UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DISEÑO Y CONSTRUCCIÓN DE UN MÓDULO DE RECONOCIMIENTO DE CITAS BIBLIOGRÁFICAS Y SU INTEGRACIÓN EN EL SISTEMA DE ANÁLISIS DE ORIGINALIDAD DOCODE 2.0

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

YERKO VLADIMIR COVACEVICH VALDEBENITO

PROFESOR GUÍA:
SR. JUAN D. VELÁSQUEZ SILVA

MIEMBROS DE LA COMISIÓN:
SR. ERIC TANTER
SR. RODRIGO PAREDES MORALEDA

SANTIAGO DE CHILE
ABRIL 2013

Este trabajo ha sido parcialmente financiado por el proyecto CORFO 11IDL2-10399

Resumen

El objetivo general de esta memoria es diseñar, construir e integrar un módulo con algoritmos de detección de citas bibliográficas para el sistema de análisis de originalidad en documentos digitales DOCODE 1.0.

El sistema DOcument COpy DEtector (DOCODE), fue creado a partir de algoritmos que calculan distintos índices de similitud entre documentos digitales. Estos se integraron dentro de una arquitectura de sistemas, en la medida que fueron desarrollándose, sin una visión concreta de las necesidades que tendría la plataforma en el largo plazo. Este es uno de los problemas fundamentales a ser abordados en este trabajo.

Por otro lado, se detectó la necesidad de identificar citas bibliográficas, debido a que ellas no constituyen un intento de copia, sino una base para permitir conclusiones más elaboradas. Al contener texto citado, algunos índices se ven afectados en forma negativa.

Se planteó la hipótesis que, es posible mejorar aspectos estructurales de DOCODE 1.0 orientados principalmente a mejorar la calidad del servicio, modificando la arquitectura del sistema. Además, se proyectó que la posibilidad de detectar citas bibliográficas permite mejorar la evaluación de DOCODE respecto del análisis de originalidad.

La nueva versión de DOCODE 2.0 se construyó en una arquitectura orientada a servicios, lo que permite tener un esquema de servicios web completamente desacoplado, de alta disponibilidad y escalable. Por otro lado las investigaciones para el módulo de citas permitieron establecer un esquema de solución al problema.

Dentro de los resultados en las citas, las pruebas de laboratorio permitieron determinar los mejores patrones para identificar citas, para después probarlos con documentos reales. Los valores de las pruebas de laboratorio fueron: 0,9941(*Precision*), 0,9478(*Recall*) y 0,9704(*F-Measure*). Los valores para las pruebas en ambiente real fueron: 0,4101(*Precision*), 0,8302(*Recall*) y 0,5490(*F-Measure*).

En base a los cambios realizados, DOCODE puede prestar servicios para cualquier plataforma de software. Además se demostró que es posible parametrizar citas bibliográficas y construir una máquina que las detecte.

A Coté Ortiz.

Agradecimientos

- La persona que me ha apoyado durante casi toda nuestra vida, María José.
- Esos locos bajitos, Lucas e Ignacio.
- Todos los integrantes de *DocoWhale 2012*.
- Lingüistas poetas: Patricio Moya y Milena Araya, gracias por todo.
- Juan Velásquez, por toda la paciencia y sus enseñanzas de la vida.
- Mis amigos más cercanos y a los que el tiempo alejó, pero siguen estando a un click de distancia.
- Todos los que de una u otra forma ayudaron a cumplir este hito.

Yerko Vladimir Covacevich Valdebenito

Tabla de contenido

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

Toda obra literaria, artística, musical, científica o didáctica está protegida por el derecho de autor [10] y, por lo tanto, es importante mantener identificado el aporte de cada colaborador dentro de una obra.

En la actualidad, con la expansión de *la Web*, cada vez es más dificil poder garantizar los originales en el ámbito de los documentos digitales. DOcument COpy DEtector (DOCODE) es un sistema cuyo principal objetivo es determinar la originalidad de un documento, y de paso, detectar posibles plagios. DOCODE es una herramienta diseñada para prestar apoyo a los diversos actores educacionales, investigadores de universidades y otros, con el fin de permitir un efectivo control de la originalidad en documentos digitales, además de servir como herramienta de enseñanza de buenas prácticas sobre la confección de trabajos y documentos académicos.

