado la perfección, realmente algunas herramientas analizan con una extraordinaria exactitud (Rusu, 2007), sobre todo si analizan frases en inglés, sin embargo, cuando se trata de analizar en idioma español, es donde surgen ambigüedades. En la actualidad existen diversos razonadores disponibles públicamente que permiten analizar automáticamente tanto textos, como documentos o páginas web, incluso, algunos de ellos permiten la integración de su herramienta de razonamiento a otros sitios web.

1.3.1 Linguakit

Esta web multilingüe, que integra, entre otras herramientas lingüísticas, un resumidor, un analizador de sentimiento o un extractor de las palabras clave que dan sentido a un texto, va dirigida a un amplio abanico de usuarios que hacen de la lengua un uso profesional, educativo o general (Masrani, 2015). Para ver la imagen del sistema ir al Anexo 1.
Esta plataforma presenta sus módulos lingüísticos organizados en cuatro apartados orientativos: un primero que atiende a aspectos más genéricos del lenguaje con módulos como el conjugador; un segundo, para un perfil de usuario más ligado al ámbito educativo, con módulos como el etiquetador morfosintáctico o el analizador sintáctico; un tercer apartado pensado para profesionales de la comunicación y marketing como el analizador de sentimiento o el extractor de palabras clave; y, por último, un apartado experimental donde Linguakit presenta las nuevas herramientas en proyecto (Gamallo, 2015).

Entre las funcionalidades que realiza este sistema se encuentran:

- ✓ Análisis completo
- ✓ Resumidor
- ✓ Conjugador verbal
- ✓ Analizador sintáctico
- ✓ Extractor de palabras clave
- ✓ Extractor de tripletas

Esta última funcionalidad abre nuevas posibilidades en el campo del análisis textual en Linguakit porque mejora la estructuración del contenido.

Para cada texto, el extractor de tripletas extrae automáticamente aquello que se dice de cada sujeto enunciado porque irá, frase por frase, extrayendo las tripletas semánticas: Sujeto-Relación-Objeto. Detecta las relaciones existentes entre elementos: de qué se está hablando o qué se dice de los diferentes sujetos de un texto.

Este sistema es el único que entre sus funcionalidades se encuentra la extracción de tripletas, pero cuenta con varias limitaciones, estas son:

- ✓ El extractor de tripleta tiene un límite de 5000 caracteres a introducir, lo que impide analizar documentos con grandes volúmenes de texto.
- ✓ El algoritmo que posee esta herramienta no reconoce una oración compuesta o una oración donde el sujeto esté omitido y por tanto devuelve el campo de tripletas vacío)
- ✓ El acceso a la herramienta se dificulta por el limitado acceso a internet que tiene Cuba debido al embargo económico.
- ✓ El sistema permite realizar solo 10 consultas y si el usuario necesita realizar más, se debe autenticar.
- ✓ El sistema no reconoce todos los formatos de documentos, ejemplo .pdf.

Teniendo en cuenta las limitaciones anteriores surge la necesidad de desarrollar una nueva propuesta de solución ante la problemática de la presente investigación.

1.4 Herramientas, lenguajes y tecnologías de desarrollo

Para dar solución al problema de esta investigación es necesario realizar un análisis de las herramientas y tecnologías más usadas en el desarrollo de estos tipos de sistemas, permitiendo definir las que se utilizarán para el desarrollo del módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión.

A continuación, se describen las herramientas, lenguajes y tecnologías de desarrollo identificadas para llevar a cabo el desarrollo del módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión.

1.4.1 Herramientas para el procesamiento lingüístico

En los estudios de homólogos realizados solo se encontró un sistema que realizara la extracción de tripletas, pero no le brindaba respuesta a la problemática de esta investigación. Por tal motivo se decidió investigar y estudiar herramientas que realizan un análisis semántico y sintáctico de los documentos. Estas son las siguientes: Apache OpenNLP, Link Gramar Parser, Minipar y Stanford parser y Freeling. A continuación, una explicación más detallada de cada una de ellas.

Apache OpenNLP

La biblioteca Apache OpenNLP es una herramienta de aprendizaje basada en herramientas para el procesamiento en lenguaje natural (NLP) de textos. Soporta las tareas más comunes de NLP, como la segmentación de oraciones, el etiquetado de parte del habla, la extracción de entidad con nombre, la fragmentación y el análisis sintáctico. Generalmente, estas tareas se requieren para crear servicios de procesamiento de texto más avanzados. OpenNLP también incluye la máxima entropía y aprendizaje basado en perceptron.org (Apache OpenNLP, 2014).

El objetivo del proyecto OpenNLP es crear un conjunto de herramientas para las tareas mencionadas. Un objetivo adicional es proporcionar un gran número de modelos pre-construidos para una variedad de idiomas, así como los recursos de texto anotado de los que se derivan esos modelos. (Rusu, 2007).

Estructura General de la Biblioteca: La biblioteca OpenNLP de Apache contiene varios componentes, lo que permite construir una línea completa de procesamiento de lenguaje natural. Estos componentes incluyen: detector de oraciones, buscador de nombres, categorizador de documentos y etiquetador de parte de la

voz. Los componentes contienen partes que permiten ejecutar la tarea de procesamiento de lenguaje natural correspondiente, formar un modelo y a menudo también evaluar un modelo. Cada una de estas instalaciones es accesible a través de su Interfaz de Programa de Aplicación (API). Además, se proporciona una Interfaz de Línea de Comandos (CLI) para la conveniencia de los experimentos y el entrenamiento.
Interfaz de programa de aplicación (API): Los componentes de OpenNLP tienen APIs similares. Normalmente, para ejecutar una tarea, se debe proporcionar un modelo y una entrada. Un modelo suele cargarse proporcionando un FileInputStream (Flujo de entrada de archivo) con un modelo a un constructor de la clase de modelo.

Link Grammar Parser

Link Grammar Parser es un analizador sintáctico del inglés, basado en la gramática del enlace, una teoría original de la sintaxis inglesa. Dada una oración, el sistema le asigna una estructura sintáctica, que consiste en un conjunto de enlaces etiquetados que conectan pares de palabras. El analizador también produce una representación "constituyente" de una oración (que muestra frases nominales y frases verbales) (Pysalo, 2006).
La compañía ha hecho todo el sistema disponible para descargar en la web. El sistema está escrito en código C genérico y se ejecuta en cualquier plataforma con un compilador C. Existe una API para facilitar la incorporación del analizador en otras aplicaciones. Link Gramar cuenta con diversas características (Temperley, 2016):

- ✓ El analizador tiene un diccionario de aproximadamente 60000 formas de la palabra.
- ✓ Tiene cobertura de una amplia variedad de construcciones sintácticas, incluyendo muchas raras e idiomáticas.
- ✓ El analizador es robusto.
- ✓ Es capaz de saltar porciones de la oración que no puede entender y asignar cierta estructura al resto de la oración.
- ✓ Es capaz de manejar vocabulario desconocido y hacer conjeturas inteligentes de contexto y ortografía sobre las categorías sintácticas de palabras desconocidas.

- ✓ Tiene conocimiento de mayúsculas, expresiones numéricas y una variedad de símbolos de puntuación.

- ✓ Licencia: A partir de diciembre de 2004, se lanzó el analizador bajo una nueva licencia; La licencia permite el uso sin restricciones en aplicaciones comerciales y también es compatible con la GNU GPL (Licencia Pública General).

MINIPAR

MINIPAR es un *parser* inglés basado en principios (Berwick, 1991), representa la gramática como una red. La evaluación basada en la dependencia del MINIPAR: Donde los nodos representan categorías gramaticales y los enlaces representan tipos de relaciones sintácticas (dependencia), se crean nodos y vínculos adicionales de forma dinámica para representar subcategorías de verbo. MINIPAR emplea un algoritmo de paso de mensajes que implementa esencialmente el análisis de gráficos distribuidos (Lin, 2003). En lugar de mantener un gráfico único, cada nodo de la red gramatical mantiene un gráfico que contiene estructuras parcialmente construidas pertenecientes a la categoría gramatical representada por el nodo. Los principios gramaticales se implementan como restricciones asociadas con los nodos y enlaces. El léxico en MINIPAR se deriva de WordNet (Miller, 1990). Las ambigüedades léxicas son manejadas por el analizador. Al igual que los analizadores de gráficos, MINIPAR construye todos los *parsers* posibles de una oración de entrada. Sin embargo, genera un único árbol de análisis con la clasificación más alta. Aunque la gramática se construye manualmente, la selección del mejor árbol de análisis se guía por la información estadística obtenida al analizar un corpus de 1GB con MINIPAR (Marneffer, 2006).

Entre los trabajos en sistemas de análisis de dependencias es indiscutible señalar a MINIPAR como un referente fundamental para el inglés. MINIPAR ha sido una herramienta profusamente utilizada en el ámbito del Procesamiento del Lenguaje Natural. Este tipo de sistemas son, realmente, generadores de analizadores de dependencias. Con los generadores de analizadores que basan su funcionamiento en el aprendizaje, se pueden obtener analizadores para cualquier lengua para la que se disponga de un corpus etiquetado con análisis de dependencias (Banko, 2007).

Stanford Parser

El Stanford Parser es un analizador sintáctico probabilístico desarrollado por la Universidad de Stanford que se basa en seleccionar el mejor análisis según aquel que sea más probable a partir de un conjunto de

ejemplos analizados correctamente por lingüistas.

El software es, en esencia, un clasificador que separa un texto en términos y no términos. En algoritmos de aprendizaje automático los términos son los llamados datos de entrenamiento, mientras que los no términos son datos irrelevantes que no afectan al clasificador, por lo que el Stanford Parser desecha estos datos. Es muy importante, por ello, que los términos sean anotados por expertos de la materia.

Actualmente la mayor parte de los analizadores sintácticos modernos emplean el enfoque de algoritmo combinatorio, mientras el Stanford Parser está basado en un ámbito puramente probabilístico para hacer la división de términos. Por lo que, necesita ser entrenado con un *Corpus* o *Penn Treebank* previamente anotado con información morfológica (Moreno-Sandoval y Campillos-Llanos, 2015).

El uso de algoritmos probabilísticos hace del software una herramienta complicada en el uso, pues emplea multitud de parámetros para intentar ajustar el análisis en su ejecución. Además, al emplear este enfoque probabilístico, se realiza un análisis ligero, es decir, se centra solo en los núcleos principales de la oración, el verbo y los sustantivos. En este tipo de análisis solo se tiene en cuenta el nivel de análisis sintáctico de la lingüística.

Los sustantivos desempeñan funciones muy diversas dentro de una frase, pero para este caso las funciones destacadas en las que se centra el analizador sintáctico son las de sujeto y objetos directo e indirecto, en otras palabras, las funciones inmediatamente dependientes del verbo. Lo que concluye que el verbo es el elemento principal de la oración, el sujeto y los objetos los elementos secundarios y relevando al resto de componentes a un plano más lejano (Colmenarejo, 2016).

El Stanford Parser trabaja con la dependencia del software coreNPL de la Universidad de Stanford. Este elemento proporciona un conjunto de herramientas de análisis lingüístico. La incorporación de esta dependencia al Stanford Parser proporciona, entre otras, las anotaciones que soporta cada lenguaje humano. Al ser una herramienta orientada principalmente al inglés, las anotaciones para otros lenguajes no están del todo desarrolladas, siendo el español el que menos anotaciones incorpora después del árabe.

Freeling

FreeLing es una librería de código abierto para el procesamiento multilingüe automático, que proporciona una amplia gama de servicios de análisis lingüístico para diversos idiomas. FreeLing ofrece a los desarrolladores de aplicaciones de Procesamiento del Lenguaje Natural funciones de análisis y anotación lingüística de textos, con la consiguiente reducción del coste de construcción de dichas aplicaciones. FreeLing es personalizable, ampliable y está fuertemente orientado a aplicaciones del mundo real en

términos de velocidad y robustez. Los desarrolladores pueden utilizar los recursos lingüísticos por defecto (diccionarios, lexicones, gramáticas y otros), ampliarlos, adaptarlos a dominios particulares, o (dado que la librería es de código abierto) desarrollar otros nuevos para idiomas específicos o necesidades especiales de las aplicaciones (Padro,2011).

FreeLing está concebido como una librería sobre la cual se puedan desarrollar potentes aplicaciones de PLN[5] y orientado a facilitar la integración con las aplicaciones de niveles superiores de los servicios lingüísticos que ofrece. La arquitectura de la librería se basa en un enfoque de dos capas cliente-servidor: una capa básica de servicios de análisis lingüístico (morfológico, morfosintáctico, sintáctico y semántico) y una capa de aplicación que, actuando como cliente, realiza las peticiones deseadas a los analizadores y usa su respuesta según la finalidad de la aplicación. La arquitectura interna de la librería se estructura en dos tipos de objetos: los que almacenan datos lingüísticos con los análisis obtenidos y los que realizan el procesamiento en sí (Castañeda,2014).