Se puede hablar de plagio cuando una obra que se presenta como original, corresponde, en realidad, a una copia total o parcial de otra anteriormente publicada y sin el consentimiento explicito del autor. Esta infracción al derecho de autor (Ley 17.336 sobre propiedad intelectual en Chile) puede tener penas de presidio menor y multas que van desde 50 a 2.000 UTM [32].

Sin embargo, existe una importante excepción al analizar un documento y detectar texto que podría corresponder a un plagio: el uso de citas. Cuando se utilizan citas dentro de un documento, no se comete una infracción, puesto que se está respetando la creación de otros, lo cual da mayor seriedad al trabajo realizado.

1.1. Motivación

Surge de la necesidad del sistema DOCODE de identificar citas, ya que al no poseer esta funcionalidad, estas son marcadas como plagio, afectando los índices relativos al análisis de originalidad

de un documento que el sistema tiene implementado. En general, si bien existen algunas implementaciones de algoritmos de detección de citas, se han desarrollado en idiomas distintos al español y ninguna ha logrado resolver completamente el problema. Además, se hace necesario rehacer la arquitectura de DOCODE para orientarla a una arquitectura de servicios que permita la comercialización y una mejor administración de la plataforma.

1.2. Objetivos

1.2.1. Objetivo general

Diseñar, construir, probar e integrar un módulo con algoritmos de análisis y detección de citas bibliográficas para el sistema de análisis de originalidad DOCODE 2.0.

1.2.2. Objetivos específicos

1. Realizar un estado del arte referente a distintas técnicas para análisis y detección de citas.
2. Rediseñar e implementar una arquitectura de servicios para el sistema DOCODE 2.0.
3. Implementar un módulo con algoritmos de análisis y detección de citas.
4. Integrar el módulo con algoritmos de análisis y detección de citas en el sistema de detección de plagios DOCODE 2.0 utilizando una definición estándar.
5. Probar y validar el correcto funcionamiento de la solución aislada e integrada dentro del sistema DOCODE.

1.3. Metodología de investigación

1. Realizar estado del arte referente a distintas técnicas para análisis y detección de citas.
   a) Investigar acerca de técnicas de análisis y detección de citas.
   b) Estudiar de algoritmos de análisis y detección de citas.
   c) Escribir estado del arte.
2. Rediseño e implementación de la arquitectura de servicios del sistema DOCODE.
   a) Estudiar arquitectura de DOCODE para detectar vulnerabilidades y falencias.
   b) Rediseñar DOCODE para utilizar una arquitectura de servicios.
   c) Implementar las mejoras definidas.

3. Implementar módulo con algoritmos de análisis y detección de citas.

   a) Creación de base de datos de citas.

   b) Implementación de algoritmos.

   c) Evaluar rendimiento de los algoritmos con pruebas de entrenamiento semi-supervisado. Utilizando bases de datos de citas, se medirá estadísticamente usando "precision and recall" [2].

4. Integrar el módulo con algoritmos de análisis y detección de citas en el sistema de análisis de originalidad DOCODE 2.0 utilizando una definición estándar.

   a) Revisar diseño de sistema DOCODE.

   b) Definir un encapsulamiento estándar para los distintos algoritmos.

   c) Utilizar dicha definición sobre el algoritmo de citas.

   d) Verificar correcto funcionamiento de la implementación.

5. Probar y validar el correcto funcionamiento de la solución aislada e integrada.

   a) Ejecutar grupo de pruebas específicas al módulo de algoritmos de análisis y detección de citas.

   b) Ejecutar grupo de pruebas integradas.

   c) Implementar correcciones y ejecutar pruebas.

1.4. Contribución de la memoria

Hay muchos resultados que podrían resultar de la realización de esta memoria, de los cuales se mencionan los siguientes:

Redefinición de la arquitectura del proyecto DOCODE Esta memoria corresponde a la segunda parte del proyecto DOCODE y por lo tanto, debe hacerse cargo de corregir y mejorar los problemas y oportunidades de DOCODE 1.0. Es por esta razón que tiene una gran contribución en el desarrollo del proyecto, que se resume en los siguientes puntos:

- Aplicación de arquitectura JEE utilizando las mejores prácticas de desarrollo de software empresarial.
- Mejora de performance de la aplicación DOCODE.
- Rediseño de interfaces de presentación.