A pesar de que las herramientas no están implementadas para el idioma español (solo tres de ellas poseen algunas anotaciones para este lenguaje), teniendo en cuenta la necesidad de desarrollar una nueva propuesta de solución se utilizarán algunas características que tienen estas herramientas las cuales se mencionan en la siguiente tabla:

Tabla 2 Aspectos utilizados de cada herramienta para el procesamiento lingüístico.

Aspecto utilizados	Apache Opennlp	Minipar	Link Grammar Parser	Stanford Parser	Freeling
Algoritmo		X	X	X	
Dependencias de procesamientos de lenguaje natural	X			X	X
Anotaciones que soportan el lenguaje humano				X	
Librerías para el idioma español				X	X
Reconocedor de expresiones temporales (fechas/horas).					X

[5] PLN: Procesamiento de lenguaje natural.

Reconocedor de nombres propios				X	X

1.4.2 Mecanismos existentes para la indexación de información

Elasticsearch

Elasticsearch es un servidor de búsqueda basado en Lucene. Provee un motor de búsqueda de texto completo, distribuido y con capacidad de multi-tenencia con una interfaz web REST y con documentos JSON. Elasticsearch está desarrollado en Java y está publicado como código abierto. Elasticsearch puede ser usado para buscar todo tipo de documentos. Provee búsqueda escalable, con búsqueda casi a tiempo real y soporta multi-tenencia. “Es distribuido, haciendo que los índices se puedan dividir en fragmentos y cada uno teniendo cero o más réplicas. Cada nodo alberga uno o más fragmentos, actuando como un coordinador para delegar operaciones a los fragmentos correctos. El rebalanceo y ruteo se realizan automáticamente. Entre sus características se destacan:

- ✓ Está libre de esquemas de datos, en el sentido de que no necesita disponer de una definición explicita del esquema.
- ✓ Búsquedas Facetadas: Muestra contador para cada categoría en los resultados de búsqueda.
- ✓ Búsqueda Geo-espacial: Búsqueda por localización y distancia. (Buscar dentro de 5 km de la posición actual).
- ✓ Los documentos (datos) no necesariamente tienen que ser planos, también se permite elementos anidados.
- ✓ Arquitectura diseñada pensando siempre en la distribución para permitir escalar una solución de un nodo a cientos, ofreciendo alta disponibilidad, soportando grandes cantidades de datos y cortos tiempos de respuesta.
- ✓ Posee búsqueda distribuida, es decir, la búsqueda puede realizarse en varios fragmentos/índices y al concluir la misma los resultados serán agregados.
- ✓ Presenta indexación distribuida lo que significa que los documentos van a ser almacenados en distintos nodos.

Actualmente Elasticsearch presenta una comunidad pequeña de colaboradores y consecuentemente una base de usuarios pequeña (Elasticsearch, 2014).

Sphinx

Motor de búsqueda de texto completo, distribuido públicamente bajo licencia GPL, su nombre es un acrónimo que se decodifica oficialmente como SQL Phrase Index. Fue especialmente diseñado para integrarse con bases de datos SQL y ser de fácil acceso para los lenguajes script. Ofrece la funcionalidad de búsqueda rápida y relevante de texto completo para aplicaciones clientes. Sin embargo, Sphinx no depende ni requiere ninguna base de datos específica para funcionar (Nugraha, 2014).

Entre sus principales características se encuentran:

- ✓ Conjunto de resultados avanzados de post-procesamiento (SELECT con expresiones, WHERE, ORDER BY, GROUP BY, HAVING entre otros, sobre los resultados de búsqueda de texto).
- ✓ Comprobada escalabilidad hasta miles de millones de documentos, terabytes de datos y miles de consultas por segundo (AKSYNOFF, 2014).
- ✓ Posee una alta velocidad de indexación (hasta 10-15 MB / seg por núcleo).
- ✓ Posee una alta velocidad de búsqueda (hasta 150-250 consultas / seg por núcleo contra 1.000.000 documentos, 1.2 GB de datos).

Sphinx es adecuado para aquellos que ya tienen contenido digital existente dentro de los servidores de bases de datos tales como MariaDB, PostgreSQL, MS SQL y desea que el contenido sea indexado para una rápida recuperación sin tener que convertir a otro formato primero.

Solr

Solr en su versión 4.10.3 es un servidor de índice empresarial, que actúa como una base de datos no SQL. Una plataforma de búsqueda de código abierto basada en Apache Lucene. Sus características principales incluyen potentes búsquedas de texto completo, resaltado de búsqueda de facetas, realiza indexación en tiempo real, *clustering*[6] dinámico, integración de bases de datos y manejo de documentos (por ejemplo, Word, PDF). Solr es altamente confiable, escalable y tolerante a fallos, proporcionando indexación

[6] Clustering: Un algoritmo de agrupamiento es un procedimiento de agrupación de una serie de vectores de acuerdo con un criterio. Esos criterios son por lo general distancia o similitud (Cardona.2006).

distribuida y configuración centralizada.

Solr está escrito en Java y se ejecuta como un servidor de búsqueda de texto completo independiente dentro de un contenedor de servlets como Tomcat. Solr Lucene utiliza la biblioteca Java de búsqueda en su base para la indexación de texto completo y de búsqueda y tiene como REST HTTP / XML y JSON APIs que hacen que sea fácil de utilizar desde prácticamente cualquier lenguaje de programación. La potente configuración externa de Solr permite que sea adaptado a casi cualquier tipo de aplicación Java sin codificación, simplemente hay que utilizarlo con peticiones GET para realizar las búsquedas en el índice y POST para agregar documentos (The Apache Software Foundation-Solr, 2014).

A continuación, se realiza una tabla comparativa entre los diferentes mecanismos de indexación identificados para seleccionar el que más aporta al desarrollo de la propuesta solución.

Tabla 3 Comparación entre mecanismos de indexación

Nombre	**Elasticsearch**	**Solr**	**Sphinx**
Documentación técnica	www.elasticsearch.org/guide	Lucene.apache.org*solr/d	Sphinxsearch.com/docs
Sistema Operativo	Todos con la MV de Java	Todos con la MV de Java y contenedor de servlets	FreeBSD, Linux, NetBSD, OS X, Solaris, Windows
Lenguaje de implementación	Java	Java	C++
Extensibilidad	No	Plugins de Java	No
Integridad en la manipulación de datos.	No	Bloqueo Optimista	No
Persistencia de datos	Si	Si	Si

APIs y otros métodos de acceso	API de Java, API RESTful HTTP/JSON	API de Java, API RESTful HTTP	Proprietary protocol

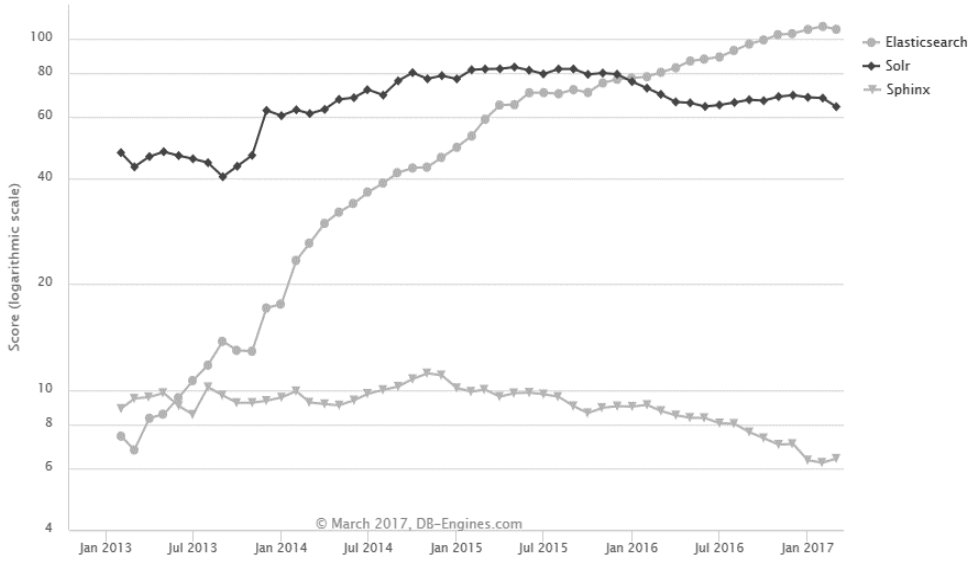

Figura 4 Comparativa de Elasticsearch, Apache Solr y Sphinx (Fuente: DB-Engines, 2017).

Como se muestra en la Figura 4 tanto Solr como Elasticsearch gozan de una gran popularidad, ambos son escritos en Java. Elasticsearch no presenta soporte para la manipulación de datos concurrentemente, mientras que Solr sí. Sphinx fue especialmente diseñado para integrarse con los servidores de bases de datos SQL y ser de fácil acceso para los lenguajes de script, mientras que, a diferencia de este, otras soluciones de indexación de contenido digital como Solr o ElasticSearch, ofrecen sus servicios en forma de una API REST y son de propósito general sin necesidad de integrarse a una base de datos SQL (DB-ENGINES, 2015).

Teniendo en cuenta la comparación realizada, la popularidad de la cual goza Solr y añadiendo que este último, hoy es el Servidor de índice utilizado por Nutch[7] en el motor de búsqueda Orión se decide utilizar como motor de indexación a Solr.

[7] Nutch: es un robot y motor de búsqueda.

1.4.2 Lenguajes de desarrollo

Los lenguajes de programación son un conjunto de programas que controlan el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Están formados por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones (Hidalgo, 2013).

1.4.3 Lenguaje de Programación

En la actualidad existen disímiles lenguajes de programación cada uno de ellos con características que lo distinguen, a continuación, se expone el que se utilizará.

Java

El lenguaje de programación Java, fue diseñado por la compañía Sun Microsystems, que describen el lenguaje de java como “simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico” (Rodríguez, 2003).
A continuación, se explican algunas características (Aquilar, 2012):

- ✓ Sencillo: Los lenguajes de programación orientados a objetos no son tan sencillos ni fáciles de utilizar, pero Java es un poco más fácil que el popular C++, lenguaje de desarrollo de software más popular hasta la implementación de Java. Este último elimina los punteros de C++ y reemplaza la herencia múltiple de C++ en una estructura única denominada interfaz (interface8).
- ✓ Orientado a Objetos: Debido a que la programación en Java se centra en la creación, manipulación y construcción de objetos. Un objeto tiene propiedades (un estado) y un comportamiento.
- ✓ Interpretado: Es interpretado y necesita un intérprete para ejecutar programas de Java. Los programas se compilan en la Máquina Virtual de Java (JVM, Java Virtual Machine) generándose un código interpretado denominado bytecode9. El *bytecode* es independiente de la máquina y se puede ejecutar en cualquier máquina que tenga un intérprete de Java.
- ✓ Multihilo: Es la capacidad de un programa de ejecutar varias tareas simultáneamente.

Se selecciona el lenguaje Java para el desarrollo del módulo debido a que las principales herramientas con las que trabaja el buscador Orión utilizan ese lenguaje y las librerías utilizadas como base para el desarrollo del módulo están implementadas en java.

1.4.4 Lenguaje de modelado

El Lenguaje Unificado de Modelado (UML)

UML es el lenguaje estándar especificado por el *Object Management Group* (OMG) para visualizar, especificar, construir y documentar los artefactos de un sistema, incluyendo su estructura y diseño. Utiliza un conjunto de símbolos y notaciones para representar gráficamente los diversos componentes que forman parte de la arquitectura de software. Permite el modelado de procesos de negocio y el modelado de requisitos apoyándose en el análisis orientado a objetos (Dumas, 2013). En la presente investigación se escoge UML como lenguaje de representación visual.

1.4.5 Herramienta CASE

Las herramientas CASE son aplicaciones informáticas que permiten agilizar la realización de determinadas actividades del proceso de desarrollo de software. Utilizadas mayoritariamente en la modelación y análisis de procesos y sistemas, a través de diagramas; pueden contribuir eficazmente a la reducción de costos en términos de tiempo y dinero. Lo anterior garantiza el aumento de la productividad en el desarrollo de software de manera significativa. Entre las facilidades más apremiadas que proporcionan estas herramientas se encuentran: el diseño de proyectos, cálculo de costos, implementación automática de parte del código dado un diseño previo, compilación automática, documentación, detección de errores, entre otras.

1.4.6 Visual Paradigm

Visual Paradigm es una herramienta CASE multiplataforma, que soporta el ciclo completo de desarrollo de software: análisis, diseño, implementación y pruebas. Facilita la construcción de aplicaciones informáticas con un menor coste que destacan por su alta calidad y contribuye a mejorar la experiencia de usuario mediante el diseño de un gran número de artefactos de ingeniería de software. Permite la generación de bases de datos, conversión de diagramas entidad-relación a tablas de base de datos, mapeos de objetos y relaciones, ingeniería directa e inversa, la gestión de requisitos de software y la modelación de procesos del negocio (Visual Paradigm, 2014).

1.4.7 Entorno de Desarrollo Integrado

Un entorno de desarrollo integrado o IDE (siglas en inglés de *Integrated Development Environment*) consiste en un conjunto de herramientas de programación puestas a disposición como un programa informático. En otras palabras, puede estar compuesto por un editor de código, un compilador, un intérprete, un sistema de

apoyo al control de versiones, un constructor de interfaces gráficas, entre otras herramientas comúnmente utilizadas para el desarrollo de software.

Eclipse

Eclipse es una plataforma de desarrollo, diseñada para ser extendida de forma indefinida a través de *plugins*. Fue concebida desde sus orígenes para convertirse en una plataforma de integración de herramientas de desarrollo. No tiene en cuenta un lenguaje específico, sino que es un IDE genérico, aunque goza de mucha popularidad entre la comunidad de desarrolladores del lenguaje Java usando el *plug-in* JDT que viene incluido en la distribución estándar del IDE.

Proporciona herramientas para la gestión de espacios de trabajo, escribir, desplegar, ejecutar y depurar aplicaciones.

Principales características

- ✓ Perspectivas, editores y vistas: en Eclipse el concepto de trabajo está basado en las perspectivas, es decir, una pre configuración de ventanas y editores, relacionadas entre sí y que nos permiten trabajar en un determinado entorno de trabajo de forma óptima.

- ✓ Gestión de proyectos: el desarrollo sobre Eclipse se basa en los proyectos, que son el conjunto de recursos relacionados entre sí, como puede ser el código fuente, documentación, ficheros configuración, árbol de directorios. El IDE proporcionará asistentes y ayudas para la creación de proyectos. Por ejemplo, cuando se crea uno, se abre la perspectiva adecuada al tipo de proyecto que se esté creando, con la colección de vistas, editores y ventanas pre configuradas por defecto.

- ✓ Depurador de código: se incluye un potente depurador, de uso fácil e intuitivo y que visualmente ayuda a mejorar el código. Para ello sólo debemos ejecutar el programa en modo depuración (con un simple botón). De nuevo, tenemos una perspectiva específica para la depuración de código, la perspectiva depuración, donde se muestra de forma ordenada toda la información necesaria para realizar dicha tarea.

1.5 Metodología de Desarrollo

Una metodología de ingeniería de software es un proceso para la producción organizada del Software, empleando para ello una colección de técnicas predefinidas y convencionales en las notaciones. Una

metodología se presenta normalmente como una serie de pasos, con técnicas y notaciones asociadas a cada paso. Los pasos de la producción del software se organizan normalmente en un ciclo de vida consistente en varias fases de desarrollo (Tomassetti, 2013).

En el desarrollo del módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión se emplea la variación creada en la UCI de la metodología AUP:

El Proceso Unificado Ágil (AUP por sus siglas en inglés) de Scott Ambler o *Agile Unified Process* (AUP) en inglés es una versión simplificada del Proceso Unificado de *Rational* (RUP). Este describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. El AUP aplica técnicas ágiles incluyendo: Desarrollo Dirigido por Pruebas (*test driven development*-TDD en inglés), modelado ágil, gestión de cambios ágil, refactorización de base de datos para mejorar la productividad y otros.

Al igual que en RUP, en AUP –UCI se establecen tres fases que transcurren de manera consecutiva. Que son: Inicio, ejecución y cierre. AUP-UCI se preocupa especialmente de la gestión de riesgos. Propone que aquellos elementos con alto riesgo obtengan prioridad en el proceso de desarrollo y sean abordados en etapas tempranas del mismo. Se logra estandarizar el proceso de desarrollo de software, dando cumplimiento además a las buenas prácticas que define CMMI-DEV v1.3. Se logra hablar un lenguaje común en cuanto a fases, disciplinas, roles y productos de trabajos (Rodríguez, 2015).

1.6 Conclusiones parciales

Luego de conformar el marco teórico y realizar el estudio del arte de la investigación, se concluye lo siguiente:

- ✓ La revisión bibliográfica evidenció que la extracción de tripletas de documentos, es un área de investigación en constante desarrollo, principalmente en el contexto de análisis semántico.
- ✓ El estudio realizado al sistema homólogo permitió identificar que no brindaba la solución absoluta al problema, por lo que surge la necesidad de implementar un módulo capaz de extraer tripletas a documentos y a su vez dé cumplimiento a las necesidades del motor de búsqueda Orión.
- ✓ La investigación realizada a las tecnologías informáticas definió la base tecnológica que se utilizará en el desarrollo del módulo, seleccionando a AUP - UCI como metodología para guiar los pasos del desarrollo, UML como lenguaje de representación visual y Visual Paradigm 8.0 como herramienta CASE para el modelado del sistema. Se escogió Java 1.8 como lenguaje para el desarrollo y definiendo como entorno de desarrollo integrado Eclipse Luna SR2. Definiendo como mecanismo

indexador a Solr 4.10.3 y como servidor web Apache Tomcat 7.0.

Capítulo 2. Análisis y diseño del Módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión.

2.1 Introducción

En el presente capítulo se define la propuesta solución de la presente investigación. Se describe el entorno mediante un modelo de dominio en el cual se analizan conceptos, entidades y sus relaciones. También se definen las principales características que debe cumplir el módulo en términos de requisitos funcionales y no funcionales. Así como se describen las historias de usuarios que se materializarán en funcionalidades. Además, se presentan los diagramas de clases de diseño para una visión general del funcionamiento del módulo y el diagrama de despliegue para una mejor comprensión entre la correspondencia de la arquitectura de *software* y la arquitectura de *hardware*.

2.2 Propuesta de Solución

Para darle respuesta al problema de la presente investigación se define como propuesta de solución desarrollar un módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión. Los principales procesos son los siguientes:

Primeramente, de los documentos que se encuentran indexados en la base de datos de Sorl se escoge su contenido para extraer como primer paso las oraciones. Este proceso se realiza mediante el método "*sentence"* que es uno de los métodos principales de *parser* e identifica donde comienza y acaba cada una de las oraciones de un documento. Luego de haber extraído las oraciones el módulo pasa al segundo proceso el cual consiste en extraer el sintagma nominal sujeto, la forma verbal y el sintagma verbal predicado a partir de otro de los métodos principales, en este caso el método "*extraction*". La forma en que lo realiza es a partir de una librería con la que cuenta el *parser* utilizado. Dicha librería es capaz de identificar sustantivos, verbos, adjetivos, pronombres y otros. De esta manera y con el método antes mencionado se realiza una extracción de las partes de la oración con un 90% de perfección. El tercer proceso que realiza el módulo es conformar la tripleta, este paso se realiza en la clase "*triplet*" donde el módulo define qué son cada uno de los componentes que se extrajeron con anterioridad, además de crear la estructura con la que se muestra la tripleta en Solr. Luego se procede al proceso final donde después de conformar la tripleta se crea un nuevo campo "*triplets*", en el cual mostrará todas las tripletas identificadas en el servidor indexador Solr, ver en la Figura 5 todos los procesos. De esa manera se contribuye al procesamiento semántico de los documentos indexados en la base de datos de Solr.

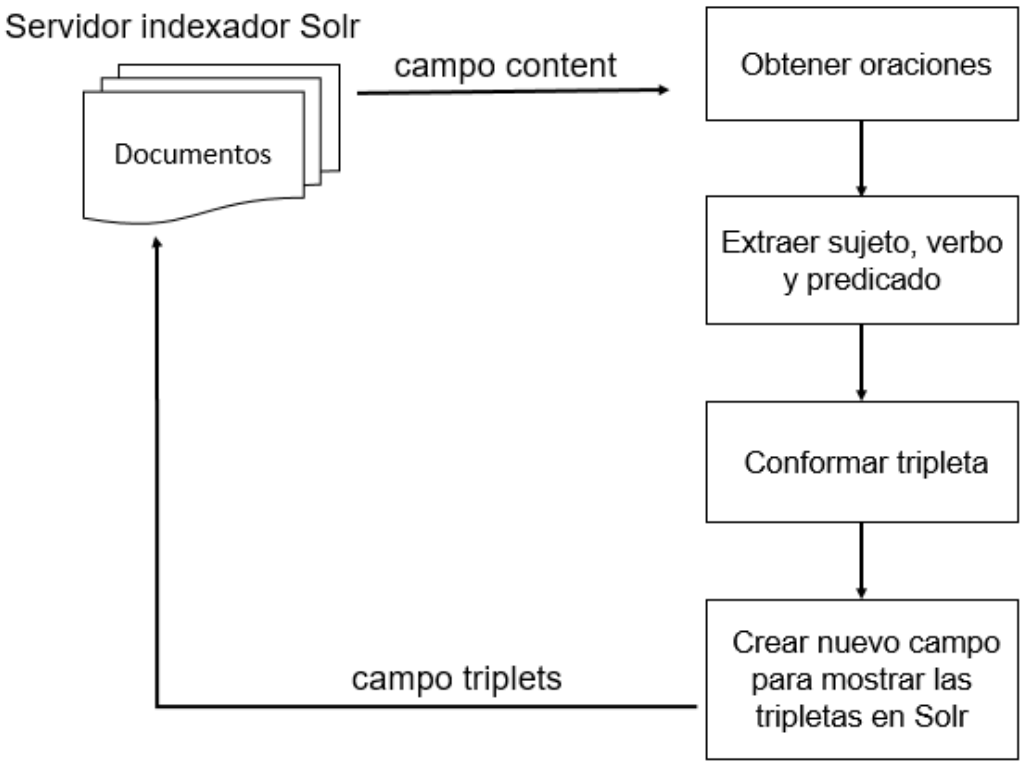

Figura 5 Propuesta de solución del "Módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión". (Fuente: elaboración propia)

2.3 Modelo del dominio

El modelo de dominio también conocido como Modelo Conceptual es una representación visual de los conceptos u objetos del mundo real, significativos para un problema o área de interés (García M, 2007).Para lograr un mejor razonamiento de la presente investigación se hace necesario describir el procedimiento de extracción de tripletas de documentos mediante una serie de conceptos, entidades y sus relaciones, agrupándose en un modelo de dominio con el fin de contribuir a la comprensión del contexto del módulo.

2.3.1 Descripción de clases del modelo de dominio

La modelación del dominio constituye la herramienta fundamental para garantizar la descripción y comprensión de las clases o conceptos y sus relaciones más importantes dentro del contexto del problema. A continuación, se presenta una tabla con la descripción de cada una de las clases del modelo de Dominio.

Tabla 4 Descripción de las clases del modelo del dominio

Conceptos	**Descripción**
Usuario	Persona que realiza las búsquedas.
Buscador	Constituye una herramienta de recuperación de información en la Web.
Rastreador	Mecanismo que se encarga de recopilar los documentos en la Web.
Servidor de índice	Mecanismo que se encarga de indexar los documentos.
Interfaz web	Constituye la vista mediante la cual el usuario realiza consultas y se le muestran los resultados a este.
Intranet	Red informática interna donde se administran los documentos.
Documento	Recursos publicados en la Web (páginas web, imágenes, videos, documentos ofimáticos.

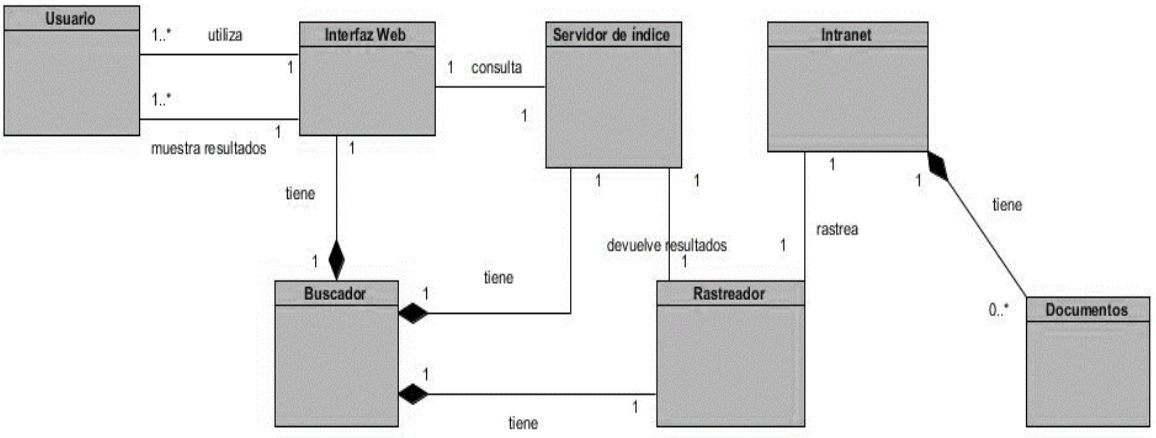

Figura 6 Modelo de dominio.