- Creación de plataforma de servicios.
- Paralelización de algunos procesos internos.

Diseño módulo de reconocimiento de citas bibliográficas El objetivo principal de investigación de esta memoria es el desarrollo e implementación de un módulo que permite identificar citas bibliográficas. Este módulo tiene mucha relevancia, ya que no existe todavía un algoritmo que permita la identificación de citas, por lo que esta investigación está en la frontera de la ciencia y podría llegar a representar un interesante campo de investigación a futuro. Algunas contribuciones que se esperan:

- Investigación sobre identificación de citas bibliográficas.
- Generación de un criterio estándar que permita clasificar citas bibliográficas.
- Implementación de algoritmos para detección de citas bibliográficas.
- Validación de los resultados mediante algún proceso automatizado que permita replicar las pruebas.
- Publicación de los resultados obtenidos.

1.5. Estructuración de la memoria

Este documento posee una estructura tal que permite comprender el trabajo realizado durante el desarrollo de la memoria. Además permite comprender cada una de las partes, ya que pretende estar autocontenido. La estructura del resto de este documento es:

- El Capítulo 2 establece una categorización para las citas, además fija conceptos importantes y define lo que es una arquitectura de servicios. Además presenta un estado del arte relacionado a citas bibliográficas.
- Dentro del Capítulo 3 se explica el funcionamiento de DOCODE 1.0. Además se muestran varios diagramas y procesos implementados dentro de esa solución, con el fin establecer el punto en que estaba el desarrollo de la solución y desde ahí presentar los cambios.
- El Capítulo 4 muestra todo el proceso de rediseño de la solución para pasar a DOCODE 2.0. También define en detalle el modelo de capas que sustenta la nueva arquitectura.
- Al interior del Capítulo 5 se encuentran los fundamentos de la implementación del modulo de detección de citas bibliográficas. Además se definen los experimentos y se muestran los resultados de ellos.

- Finalmente, el Capítulo 6 contiene en extenso las conclusiones del trabajo. Así también incluye aplicaciones, recomendaciones y posibles trabajos a futuro.

Capítulo 2

Marco Conceptual

A continuación, se procede a definir varios aspectos relevantes para el entendimiento de la memoria, se define una categorización de citas, se introducen conceptos relevantes, se presentan definiciones y un estado del arte. Todo es con el fin de establecer el contexto donde se sitúa esta investigación.

2.1. Categorización de las citas

Se entiende como cita a la utilización de contenido no original del autor y que presente alguna marca o haga referencia hacia la fuente original. De esta manera, se analizarán y clasificarán las citas de acuerdo a dos criterios de categorización independientes entre si: sintáctico y discursivo [23].

1. **Categorización sintáctica:** se refiere a las partes de la oración con su función sintáctica. En ella, es posible distinguir dos categorías de presentación de la información proveniente de otros estudios. Éstas son la cita directa y la cita indirecta [23].

   a) **Citas directas:** se reproducen de manera textual las palabras del autor. Ejemplo: Juan dijo: “Ponle prioridad 1”.

   b) **Citas indirectas:** se parafrasean las palabras del autor. Ejemplo: Juan dijo que le pusieras prioridad 1.

2. **Categorización discursiva:** este nivel de análisis se encuentra relacionado con el grado de relevancia que se le otorga al referente, manifiesto en su inclusión o no dentro del enunciado. Distinguimos, en este caso, dos categorías: integrales y no integrales [40].

   a) **Citas integrales:** se realiza una mención del autor de la información presentada dentro del enunciado. Ejemplo: Según Velásquez (2009) ocurrió de esa forma.

b) **Citas no integrales:** el autor aparece mencionado fuera del enunciado, ya sea mediante una nota al pie de página o entre paréntesis al finalizar la cita. Ejemplo: En ese sentido, tuvo mucha fortuna (Velásquez, 2008).