La Figura 6 muestra la relación existente entre todos los conceptos que intervienen en la investigación que se presenta. Se puede observar que el usuario realiza peticiones y recibe los resultados a través de una

interfaz web con la que cuenta el buscador. A partir de la solicitud de información realizada se consulta al servidor de índice Solr en el cual se almacena la información (documentos) rastreada y procesada por Nutch que se encuentran en la intranet. Finalmente, se le devuelve el resultado de búsqueda al usuario.

2.4 Especificación de los Requisitos del Software

En la ingeniería del software, en la etapa de diseño del producto, los requisitos se utilizan como datos de entrada y establecen qué debe hacer el sistema, pero no cómo hacerlo. Son una condición o capacidad que un usuario necesita para poder resolver un problema o lograr un objetivo. De manera general estos requisitos son lo que el sistema debe hacer o una cualidad que el sistema debe poseer (Somerville, 2007). La definición de requisito en la literatura científica cuenta con varias acepciones. La Real Academia Española lo define como: circunstancia o condición necesaria para algo. Por otra parte, IEEE en (ISO 2011) enuncia el concepto de la siguiente manera: declaración que se traduce o expresa una necesidad y sus limitaciones y las condiciones correspondientes. A continuación de listan las técnicas empleadas para la captura de requisitos, los requisitos funcionales y no funcionales identificados y las técnicas para su validación.

Técnicas de captura de requisitos.

Las técnicas de captura de requisitos trabajadas en la presente investigación son (Monsalve, 2010):

Tormenta de ideas

La tormenta de ideas (del inglés: *brainstorming*) es una técnica de reuniones en grupo, se caracteriza por ser sencilla y fácil de aplicar. El objetivo de la misma es que los participantes muestren sus ideas en un ambiente libre de críticas o juicios. Consiste en la mera acumulación de ideas y/o información sin evaluar las mismas. Las tormentas de ideas suelen ofrecer una visión general de las necesidades del sistema, pero normalmente no sirve para obtener detalles concretos del mismo, por lo que suele aplicarse en los primeros encuentros.

Reunión con el cliente

Resulta una técnica muy aceptada dentro de la ingeniería de requisitos y su uso está ampliamente extendido. A través de esta técnica el equipo de trabajo se acerca al problema de una forma natural y le permite comprender los objetivos de la solución buscada.

Prototipado

Algunas propuestas se basan en obtener de la definición de requisitos prototipos que, sin tener la totalidad

de la funcionalidad del sistema, permitan al usuario hacerse una idea de la estructura de la interfaz del sistema con el usuario. Un prototipo en software es un modelo del comportamiento del sistema que puede ser usado para entenderlo completamente o ciertos aspectos de él y así clarificar los requisitos (Montoya, 2013).

2.4.1 Requisitos Funcionales

El objetivo principal de la captura de los requisitos es guiar el desarrollo hacia el sistema correcto. Esto se consigue mediante una descripción de los requisitos del sistema suficientemente buena como para que pueda llegarse a un acuerdo entre el cliente (incluyendo a los usuarios) y los desarrolladores sobre qué debe y qué no debe hacer el sistema (Fuentes M, 2011).

RF 1. Identificar oraciones dado un documento.

RF 2. Identificar sintagma nominal sujeto de cada oración.

RF 3. Identificar verbos de cada oración.

RF 4. Identificar sintagma verbal predicado de cada oración.

RF 5. Confeccionar las tripletas necesarias dado el sujeto, relación y predicado de una oración.

RF 6. Añadir el campo tripleta para cada documento en Solr.

RF 7. Mostrar todas las tripletas asociadas a un mismo documento en el campo tripleta.

2.4.2 Requisitos no Funcionales

Para especificar los criterios que evalúan la operación del sistema, en contraste con los requisitos funcionales que especifican los comportamientos específicos se plantean los requisitos no funcionales los cuales describen las características del funcionamiento del módulo de la presente investigación.

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto usable, rápido o confiable. Los requerimientos no funcionales, aunque no aportan funcionalidades propiamente dichas dentro del módulo, son de vital importancia para una puesta en marcha exitosa del software y para lograr que este responda a las expectativas del usuario.

Requerimientos de software:

- ✓ RNF 1 Se requiere que el sistema operativo que tenga instalada la máquina virtual de Java.
- ✓ RNF 2 Se requiere la instalación del servidor web y de servlets Tomcat 7 para el correcto funcionamiento del servidor de Solr.

Requerimientos de hardware:

- ✓ RNF 3 Para el servidor de índice se necesita como mínimo: 4 GB RAM, CPU de 4 núcleos y 1TB de almacenamiento.

Restricciones de Diseño e implementación:

- ✓ RNF 5 Como lenguaje de programación para el *plugin* del componente se deberá utilizar Java.

2.4.3 Historias de usuario

Luego de realizar el levantamiento de requisitos se decide crear las historias de usuario para así tener una mejor descripción de cada uno de los requisitos funcionales.

Las historias de usuario son utilizadas en las metodologías de desarrollo ágil puesto que son una forma rápida para la especificación y administración de requisitos, sin necesidad de elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Las historias de usuario permiten responder rápidamente a los requisitos cambiantes y deben cumplir varias restricciones, entre ellas: ser independientes unas de otras, negociables, estimables, pequeñas y verificables, por mencionar algunas. Seguidamente se enumeran las historias de usuario más importantes definidas para las actividades del módulo propuesto las demás se encuentran en el Anexo 2.

Historias de usuario:

Tabla 5 Descripción de la Historia de usuario # 2.

	Historia de usuario
Número: HU #02	**Nombre Historia de Usuario** Identificar sintagma nominal sujeto de cada oración.
Prioridad en negocio: Alta	
Descripción: Permite identificar el sujeto de una oración. La funcionalidad comienza cuando el módulo se ejecuta y comienza a buscar el sujeto de la primera oración, lo realiza de acorde a un diccionario que contiene el mismo módulo en el cual cada parte de la oración (dígase sujeto, verbo, sustantivo o adjetivo) tiene un identificador, si alguna de esas palabras que contiene la oración coincide con algún identificador que corresponda al sujeto, el módulo muestra esa palabra como sujeto de la oración.	

En caso de que el sujeto este omitido, el módulo devuelve: sujeto omitido.

Tabla 6 Descripción de la Historia de usuario # 3.

Historia de usuario	
Número: HU #03	**Nombre Historia de Usuario:** Identificar verbos de cada oración.
Prioridad en negocio: Alta	
Descripción: Permite identificar el verbo de una oración. La funcionalidad comienza cuando el módulo se ejecuta y comienza a buscar el verbo de la primera oración, lo realiza de acorde a un diccionario que contiene el mismo módulo en el cual cada parte de la oración (dígase sujeto, verbo, sustantivo o adjetivo) tiene un identificador, si alguna de esas palabras que contiene la oración coincide con algún identificador que corresponda al verbo el módulo muestra esa palabra como verbo de la oración. En caso de que el verbo este omitido, el módulo devuelve: verbo omitido.	

Tabla 7 Descripción de la Historia de usuario # 4.

Historia de usuario	
Número: HU #04	**Nombre Historia de Usuario:** Identificar sintagma verbal predicado de cada oración.
Prioridad en negocio: Alta	
Descripción: Permite identificar el predicado de una oración. La funcionalidad comienza cuando el módulo se ejecuta y luego de haber identificado el sujeto y el verbo de la oración, todo lo que resta de la oración el módulo lo muestra como predicado. Ejemplo: La bandera tiene tres preciosos colores. El módulo identifica: bandera como sujeto, tiene como forma verbal y tres preciosos colores como predicado.	

2.5 Estilo Arquitectónico.

Los estilos expresan la arquitectura en el sentido más formal y teórico, describen entonces una clase de arquitectura, o piezas identificables de las arquitecturas empíricamente dadas. Esas piezas se encuentran repetidamente en la práctica, trasuntando la existencia de decisiones estructurales coherentes. Una vez que se han identificado los estilos, es lógico y natural pensar en reutilizarlos en situaciones semejantes que se presenten en el futuro. (Quiñones, 2015).

El módulo para la extracción de tripletas de documentos indexado por el motor de búsqueda Orión como propuesta de solución sigue un estilo arquitectónico de flujo de datos, que enfatiza la reutilización y

modificabilidad; siendo apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos. Se recomienda ser aplicada en escenarios donde se demande el uso de un índice para lograr mejores resultados a consultas formuladas por usuarios. La arquitectura utilizada es de tuberías y filtros, que consiste en ir transformando un flujo de datos en un proceso comprendido por varias fases secuenciales, siendo la entrada de cada una la salida de la anterior. Una tubería (del inglés, pipeline) conecta componentes computacionales (filtros) a través de conectores (del inglés, pipes), de modo que las computaciones se ejecutan a la manera de un flujo. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas (Reynoso, 2004). En la siguiente figura se muestran los pasos fundamentales de la propuesta de solución.

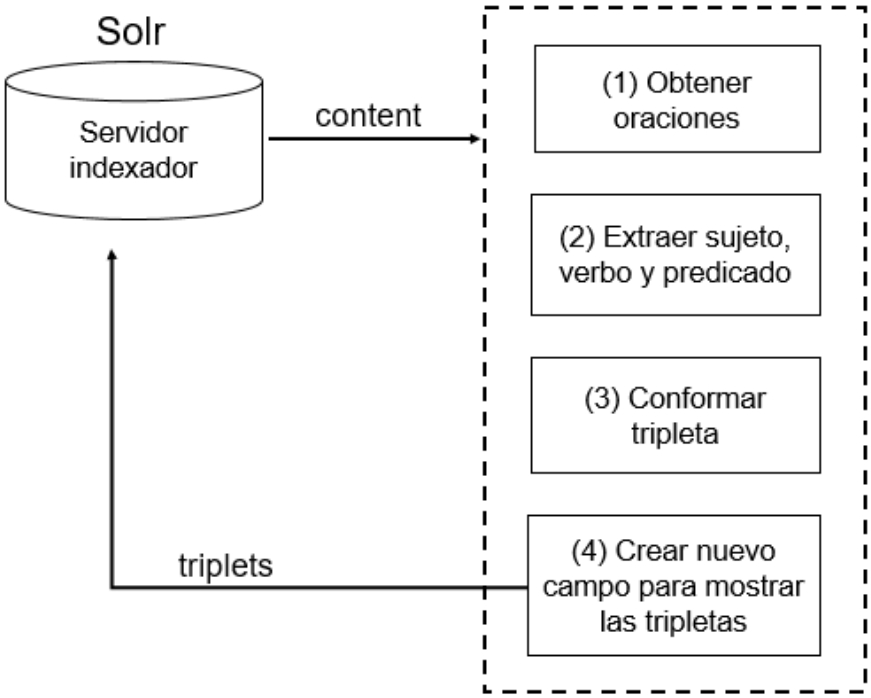

Figura 7 Arquitectura del módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión. (Fuente: elaboración propia)

En la Figura 7 se muestra la propuesta para la extracción de tripletas de documentos indexados en Solr. Se refleja el orden de las actividades que lo componen. La arquitectura presentada, contempla dos elementos fundamentales: (1) el módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión y (2) el mecanismo indexador Solr. La interacción entre ambos se debe a la necesidad de

encontrar una alternativa que contribuya al procesamiento semántico de los documentos indexados en Solr.

2.6 Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Son parejas de problema/solución con un nombre, que codifican buenos principios y sugerencias relacionados generalmente con la asignación de responsabilidades (Agudelo, A, 2015).

En el diseño de la herramienta se tuvieron en cuenta los patrones GRASP (Patrones Generales de Software para Asignación de Responsabilidades), los cuales describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.

De los patrones GRASP se utilizan los siguientes:

Experto: Este patrón plantea que se debe asignar una responsabilidad al experto en información, o sea, a la clase que cuenta con la información necesaria para cumplir la responsabilidad.

Beneficios del patrón Experto:

Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto posibilita tener sistemas de fácil mantenimiento. El comportamiento se distribuye entre las clases que cuentan con la información requerida para cumplir con la responsabilidad asignada.

En la solución implementada se pone de manifiesto este patrón en la clase *Triplet*, debido a que domina la información de su clase la cual es sujeto-relación-predicado.

Creador: Este patrón plantea que se debe asignar a una clase X la responsabilidad de crear una instancia de una clase Y. La creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. Este patrón es el encargado de guiar la asignación de responsabilidades relacionadas con la creación de objetos.

Beneficios del patrón Creador:

Se crean menos dependencias y existe mayor posibilidad de reutilización de código.

Este patrón se pone de manifiesto en la clase *Triplet*, ya que es la encargada de crear la estructura árbol y luego crear sujeto-relación-predicado.

Bajo acoplamiento: Este patrón plantea que se debe asignar las responsabilidades de forma tal que las clases se comuniquen con el menor número de clases que sea posible. Una clase con bajo acoplamiento no depende de muchas otras.

Beneficios del patrón Bajo acoplamiento

Fáciles de entender por separado y fáciles de reutilizar.

Este patrón se evidencia en la clase *Triplet* debido a que solamente llama al método *Tree*, o sea que solo depende de una sola clase la cual es la clase fundamental. Esta clase es donde se guardan las instancias para almacenar las tripletas.

Alta cohesión: Este patrón plantea que se debe asignar una responsabilidad de modo que la cohesión siga siendo alta (una clase tiene responsabilidades moderadas). Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas, que no realicen un trabajo enorme. Una clase con baja cohesión hace muchas cosas no afines o un trabajo excesivo.

Beneficios del patrón Alta cohesión

Mejoran la claridad y la facilidad con que se entiende el diseño. Además, se simplifican el mantenimiento y las mejoras en funcionalidad.

El patrón de alta cohesión se evidencia en la clase *Triplet*, en la cual se hace responsable de los documentos y el análisis respectivamente.

2.7 Modelo de diseño

Para transmitir, a través de la representación de diagramas, una comprensión en profundidad de los aspectos relacionados con los requerimientos no funcionales y restricciones se crea el modelo de diseño.

El modelo de diseño es un modelo de objetos que describe la realización de los casos de uso y sirve como una abstracción del modelo de implementación y el código fuente. Su objetivo fundamental es adquirir una comprensión en profundidad de los aspectos relacionados con los requerimientos no funcionales y restricciones relacionados con los lenguajes de programación (Larman, 2003).

2.7.1 Diagrama de clases del diseño

A continuación, se muestran los Diagramas de clases del diseño de los principales casos de uso del módulo en el cual se expresan las relaciones entre las clases. La única diferencia con los diagramas de clases tradicionales, es que cuentan con un grupo de estereotipos de UML adicionales que son específicos para los diagramas de clases web. Estos permiten modelar aplicaciones con esta arquitectura como se muestra en la Figura 8. En la imagen se observan las clases correspondientes al módulo desarrollado. Las clases Queue, TripletExtraction y Parser dependen de la clase controladora Tripleta. A continuación, se describe el funcionamiento de cada clase.

Tripleta: Tiene la función de gestionar todos los componentes de cada tripleta y al ser la clase controladora

realiza llamada a todos los métodos del proyecto.

Parser: Realiza el algoritmo básico de la extracción de tripleta y la encargada de importar las librerías para identificar cada componente de la oración.

Queue: Tiene como función crear la estructura de cada tripleta.

TripletExtraction: Se encarga de realizar la extracción de los documentos indexados en Solr por tanto es la clase que realiza la comunicación del módulo con el indexador.

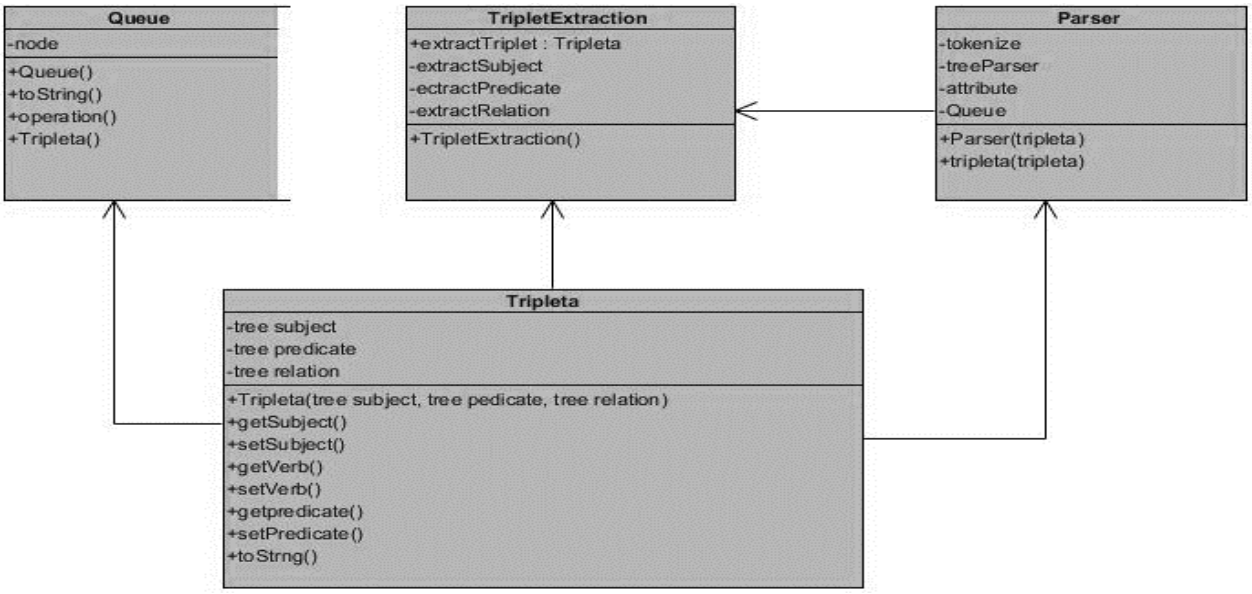

Figura 8 Diagrama de clases de diseño.

2.8 Modelo de Despliegue

El diagrama de despliegue permite modelar la disposición física o topología de un sistema. Muestra el hardware usado y los componentes instalados en el hardware, además de las conexiones físicas entre este y las relaciones entre componentes. Es utilizado para capturar los elementos de configuración del procesamiento y las conexiones entre esos elementos. También se utiliza para visualizar la distribución de los componentes de software en los nodos físicos. El mismo está compuesto por: nodos, dispositivos y conectores (Zamora, 2013).

El modelo de despliegue que se presenta a continuación muestra el nodo “Dispositivo_Cliente”, representado por un nodo ordenador el cual contiene un navegador para Internet. Este se encarga de

comunicarse a través del protocolo HTTP con el nodo “Servidor de aplicaciones web Apache Tomcat” y este a su vez con el “Servidor de Bases de Datos Solr” a través del protocolo HTTPS, donde se encuentran todos los documentos indexados, como se observa en la Figura 9.

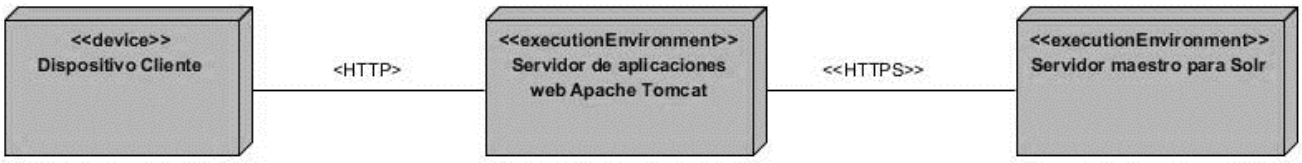

Figura 9 Modelo de despliegue

2.9 Conclusiones parciales

Con la realización de análisis y diseño del módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión se puede concluir que:

- ✓ El módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión como propuesta de solución, está basado en cuatro procesos fundamentales. Se implementa un prototipo con el fin de contribuir al procesamiento semántico de los documentos indexados en la base de datos de Solr.
- ✓ La conformación de los Diagramas de Clases del diseño permitió definir las relaciones entre las clases del módulo permitiendo visualizar la relación entre las mismas, así como las funcionalidades y atributos que deben presentar cada una de estas.
- ✓ El Modelo de Dominio realizado a partir de los procesos identificados permitió conocer todos los términos y conceptos presentes en el entorno.
- ✓ La conformación de los Diagramas de Clases del diseño permitió definir las relaciones entre las clases del módulo permitiendo visualizar la relación entre las mismas, así como las funcionalidades y atributos que deben presentar cada una de estas.
- ✓ La especificación de los requisitos funcionales y no funcionales del módulo, dieron paso a una mejor comprensión, por parte de la autora, de los resultados que se pretenden obtener de una manera precisa y sirvieron de guía para la implementación del módulo.
- ✓ La definición de la arquitectura y los patrones de diseño a utilizar, permitieron establecer las bases para fomentar la reutilización y las buenas prácticas de programación durante la fase de implementación, así como disminuir el impacto de los cambios futuros en el código fuente.

- ✓ La elaboración del diagrama de despliegue permitió identificar la disposición física de los artefactos del módulo a desarrollarse.

Capítulo 3: Implementación y validación del Módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión.

3.1 Introducción

La fase de implementación en el desarrollo de un producto de software, es el mecanismo donde se ponen en práctica todas las descripciones y arquitecturas propuestas en las fases de análisis y diseño, es el complemento del trabajo de las fases que lo preceden dentro del proceso de desarrollo de software. La implementación ofrece una materialización precisa de los requisitos.

Una de las últimas fases del ciclo de vida antes de entregar un programa para su explotación es la fase de pruebas, cuyo objetivo es comprobar si este cumple sus requisitos. Dentro de ella pueden desarrollarse varios tipos de pruebas en función de los objetivos de las mismas.

3.2 Modelo de componentes que integran la solución informática

El modelo de componentes representa la forma en que es estructurado un sistema informático atendiendo a las diferentes partes que lo componen. Partiendo de este punto, (Sommerville, 2005) puntualiza que cada componente debe ser tratado como una unidad de composición independiente e indispensable dentro de un sistema y que puede contraer relaciones de dependencia con otros componentes. Algunos ejemplos de componentes físicos lo constituyen los archivos, módulos, librerías, ejecutables, binarios, entre otros.

3.2.1 Diagrama de componentes

El Diagrama de componentes describe cómo se implementan las clases en término de componentes, como pueden ser ficheros de código fuente, ejecutables, entre otros. Describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponible en el entorno de implementación y en el lenguaje de programación utilizado. Además, muestra las dependencias entre componentes (Toro, 2016).

A continuación, se muestra el diagrama de componentes propuesto, el cual está acorde con los requerimientos del módulo a implementar y con el patrón arquitectónico basado en componente. Como se observa en la Figura 10. El diagrama cuenta con la aplicación como componente principal, de él depende el componente *triplets* en el cual se gestionan todas las tripletas. Además, el componente principal está integrado un paquete de librerías divididas en dos secciones, una son las librerías Stanford que contienen todos los paquetes del analizador y las otras son las Solrj que integran al indexador Solr con el *plugin* implementado. Se puede observar que el componente principal está integrado a Solr y este a su vez al

motor de búsqueda Orión.

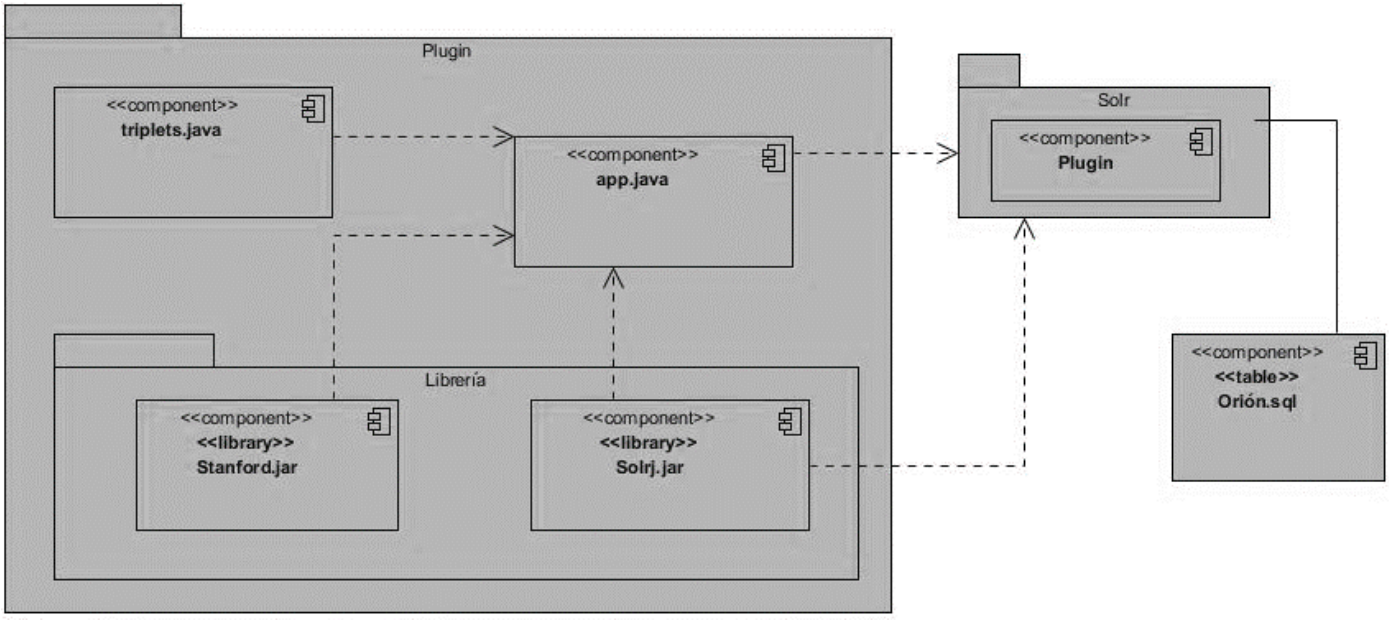

Figura 10 Diagrama de componentes correspondiente al plugin de Solr.