2.1.1. Categorización sintáctica: citas directas

Corresponde a la introducción de información ajena textual, la que se encuentra marcada tipográficamente (uso de comillas, cursiva, al margen, etc.) y precedida por algún mecanismo sintáctico/textual mediante el cual se identifica la procedencia o atribución de la información, y a la vez se relacionan explícitamente dos segmentos textuales [23]. Esta categoría se manifiesta de diferentes maneras, sin embargo, la prototípica se relaciona con el siguiente esquema: Nombre de persona (entidad) + verbo de reporte + cita (entre comillas o con alguna marca tipográfica, como negritas, cursivas o subrayado).

Ejemplo: Como señala María Elena Martínez, "el sistema de clasificación emergente contaba con la idea de que cada una de las tres principales categorías coloniales –españoles, indios y negros- se caracterizaba por una unidad o sustancia que era mantenida a través de la endogamia pero podía romperse por las relaciones sexuales fuera del grupo".

2.1.2. Categorización sintáctica: citas indirectas

Corresponde a la presentación de la información ajena de forma no textual, realizada a través de un verbo de reporte (indica, señala, plantea, establece, etc.) seguido de la conjunción subordinante 'que' [23]. Habitualmente, se parafrasean las palabras del autor. Al igual que la anterior, esta categoría se manifiesta de diferentes maneras, sin embargo, la prototípica se relaciona con el siguiente esquema: Nombre de persona (entidad) + verbo de reporte + que + cita (sin marca tipográfica).

Ejemplo: A la vez, Oates en 1972 plantea en su teorema de la descentralización, que no habiendo menores costos por una provisión centralizada de los bienes o efectos externos entre los niveles subnacionales, el bienestar siempre será igual o mayor si la provisión de bienes se realiza en cada región, en vez de ser único y uniforme para todas ellas.

2.1.3. Categorización discursiva: citas integrales

Se realiza una mención del autor de la información presentada dentro del enunciado. Ésta se puede dar con o sin la referencia al texto de donde se extrajo la cita, o bien puede aparecer de forma incompleta (por ejemplo, explicitar únicamente el título del texto) [23].

Ejemplo: Jaime Valenzuela se refiere a este tipo de grados, que ve relacionados en gran medida a la élite, en especial los grados de Maestre de Campo y General, monopolizados por los encomenderos.

2.1.4. Categorización discursiva: citas no integrales

El autor aparece mencionado fuera del enunciado, ya sea mediante una nota al pie de página o entre paréntesis al finalizar la cita. Tal como ocurre con las citas integrales, es posible encontrar casos donde la referencia esté ausente o incompleta.

Ejemplo: El GORE es un "órgano colegiado encargado de hacer efectiva la participación de la comunidad regional e investido con facultades normativas, resolutivas y fiscalizadoras"(GORE Valparaíso, 2008a).

2.2. Conceptos importantes

Arquitectura Se refiere a la organización fundamental de un sistema, representada en sus componentes, las relaciones con otros y el ambiente y los principios que gobiernan su diseño y evolución [24].

Corpus Un Corpus lingüístico es un conjunto, normalmente muy amplio, de ejemplos reales de uso de una lengua. Estos ejemplos pueden ser textos (típicamente), o muestras orales (normalmente transcritas).

Cita También llamada mecanismo de citación, es un fragmento de una expresión humana, a menudo escrita u oral, que ha sido insertada en otra expresión humana. Este último tipo de cita es la mayoría de veces tomada de la literatura, si bien los discursos transcritos, películas, canciones, literatura y otras fuentes también son comunes y válidos. Además, una cita puede también referirse al uso de una obra en otros trabajos artísticos - elementos de una pintura, escenas de una película o secciones de una composición musical dentro de otra [23].

JEE (Java Enterprise Edition) Es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java con arquitectura distribuida, basándose ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones. Dentro de esta implementación, se pueden destacar: EJB (Enterprise Java Beans), JSP (servicio de páginas), JMS (servicio de mensajes mediante colas), JNDI (directorio de nombres), RMI (invocación remota de objetos) y JDBC (conector de base de datos), entre otras [39]. La Figura 2.1 es un esquema simple de la arquitectura de Java EE. Más adelante se profundizará en este tema.