3.3 Estándares de codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez.

Para facilitar el entendimiento del código y fijar un modelo a seguir, se establecieron estándares de codificación. Los aspectos para los que generalmente se establecen estándares son los siguientes.

3.3.1 Nomenclatura según el tipo de clases

Clases de dominio: las clases que se encuentran dentro de la carpeta *domain* y al igual que las clases controladoras siguen la notación *UpperCamelCase*.

Ejemplo:

```
public class TripletExtraction {
```

Figura 11 Ejemplo de estándar UpperCamelCase.

3.3.2 Nomenclatura de las funcionalidades y atributos

El nombre a emplear para las funciones y los atributos se escriben con la inicial minúscula, en caso de que sea un nombre compuesto se empleará la notación *CamelCase.*

Ejemplo de función: parse(). El nombre de este método está compuesto por una sola palabra, debido a esto es que se escribe con minúscula, si fuera un nombre compuesto por más de una palabra se procede a aplicar la nomenclatura antes mencionada.

Ejemplo de atributo: El nombre del atributo está compuesto por dos palabras, la primera en minúscula y la segunda iniciando con letra mayúscula como se muestra en la siguiente imagen.

```
private static TregexPattern adjPhraseTregexPattern = TregexPattern
```

Figura 12 Ejemplo de estándar CamelCase.

3.3.3 Nomenclatura de los comentarios

Los comentarios deben ser lo suficientemente claros y precisos de forma tal que se entienda el propósito de lo que se está desarrollando. Su uso ayuda a una mejor comprensión del código por parte de terceros para futuras adaptaciones. A continuación, un ejemplo de cómo se emplean en el código de la propuesta de solución la nomenclatura de los comentarios:

```
// helper linked list class
```

Figura 13 Ejemplo de estándar de comentarios.

3.4 Validación del módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión.

A continuación, se detallan los tipos de pruebas de software aplicados al módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión implementado. Las mismas persiguen como objetivo fundamental, la detección de las no conformidades respecto a las funcionalidades de la aplicación, las vulnerabilidades que atentan contra la seguridad de la información que se manipula con el software, la medición del grado de usabilidad de las funcionalidades implementadas, así como también la correcta integración entre los diferentes componentes de la arquitectura del módulo.

3.4.1 Pruebas funcionales

Las pruebas funcionales son aquellas que se aplican a un software determinado, con el objetivo de validar que las funcionalidades implementadas funcionen de acuerdo a las especificaciones de los requisitos definidos con anterioridad. Para la ejecución de este tipo de pruebas, suelen emplearse dos métodos fundamentales: el método de Caja Blanca y el método de Caja Negra. El primero se centra en las pruebas al código de las aplicaciones; mientras que el segundo permite a los probadores enfocar su atención en el funcionamiento de la interfaz, a través del análisis de los datos de entrada y los de salida. En este epígrafe se exponen los aspectos concernientes a las pruebas funcionales realizadas utilizando el método de Caja Blanca.

Las pruebas de caja blanca, denominada a veces “prueba de caja de cristal”, a consideración de (Pressman 2005), es un método de diseño de casos de prueba que se basa en el minucioso examen de los detalles procedimentales.

Este tipo de prueba será aplicada a la estructura de control del diseño procedimental (código fuente), a través de la técnica del camino básico. La prueba del camino básico es una técnica de prueba de caja blanca propuesta por Tom McCabe, como expone (Pressman 2005). Esta técnica, permite obtener una medida de la complejidad lógica de un diseño y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Mediante su aplicación: se garantiza la ejecución por lo menos una vez de los caminos independientes de cada módulo, se ejercitan las decisiones lógicas y estructuras internas de datos para asegurar su validez.

Para ello se realizaron tres iteraciones de pruebas unitarias utilizando el Eclipse Luna. A continuación, se muestran las no conformidades detectadas durante su realización. Los restantes diagramas se ubican en el Anexo 3**.**

Iteración #1

Se realizó la prueba a las HU Identificar oraciones dado un documento y confeccionar las tripletas necesarias dado el sujeto, relación y predicado de una oración.

- ✓ No delimitaba correctamente las oraciones.
- ✓ Excepción cuando el documento se encontraba vacío.
- ✓ No mostraba todas las tripletas asociadas a una oración compuesta, sólo mostraba una tripleta.
- ✓ No identificaba el predicado cuando era más de una palabra.

- ✓ Mostraba una tripleta con sujeto omitido, verbo omitido y Predicado omitido cuando en realidad el documento estaba vacío y no debía mostrar nada.

Iteración #2

Se realizó la prueba a las HU Identificar el sintagma nominal sujeto e Identificar verbos de cada oración.

- ✓ No identificó el sujeto omitido.
- ✓ No mostraba el sintagma nominal sujeto completo
- ✓ Identificaba un solo verbo de la oración cuando en realidad la oración era compuesta y tenía dos formas verbales.
- ✓ No identificaba la forma verbal cuando estaba al principio de la oración.

Iteración #3

No se encontraron no conformidades.

Las no conformidades detectadas durante la realización de las pruebas unitarias fueron corregidas satisfactoriamente.

Resultados de las pruebas funcionales

Con el objetivo de probar el correcto funcionamiento de las funcionalidades del módulo se realizaron tres iteraciones de pruebas a la herramienta. En la Tabla 10 se muestran los resultados obtenidos en cada iteración de prueba al módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión, así como la corrección de cada uno de los errores.

Tabla 8 Cantidad de no conformidades por cada iteración de las pruebas funcionales.

No conformidades	1era iteración	2da iteración	3era iteración
Detectadas	5	4	0
Resueltas	5	4	0
Pendientes	0	0	0

3.4.2 Pruebas de integración

Las pruebas de integración son definidas para verificar el correcto ensamblaje entre los distintos módulos

que conforman un sistema informático. Las mismas validan que estos componentes realmente funcionan juntos, son llamados correctamente, además, transfieren los datos correctos en el tiempo preciso y por las vías de comunicación establecidas (Sommerville, 2005).
Una vez realizadas las pruebas funcionales Posterior a estas pruebas, se hace necesaria la realización de pruebas de integración, con la finalidad de validar la correcta integración entre el módulo implementado y el mecanismo indexador Solr.

Para la realización de las pruebas de integración se llevaron a cabo diferentes acciones, a continuación, se mencionan:

- ✓ Verificación de la conexión del Módulo para la extracción de tripletas y Solr para indexar las tripletas extraídas del documento.
- ✓ Comprobar el correcto funcionamiento del motor de búsqueda Orión con el módulo incorporado.

Para realizar las pruebas de integración del módulo implementado con el mecanismo indexador Solr se realizaron 4 iteraciones.

Iteración 1:

- ✓ No se lograba realizar la conexión del módulo con el mecanismo indexador Solr.
- ✓ El módulo no lograba extraer todos los documentos a los que tenía que realizarle el procesamiento.
- ✓ No mostraba el campo *triplets* en el mecanismo indexador Solr.

Iteración 2:

- ✓ El campo *triplets* en Solr se mostraba vacío cuando en realidad, el módulo si había extraído tripletas.
- ✓ El campo *triplets* no mostraba el formato correcto de las tripletas.

En las primeras iteraciones se detectaron errores que fueron solucionados. Ya en la última iteración no se detectó ningún error, como se muestra en la siguiente tabla:

Tabla 9 Cantidad de no conformidades por cada iteración de las pruebas de integración.

No conformidades	1era iteración	2da iteración	3era iteración
Detectadas	3	2	0
Resueltas	3	2	0
Pendientes	0	0	0

La ejecución de las pruebas de integración permitió verificar el trabajo conjunto de los componentes de la herramienta en cuestión. Se hizo énfasis en la integración entre el motor de indexación Solr y el módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión, para detectar incoherencias en el funcionamiento de la aplicación, no encontrándose ninguna no conformidad, llegándose a la conclusión de que existe una correcta integración entre los componentes internos de la herramienta.

3.4.3 Validación de la hipótesis de la investigación.

Con el propósito de medir la contribución del Módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión, se definieron las principales funcionalidades del módulo correspondientes a la variable dependiente definida.

Como parte de la hipótesis de la investigación, se realizó una encuesta a 8 profesionales en la rama de la lingüística que pudiesen validar la calidad de la tripleta que extrae el módulo. De ellos uno es doctor, dos son máster y cinco licenciados, todos con más de 10 años de profesión, ver Anexo 4. El método utilizado fue el criterio de experto y la técnica puesta en práctica fue la escala Likert.

La escala Likert es una de las formas más utilizadas (y confiables) para hacerlo. La escala Likert mide las actitudes y los comportamientos utilizando opciones de respuestas que van de un extremo a otro (por ejemplo, muy improbable a extremadamente probable). A diferencia de las preguntas simples con respuesta sí/no, la escala Likert le permite descubrir distintos niveles de opinión, lo que puede resultar particularmente útil para temas o asuntos delicados o desafiantes. Contar con un rango de respuestas también le permitirá identificar fácilmente las áreas de mejora, Independientemente de que esté enviando un cuestionario para comprender los niveles de eficacia del curso que está dictando o recogiendo las opiniones de sus clientes. (Luna, 2012). Las preguntas realizadas fueron de acuerdo a la calidad las funcionalidades del módulo son

las siguientes:

Tabla 10 Preguntas realizadas en el cuestionario.

Preguntas realizadas en el cuestionario
P1 ¿El módulo identifica las oraciones de un documento correctamente?
P2 ¿El módulo identifica el sujeto correctamente de acuerdo a la estructura antes planteada?
P3 ¿El módulo identifica las formas verbales de cada oración correctamente?
P4 ¿El módulo identifica el predicado de cada oración correctamente?
P5 ¿El módulo confecciona la tripleta correctamente?
P6 ¿El módulo muestra en el campo *triplets* todas las tripletas asociadas a cada documento?

En la tabla siguiente se representan los resultados generales obtenidos por cada cuestionario.

Tabla 11 Resultados generales del cuestionario realizado

Preguntas	P1	P2	P3	P4	P5	P6	TOTAL
ESCALA	ni	ni	ni	ni	ni	ni	
TD	0	0	0	0	0	0	0
ED	0	0	0	0	0	0	0
NI	0	3	2	2	0	0	7
DA	0	3	3	4	2	3	15
TA	8	2	3	2	6	5	26
Total	8	8	8	8	8	8	48

Leyenda:

- ✓ TD - Totalmente en desacuerdo.
- ✓ ED - En desacuerdo.
- ✓ NI - Ni de acuerdo ni en desacuerdo.
- ✓ DA - De acuerdo.
- ✓ TA - totalmente de acuerdo.
- ✓ P- Pregunta.

A partir de lo anteriormente explicado, se evidencia que utilizando el módulo desarrollado se obtienen resultados con mayor calidad de relevancia para el usuario con respecto a Orión, siendo esto un resultado satisfactorio que apoya la hipótesis de la presente investigación.

3.5 Conclusiones parciales

En este capítulo se abordaron una serie de aspectos correspondientes a la implementación y validación del módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión llegándose a la siguiente conclusión:

- ✓ La representación y descripción del diagrama de componentes permitió visualizar con más facilidad la estructura general del módulo.

- ✓ La ejecución de pruebas al módulo permitió detectar las deficiencias presentes, subsanarlas en el menor tiempo posible y ofrecer una aplicación con mayor calidad.

- ✓ La realización de las encuestas a profesionales en la rama de la lingüística aportó elementos sustanciales que permitieron validar la hipótesis de la investigación planteada con anterioridad y con ello la factibilidad del módulo implementado.

Conclusiones

Después de desarrollar el módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión, se arribó a las siguientes conclusiones:

- ✓ El estudio del estado del arte sobre los diferentes sistemas homólogos permitió detectar determinados elementos que sirvieron como punto de partida para el desarrollo del módulo.
- ✓ EL análisis de herramientas y tecnologías existentes para el desarrollo aportó la base tecnológica más acorde para el desarrollo del proyecto.
- ✓ La generación de artefactos y esquemas permitió definir la propuesta de solución, lo que posibilitó a su vez obtener una visión del funcionamiento y una mayor claridad a la hora de implementar el módulo.
- ✓ El módulo para la extracción de tripletas de documentos desarrollado permite la extracción de sujeto, forma verbal y predicado de cada oración que contiene cada documento que se encuentra indexado en la base de datos Solr.
- ✓ Las pruebas realizadas al módulo para la extracción de tripletas de documentos indexados por el motor de búsqueda Orión permitieron que se validara el comportamiento de manera correcta del módulo y detectar errores que fueron corregidos.
- ✓ La ejecución del método criterio de expertos empleando la técnica Likert, permitió validar la hipótesis planteada en la investigación.

Recomendaciones

Luego de haber concluido la investigación y el desarrollo de la propuesta de solución se recomienda:

- ✓ Mejorar la calidad de la extracción de tripletas de documentos indexados por el motor de búsqueda Orión para oraciones compuestas.
- ✓ Realizar una aplicación que permita realizar un completo procesamiento semántico de los documentos indexados por el motor de búsqueda Orión teniendo en cuenta el módulo implementado en la presente investigación.

Referencias

ABELA, Jaime Andréu. *Las técnicas de análisis de contenido: una revisión actualizada*. 2002**.**

AGUDELO, Andrea López, et al. *DISEÑO DE UN PROTOTIPO PARA LA PROGRAMACIÓN DETALLADA Y DESPACHO DE LA PRODUCCIÓN BASADO EN EL ESTÁNDAR ISA-95. REVISTA GTI*, 2015, vol. 13, no 37, p. 81-85.

BAEZA-YATES, R.; RIBEIRO-NETO, B. *Modern information retrieval.* New York: ACM Press; Harlow [etc.]: Addison-Wesley, 1999. ISBN 0-201-39829.

BANKO, MICHELE. *Open Information Extraction from the Web*. *En IJCAI. 2007*. p. 2670-2676.

BATANERO, Jaime; CAMILO, Cristian; BELTRÁN MUÑOZ, Diego Alejandro. *Diseño de un plan de mejoramiento del sistema de seguridad físico y lógico de acuerdo con modelos de gestión de la infraestructura de redes aplicable a una institución educativa*. 2013. Tesis de Licenciatura. Universidad Militar Nueva Granada.

BERNERS-LEE, TIM. *The semantic web*. Scientific american, 2001, vol. 284, no 5, [Citado el: 20 de Octubre de 2016.]. p. 28-37.

CARDONA, Alberto, et al. *Análisis numérico de diferentes criterios de similitud en algoritmos de clustering*. *Mecánica Computacional*, 2006, vol. 25, p. 993-1011.

CASTAÑEDA, Yenier, et al. *UMCC_DLSI_Prob: A Probabilistic Automata for Aspect Based Sentiment Analysis. SemEval 2014*, 2014, p. 722.

CERAN, Betul, et al. *A semantic triplet based story classifier. En Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*. IEEE Computer Society, 2012. p. 573-580.

CHOMSKY, Noam. *Una aproximación naturalista a la mente y al lenguaje*. 1998.

COLMENAREJO GARCÍA, Borja. *Adaptación del Stanford Parser al español*. 2016. Tesis de Licenciatura.

DADZIE, ABA-SAH; ROWE, MATTHEW. *Approaches to visualising linked data*: A survey. *Semantic Web*, 2011, vol. 2, no 2. p. 89-124.

MARNEFFE, MARIE-CATHERINE. *Generating typed dependency parses from phrase structure parses*. En *Proceedings of LREC*. 2006. p. 449-454.

DUMAS, MARLON. *Fundamentals of business process management*. Heidelberg: Springer, 2013.

ESTIGARRIBIA, HÉCTOR. *Revisión Bibliográfica*. *POSTGRADO*. . [En línea]. 2015, vol. 23, p. 10-2015.

ETZIONI, OREN; BANKO, MICHELE; CENTER, TURING. *The Tradeoffs Between Open and Traditional Relation Extraction*. En *ACL*. 2008. p. 28-36.

FUENTES, María del Carmen Gómez. *MATERIAL DIDÁCTICO NOTAS DEL CURSO ANÁLISIS DE REQUERIMIENTOS*. 2011.

GAMALLO, Pablo*. Dependency parsing with compression rules*. *IWPT 2015*, 2015, p. 107.

KUNA, Horacio Daniel, et al. *Avances en la Construcción de un Sistema de Recuperación de Información para Información Científica en Ciencias de la Computación. En XVIII Workshop de Investigadores en Ciencias de la Computación (WICC 2016, Entre Ríos, Argentina)*. 2016.

MASRANI, Atiqah Izzati; GOTOH, Yoshihiko. *Corpus Generation and Analysis: Incorporating Audio Data Towards Curbing Missing Information.*2015.

GARCÍA M, Jesús; ORTÍN, M. José; MOROS, Begoña. *De los Procesos del Negocio a los Casos de Uso*. 2007.

HERNÁNDEZ, ASUNCIÓN; KÜSTER, INÉS. *De la Web 2.0 a la Web 3.0: antecedentes y consecuencias de la actitud e intención de uso de las redes sociales en la web semántica*. Universia Business Review, 2013, vol. 1, no 37.

HERRERA, A., G., L. *Modelos de Sistemas de Recuperación de Información Lingüística Difusa*. 2006.

LADO, RAQUEL TRILLO. *Introducción a la Recuperación de Información*. 2010.

LUNA, Sandra Margarita Maldonado. *Manual práctico para el diseño de la Escala Likert. Revista Xihmai*, 2012, vol. 2, no 4.

MACARIO, CG DO N. *Anotação semântica de dados geoespaciais para a agricultura*. 2010.

MARTÍNEZ, Aguillón, et al. *Automatización del desarrollo de aplicaciones web mediante el enfoque mda-mde*. 2014.

MARTÍNEZ, Layla Hirsh. *Intérprete y Entorno de Desarrollo para el Aprendizaje de Lenguajes de Programación Estructurada. En JIISIC*. 2008. p. 189-196.

MARTÍNEZ MÉNDEZ, FRANCISCO JAVIER. *Recuperación de información: modelos, sistemas y evaluación*. Murcia: 2004.

MONSALVE, E.; WERNECK, V.; LEITE, J. C. S. P. *Evolución de un Juego Educacional de Ingeniería de Software a través de Técnicas de Elicitación de Requisitos. En Proceedings of XIII Workshop on Requirements Engineering (WER'2010), Cuenca, Ecuador*. 2010. p. 12-23.

MONTOYA-SUÁREZ, Lina María; PULGARÍN-MEJÍA, Elizabeth. *Enseñanza en la Ingeniería de Software: Aproximación a un Estado del Arte. Lámpsakos*, 2013, no 10, p. 76-91.

QUIÑONES, Martha Elena Vargas; DE VEGA, *Luzángela Aldana. Calidad y servicio: conceptos y herramientas.* Ecoe Ediciones, 2015.

PADRÓ, Lluís. *Analizadores multilingües en freeling. Linguamática*, 2011, vol. 3, no 1, p. 13-20.

PEREZ-CARBALLO, JOSE; STRZALKOWSKI, TOMEK. *Natural language information retrieval: progress report*. Information Processing & Management, 2000, vol. 36, no 1, p. 155-178.

PERIÑÁN PASCUAL, José Carlos. *En defensa del procesamiento del lenguaje natural fundamentado en la lingüística teórica. Onomázein: Revista de Lingüística, Filología y Traducción*, 2012, no 26, p. 13-48.

POTENCIER, Fabien; ZANINOTTO, François. *Symfony, la guía definitiva. Symfony, la guia definitiva.[En línea][Citado el: 20 de noviembre de 2015.] http://librosweb. es/libro/symfony_1_4*, 2008.

PRESSMAN, R.S., 2005. *Ingeniería de Software, Sexta Edición*, Ed. S.l.: Mc Graw Hill, Mexico

PYYSALO, SAMPO. *Lexical adaptation of link grammar to the biomedical sublanguage: a comparative evaluation of three approaches*. BMC bioinformatics, 2006, vol. 7, no 3, p. 1.

REAL ACADEMIA ESPAÑOLA. *Diccionario de la lengua española*. [En línea]. Real Academia Española, 2016. [Citado el: 1 de noviembre de 2016.]. Disponible en: [http://lema.rae.es/drae/?val=informaci%C3%B3n].

REYNOSO, Carlos; KICILLOF, Nicolás. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft*. *Buenos Aires: Universidad de Buenos Aires*, 2004

RODRIGUEZ, JOSE IGNACIO. *Dynamic simulation of multi-body systems on Internet using CORBA, Java and XML*. Multibody System Dynamics, 2003, vol. 10, no 2, p. 177-199.

RODRIGUÉZ, T. *Metodología de desarrollo para la Actividad productiva de la UCI. Programa de Mejoras*, Universidad de las Ciencias. Cuidad da la Habana.

RUSU, DELIA. *Triplet extraction from sentences.* Proceedings of the 10th International *Multiconference* Information Society-IS. 2007. p. 8-12.

STUDER, RUDI; GRIMM, STEPHAN; ABECKER, ANDREA. *Semantic web services*. Springer, 2007.

SUDEEPTHI, G.; ANURADHA, G.; BABU, M. **SURENDRA PRASAD**. *A survey on semantic web search engine*. IJCSI International Journal of Computer Science Issues, 2012, vol. 9, no 2. [Citado el: 30 de Octubre de 2016.].

TEMPERLEY, DAVY, SLEATOR, DANIEL yL AFFERTY, JOHN. *Link Grammar.* 2016.

THE APACHE OPENNLP DEVELOPER. *Apache OpenNLP* [En línea]., 2014. [Citado el: 25 de noviembre de 2016] Disponible en: [http://tika.apache.org/].

THE APACHE SOFTWARE FOUNDATION-TIKA. *Apache Tika. [En línea]. Apache OpenNLP, 2016. Disponible en: [http://opennlp.apache.org/documentation.html]¨.*

THE APACHE SOFTWARE FOUNDATION-SOLR. *Apache Solr*. [En línea]. Apache Solr , 2014. [Citado el: 3 de Octubre de 2016.] Disponible en: [http://lucene.apache.org/solr/].

TOMASSETTI, FEDERICO; TORCHIANO, MARCO; BAZZANI, LORENZO. *MDD Adoption in a Small Company: Risk Management and Stakeholders'* Acceptance. 2013, vol. 19, no 2, p. 186-206.

TORDERA ILLESCAS, Juan Carlos. *Lingüística computacional y anáfora*. 2010.

TORO, A.; GÁLVEZ, J. G. *Especificación de requisitos de software: una mirada desde la revisión teórica de antecedentes. Entre Ciencia e Ingeniería, 2016,* no 19, p. 108-113.

TUPE, Cirley; CISNEROS, Juan. Evaluación y Selección de Framework de

VISUAL PARADIGM. *Visual Paradigm for UML - Software design tools for agile software development*. [En línea] 2014. [Citado el: 24 de noviembre de 2016.]. Disponible en: [http://www.visualparadigm.com/product/vpuml/].

Zamora, Yanai Rios. *Biblioteca. Diseño e implementación del Módulo de Manejo de Archivos para SIPP v2.0. [En línea] [Citado el: 24 de febrero de 2013.]* [Disponible en:http://bibliodoc.uci.cu/RDigitales/2013/enero/17/TD_05613_12.pdf.]