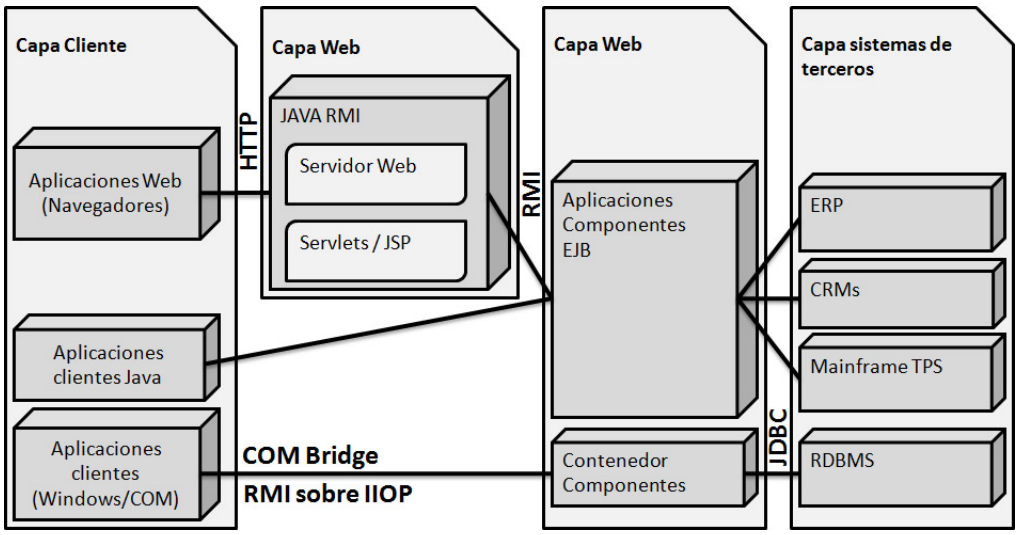

Figura 2.1: Arquitectura básica JEE.
Fuente: Basado en Sitio Web Oracle [39]

SaaS (Software as a Service) Es un modelo de distribución de software, mediante el cual se ofrece un servicio alojado en la compañia comercializadora y los clientes acceden a él mediante el uso de navegadores o utilizando clientes especializados. Esta modalidad permite a los clientes independencia respecto de los costos asociados a mantener complejos sistemas funcionando al interior de sus empresas.

SOA (Service Oriented Architecture) Concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requisitos del negocio. Permite la creación de sistemas altamente escalables que reflejan el negocio de la organización, brindando una forma estándar de exposición e invocación de servicios, lo cual facilita la interacción entre diferentes sistemas propios o de terceros [24].

SOAP (Simple Object Acccess Protocol) Es un protocolo que define como intercambiar datos mediante el uso de XML. La definición completa del protocolo se puede encontrar en [42]. Vale destacar que cada mensaje SOAP tiene 3 partes:

- Envelope: define qué hay en el mensaje y cómo se procesa.
- Header: conjunto de reglas que especifican el tipo de datos de cada parámetro.
- Body: donde se especifica una llamada a procedimientos y las respuestas de cada uno.

WS-Security (Seguridad en Servicios Web) Es un protocolo de comunicaciones que permite agregar seguridad en los servicios web. En él se especifica como debe garantizarse la integridad y seguridad en la mensajería. La definición completa está publicada por Oasis-Open en [26].

Si bien se critica la sobrecarga de WSS frente a otros protocolos como HTTPS, este garantiza que los paquetes transmitidos no sufren modificaciones con el mínimo impacto sobre la red.

2.3. Arquitectura Orientada a Servicios

Se decidió hacer cambios en la arquitectura de DOCODE para asegurar los aspectos más importantes del sistema. Está definido que DOCODE se comercializará como servicio en un esquema de *Software as a Service* (en inglés, SaaS), por esto se decidió utilizar una Arquitectura Orientada a Servicios (en inglés Service Oriented Arquitecture - SOA) ya que considera los patrones que cubren las necesidades del proyecto. Este tipo de arquitectura aborda los aspectos más importantes de una arquitectura de software empresarial.