Anexos

Anexo #1

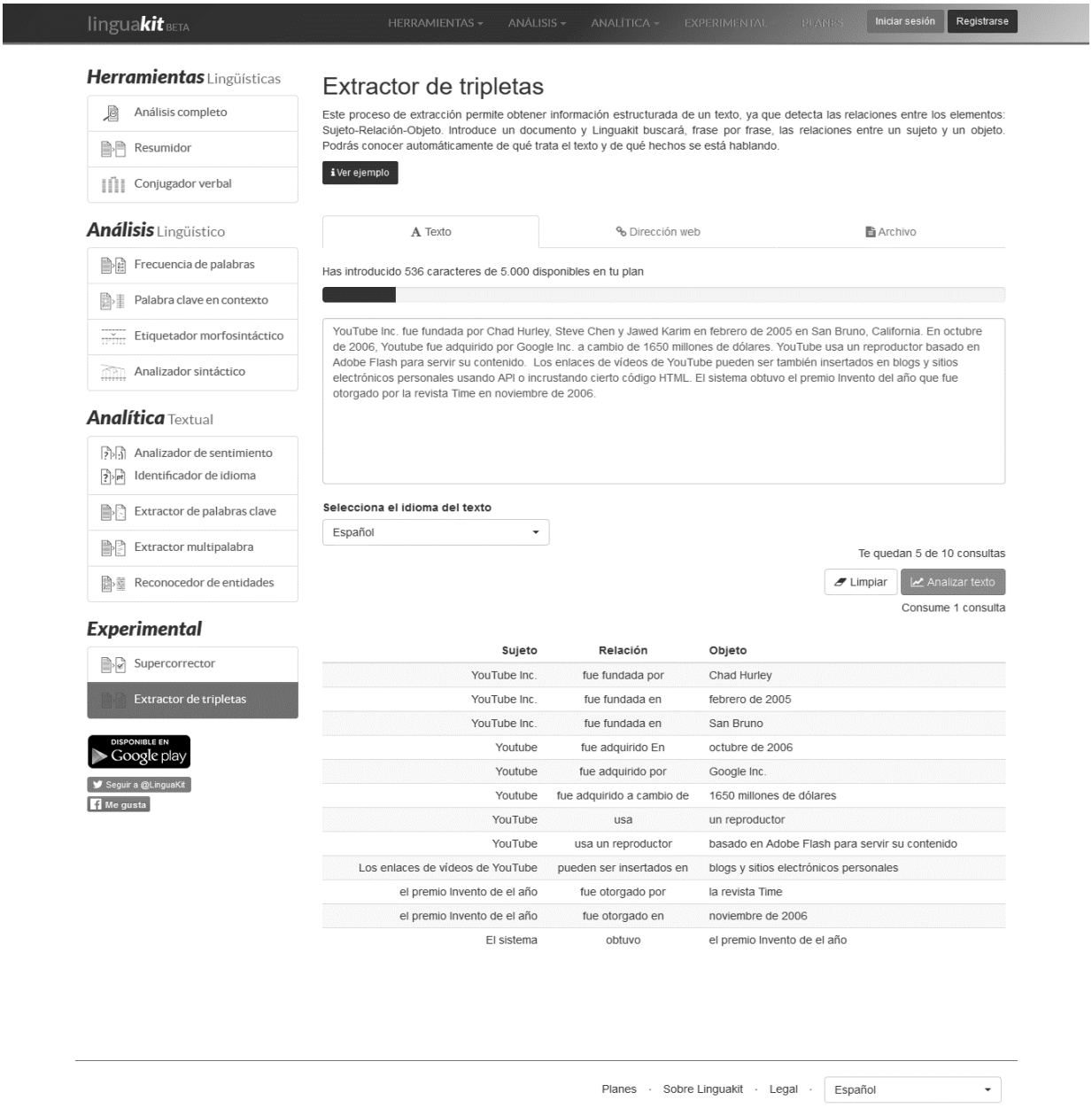

Figura 14 Sistema homologo Linguakit.

Anexo #2 Historias de usuario.

Tabla 12 Descripción de la Historia de usuario #1.

Historia de usuario	
Número: HU #01	**Nombre Historia de Usuario:** Identificar oraciones dado un documento.
Prioridad en negocio: Alta	
Descripción: Permite identificar las oraciones dado un documento. Primeramente, de todos los campos con que cuenta el indexador solr, el módulo solo utiliza el campo content donde se encuentra el contenido del documento. La funcionalidad comienza cuando el sistema identifica una oración en cada documento donde existe un punto.	

Tabla 13 Descripción de la Historia de usuario # 5.

Historia de usuario	
Número: HU #05	**Nombre Historia de Usuario:** Confeccionar las tripletas necesarias dado el sujeto, verbo y predicado de una oración.
Prioridad en negocio: Alta	
Descripción: Permite confeccionar la tripleta de una oración. Luego que el módulo identifique el sujeto, el verbo y el predicado de la oración, se confecciona la tripleta la cual tendrá esos tres campos.	

Tabla 14 Descripción de la Historia de usuario #6.

Historia de usuario	
Número: HU #06	**Nombre Historia de Usuario:** Añadir el campo triplet para cada documento en Solr.
Prioridad en negocio: Alta	
Descripción: Permite añadirle un nuevo campo para cada documento en Solr. La funcionalidad comienza cuando, luego de ejecutar el módulo, se actualiza el sistema indexador Solr y se muestra un nuevo campo nombrado triplet que contendrá todas las tripletas de cada documento.	

Tabla 15 Descripción de la Historia de usuario #7.

Historia de usuario	
Número: HU #07	**Nombre Historia de Usuario:** Mostrar todas las tripletas asociadas a un mismo documento en el campo *triplets*.
Prioridad en negocio: Alta	
Descripción: Permite mostrar todas las tripletas de un documento. Luego de ejecutarse el módulo, en el	

campo añadido *triplets* se mostrarán todas las tripletas que el módulo identificó, en forma de columnas y separadas por llaves.

Anexo # 3

```
// parser.Extraction(content);
Map<String,Object> fieldModifier = new HashMap<>();
for (int j = 0; j < lt.size(); j++) {
    for (int i = 0; i < lt.get(j).size(); i++) {
        id="Tripleta"+cont+"";
        if(j+1 == lt.size()&& i+1 <= lt.get(j).size()){
            t=lt.get(j).get(i).toString()+"";
        }
        else{
            t=lt.get(j).get(i).toString()+"; ";
        }


        fieldModifier.put("add", t);
        System.out.println("Tripleta "
                + lt.get(j).get(i).toString());
        cont++;
    }

}

resultDoc.addField("triplets", fieldModifier);// add the map as the field value
cont=1;
t="";

try {
    server.commit();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Figura 15 Integración del módulo implementado con el mecanismo indexador Solr.

```
private Queue<Tree> verbXtraction(Tree senteceTree) {
    Queue<Tree> list = new Queue<>();
    Tree[] children;
    // use Tregex to match
    TregexMatcher verbPhraseTregexMatcher = verbPhraseTregexPattern
            .matcher(senteceTree);
    while (verbPhraseTregexMatcher.find()) {

        children = verbPhraseTregexMatcher.getMatch().children();
        for (Tree child : children) {
            for (Tree child2 : child.children()) {
                list.enqueue(child2);
            }

        }
    }
    return list;
}
```

Figura 16 Identificar verbos de cada oración.

```
private Queue<Tree> nounXtraction(Tree senteceTree) {
    Queue<Tree> list = new Queue<>();
    Tree[] children;
    // use Tregex to match
    TregexMatcher nounPhraseTregexMatcher = nounPhraseTregexPattern
            .matcher(senteceTree);
    while (nounPhraseTregexMatcher.find()) {

        children = nounPhraseTregexMatcher.getMatch().children();
        for (Tree child : children) {
            for (Tree child2 : child.children()) {
                list.enqueue(child2);
            }

        }
    }
    return list;
}
```

Figura 17 Identificar sujeto de cada oración.

```
public Triplet(String subjectTree, String predicateTree, String objectTree) {
    this.subject = subjectTree;
    this.verb = predicateTree;
    this.predicate = objectTree;
}
public String getSubject() {
    return subject;
}
public void setSubject(String subject) {
    this.subject = subject;
}
public String getVerb() {
    return verb;
}
public void setVerb(String verb) {
    this.verb = verb;
}
public String getPredicate() {
    return predicate;
}
public void setPredicate(String predicate) {
    this.predicate = predicate;}
@Override
public String toString() {
    return  "{ sujeto: " + subject.toString() + " , relación: " + verb.toString() + " , predicado: " + predicate.toString()+"
}
```

Figura 18 Confeccionar las tripletas necesarias dado el sujeto, relación y predicado de una oración.

Anexo # 4

Tabla 16 Escala de Likert en relación a las probabilidades de respuesta.

5	4	6	2	1
Totalmente de acuerdo	De acuerdo	Ni de acuerdo ni en desacuerdo	En desacuerdo	Totalmente en Desacuerdo

Tabla 17 Cuestionario Likert # 1.

Cuestionario	Profesional no. 1

Afirmaciones	Alternativas de respuesta				
	1	2	3	4	5
El módulo identifica las oraciones de un documento correctamente.					X
El módulo identifica el sujeto correctamente de acuerdo a la estructura antes planteada.				X	
El módulo identifica las formas verbales de cada oración correctamente.					X
El módulo identifica el predicado de cada oración correctamente.					X
El módulo confecciona la tripleta correctamente.					X
El módulo muestra en el campo *triplets* todas las tripletas asociadas a cada documento.				x	

Tabla 18 Cuestionario Likert # 2.

Cuestionario	Profesional no. 2				
Afirmaciones	Alternativas de respuesta				
	1	2	3	4	5
El módulo identifica las oraciones de un documento correctamente.					x
El módulo identifica el sujeto correctamente de acuerdo a la estructura antes planteada.					x
El módulo identifica las formas verbales de cada oración correctamente.					x
El módulo identifica el predicado de cada oración correctamente.				x	
El módulo confecciona la tripleta correctamente.					x

El módulo muestra en el campo *triplets* todas las tripletas asociadas a cada documento.				x	

Tabla 19 Cuestionario Likert # 3.

Cuestionario	Profesional no. 3				
Afirmaciones	Alternativas de respuesta				
	1	2	3	4	5
El módulo identifica las oraciones de un documento correctamente.					x
El módulo identifica el sujeto correctamente de acuerdo a la estructura antes planteada.				x	
El módulo identifica las formas verbales de cada oración correctamente.					x
El módulo identifica el predicado de cada oración correctamente.				x	
El módulo confecciona la tripleta correctamente.					x
El módulo muestra en el campo *triplets* todas las tripletas asociadas a cada documento.					x

Tabla 20 Cuestionario Likert # 4

Cuestionario	Profesional no. 4				
Afirmaciones	Alternativas de respuesta				
	1	2	3	4	5
El módulo identifica las oraciones de un documento correctamente.					x
El módulo identifica el sujeto correctamente de acuerdo a la estructura antes planteada.				x	

El módulo identifica las formas verbales de cada oración correctamente.				x	
El módulo identifica el predicado de cada oración correctamente.					x
El módulo confecciona la tripleta correctamente.				x	
El módulo muestra en el campo *triplets* todas las tripletas asociadas a cada documento.					x

Tabla 21 Cuestionario Likert #.5

Cuestionario	Profesional no. 5				
Afirmaciones	Alternativas de respuesta				
	1	2	3	4	5
El módulo identifica las oraciones de un documento correctamente.					x
El módulo identifica el sujeto correctamente de acuerdo a la estructura antes planteada.					x
El módulo identifica las formas verbales de cada oración correctamente.				x	
El módulo identifica el predicado de cada oración correctamente.				x	
El módulo confecciona la tripleta correctamente.					x
El módulo muestra en el campo *triplets* todas las tripletas asociadas a cada documento.				x	

Tabla 22 Cuestionario Likert # 6.

Cuestionario	Profesional no. 6				
Afirmaciones	Alternativas de respuesta				
	1	2	3	4	5
El módulo identifica las oraciones de un documento correctamente.					x
El módulo identifica el sujeto correctamente de acuerdo a la estructura antes planteada.			x		
El módulo identifica las formas verbales de cada oración correctamente.				x	
El módulo identifica el predicado de cada oración correctamente.			x		
El módulo confecciona la tripleta correctamente.				x	
El módulo muestra en el campo *triplets* todas las tripletas asociadas a cada documento.					x

Tabla 23 Cuestionario Likert # 7.

Cuestionario	Profesional no. 7				
Afirmaciones	Alternativas de respuesta				
	1	2	3	4	5
El módulo identifica las oraciones de un documento correctamente.					x
El módulo identifica el sujeto correctamente de acuerdo a la estructura antes planteada.			x		
El módulo identifica las formas verbales de cada oración correctamente.			x		

El módulo identifica el predicado de cada oración correctamente.			x		
El módulo confecciona la tripleta correctamente.					x
El módulo muestra en el campo *triplets* todas las tripletas asociadas a cada documento.					x

Tabla 24 Cuestionario Likert # 8.

Cuestionario	Profesional no. 8				
Afirmaciones	Alternativas de respuesta				
	1	2	3	4	5
El módulo identifica las oraciones de un documento correctamente.					x
El módulo identifica el sujeto correctamente de acuerdo a la estructura antes planteada.			x		
El módulo identifica las formas verbales de cada oración correctamente.			x		
El módulo identifica el predicado de cada oración correctamente.				x	
El módulo confecciona la tripleta correctamente.					x
El módulo muestra en el campo *triplets* todas las tripletas asociadas a cada documento.					x