Para implementar SOA, se utilizó un modelo de cuatro capas descrito por Panda en [31], el cual corresponde al modelo más utilizado para implementar SOA. En la Figura 2.2 se puede ver la posición relativa de cada una de ellas y su interacción.

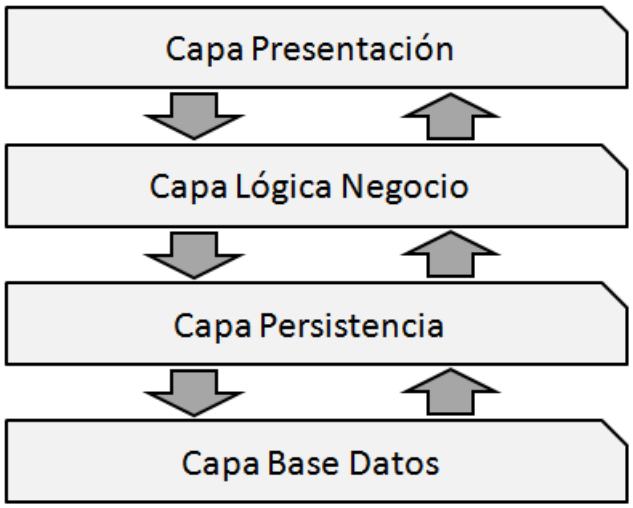

Figura 2.2: Diagrama de arquitectura de cuatro capas
Fuente: Basado en Panda et al. [31]

Este modelo de arquitectura es muy difundido debido principalmente a que es intuitivo. La capa de presentación es responsable de dibujar la interfaz gráfica del usuario (Graphical User Interface GUI) y manejar la entrada de datos del usuario. La capa de presentación toma los requerimientos para la aplicación de las lógicas disponibles en la capa de lógica de negocio. En esta capa están modeladas las distintas acciones o procesos que la aplicación puede ejecutar. Esta capa recupera y almacena los datos a través de la capa de persistencia, la cual provee una abstracción de alto

nivel orientada a objetos sobre el modelo de datos relacional de la capa de datos. La capa de datos tipicamente consiste en bases de datos relacionales implementadas en PostgreSQL o MySQL [31].

La arquitectura tradicional de cuatro capas tiene defectos. Una de las más importantes críticas es que se indetermina el modelo de objetos ideal al modelar el negocio como objetos que encapsulan datos y lógica. Esto debido a que la arquitectura tradicional se centra en modelar los procesos de negocio en vez del problema de negocio, de esta forma, la capa de lógica de negocio se parece más a una aplicación de procedimientos de base de datos que una aplicación con un modelo de objetos. Así, los componentes de la capa de persistencia pasan a ser simples administradores de datos que se parecen más a registros de la base de datos que objetos [31].

2.3.1. Justificación SOA

Como plantea Josuttis en [17], las principales características a considerar en un esquema SOA son las siguientes:

- **Servicios (Services)** Al generar abstracciones de la realidad, se busca dejar sólo los aspectos más importantes del problema a resolver. SOA se preocupa de abstraer los problemas de negocio, para crear servicios, los cuales en escencia representan una funcionalidad del negocio. Uno de los objetivos de SOA es estructurar grandes sistemas distribuidos basándose en las reglas de negocio y sus funciones. Una de las principales ventajas de hacer servicios es que la mayor parte de los problemas específicos de cada plataforma quedan en ella.

- **Interoperabilidad (Interoperability)** Al existir sistemas heterogéneos, es necesario tener una forma de conectar esos sistemas fácilmente. Si bien este no es un concepto nuevo creado para SOA, en esta arquitectura es un principio, no una consecuencia. Es la base desde donde se construyen los servicios que se extienden dentro de los sistemas distribuidos.

- **Desacoplamiento (Loose Coupling)** Este concepto corresponde a minimizar las dependencias entre los componentes de software. Cuando se minimiza el acoplamiento, el efecto de los cambios y de los errores también se reduce y los sistemas pueden seguir funcionando en una caída a pesar de perder parte de sus funcionalidades. También contribuye en buena medida a aumentar la tolerancia a fallos y la flexibilidad, además permite hacer diseños escalables. Todos los grandes sistemas sólo funcionan si el *Core Businness* se puede hacer de una manera descentralizada.

2.3.2. JEE como solución a la implementación

En DOCODE se decidió utilizar JEE como la plataforma para implementar la infraestructura de software. Ésto está basado en lo siguiente:

- **Rendimiento (Performance)** Desde el punto de vista del rendimiento, el uso de una plataforma JEE asegura que todos los recursos se utilizarán de la mejor forma posible, ya que el servidor de aplicaciones se encarga de la administración y optimización del uso de los recursos que tiene disponible en su instancia de ejecución.

- **Confiabilidad (Reliability)** Desde el punto de vista de la confiabilidad, la API de EJB3 garantiza que todos los procesos sean ejecutados de manera transaccional, permitiendo recuperar los procesos que no hayan concluido y terminar de procesarlos.

- **Escalabilidad (Escalability)** Desde el punto de vista de la escalabilidad, la interacción entre los componentes de la plataforma debe estar desacoplada. Ésto se logra utilizando *web services*, los que permiten tener servicios distribuidos en ambientes distintos y hacer que la ejecución ocurra paralelamente en múltiples servidores de aplicaciones.

- **Portabilidad (Portability)** Desde el punto de vista de la portabilidad, la utilización de JEE garantiza el funcionamiento sobre cualquier plataforma que tenga una máquina virtual JAVA y los servicios necesarios para levantar los servicios de la plataforma. Toda la especificación que define al estándar JEE está publicada, de forma que varios interesados puedan crear nuevos ambientes de desarrollo. Además, siguiendo varios estándares como RMI que permiten la ejecución remota de objetos, es posible ejecutar llamadas desde otros sistemas desarrollados en otros lenguajes.

- **Disponibilidad (Availability)** Desde el punto de vista de la disponibilidad, la capacidad de replicar los *web services* en distintas máquinas permite utilizar balanceadores de carga que administran las ejecuciones de modo de mantener disponible los servicios la mayor parte del tiempo.

2.3.3. JEE lógicas de servicios

Se utilizará un ESB, ya que como comenta Conner en [11], el uso de un ESB provee distintas características muy relevantes para la correcta utilización de SOA, las que se comentan a continuación.

Registro de servicios y hosting El ESB es el lugar donde se habilitan los servicios y son estos los que implementan las lógicas de negocio. Un registro es utilizado para buscar los *endpoints* de los servicios en tiempo de ejecución. Sin el registro, las aplicaciones clientes no son capaces de ubicar los servicios que están disponibles en el ESB.

Protocolos de traducción mediante adaptadores A fin de soportar una mayor cantidad de sistemas, el ESB es capaz de traducir los datos utilizando varios protocolos a un formato que puede ser transportado por el ESB, lo cual se hace mediante el uso de adaptadores (desde JMS, FTP, archivos, etc.) en mensajes dentro del ESB.

Orquestación de procesos Está la posibilidad de controlar múltiples procesos dentro de un proceso consolidado.

Administración de cambios El ESB permite cambiar las versiones "en caliente", soportando el uso de versiones, monitoreando y administrando los servicios desplegados mediante la consola. Esta funcionalidad es de vital importancia ya que los servicios en producción sufren cambios y actualizaciones, los que deben ser actualizados con el mínimo impacto sobre los clientes.

Calidad de servicio El ESB soporta transacciones de modo de asegurar que los servicios sean confiables.

Balance de carga Entre las distintas instancias de ejecución de servicios.

Seguridad Se habilitan los servicios en forma segura utilizando *WS-Security*.

Conjunto de acciones predefinidas En particular, Glassfish ESB permite crear un servicio personalizado de manera mucho más fácil utilizando un conjunto de acciones predefinidas, las que pueden ser incorporadas en el servicio. Estas acciones incluyen:

- Transformadores y convertidores: convierten mensajes de datos desde un formato a otro.
- Integración con Glassfish BPM.
- Scripting: existen tareas automáticas dentro de los lenguajes soportados.
- Servicios: permite la integración directa utilizando EJBs y otros servicios.
- Notifier: envío de datos hacia destinos incompatibles con un ESB.
- WebServices/SOAP: completo soporte para servicios web y servicios finales.

Un resumen de estas y otras características se ven en la Figura 2.3.

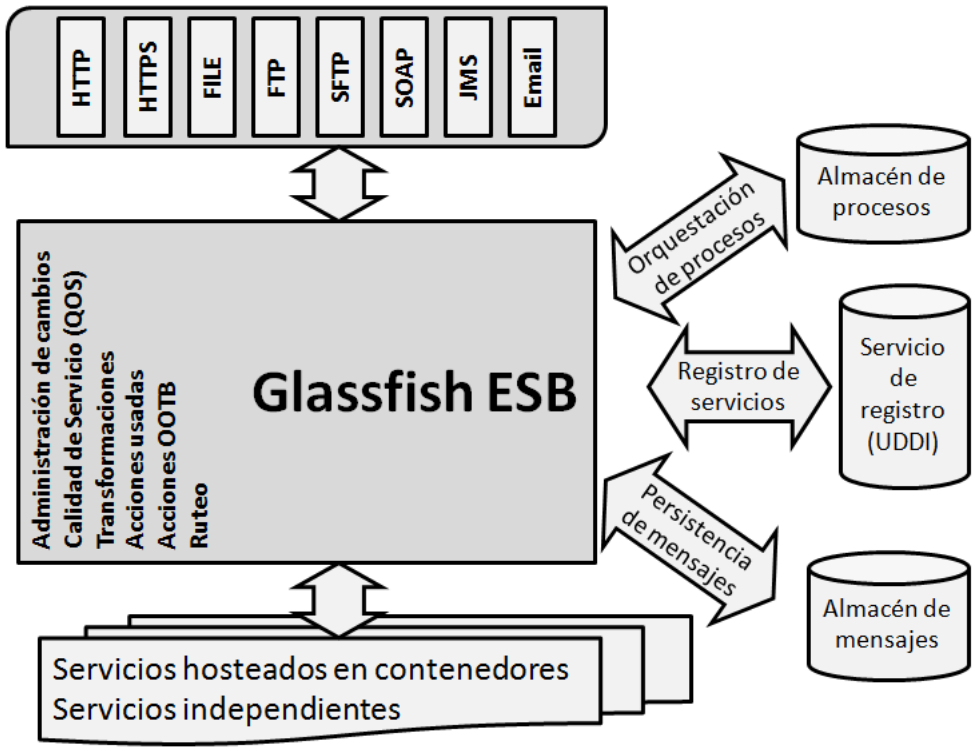

Figura 2.3: Principales caracteristicas del ESB.
Fuente: Conner, et al [11]

2.3.4. JEE lógicas de negocio

Una vez que se ha realizado el levantamiento del proceso de negocio, se debe pasar a la implementación. En JEE esta responsabilidad está asignada a los *Enterprise JavaBeans*. Actualmente, DOCODE está utilizando la definición 3.0 de esta API.

Enterprise JavaBeans (EJB) Los EJB brindan un modelo de componentes distribuidos del lado del servidor, cuyo principal objetivo es ofrecer una base al programador que permita abstraerse de los problemas clásicos de los ambientes empresariales, para centrarse en el desarrollo de las lógicas propias del negocio.

La especificación de EJB3 define ocho servicios básicos, sobre los cuales se trabaja para garantizar todo el funcionamiento del sistema, estos son:

Concurrencia Todas las llamadas dentro de un EJB son concurrentes, esto permite mantener varios hilos de ejecución independientes ya que todas las invocaciones son *full synchronized*.

Transacciones Todas las ejecuciones son transaccionales, esto significa que ante cualquier error o excepción, se ejecutan las reversas correspondientes a esa ejecución. Esto corresponde a una

API.

Persistencia Mediante el uso de la API de persistencia, es posible recuperar y guardar datos trabajando en un modelo de objetos independizándose del modelo relacional que existe en la base de datos.

Ejecución de Objetos Distribuidos Es posible coordinar la ejecución de objetos que están en distintos dominios ubicados a su vez en distintas máquinas usando sólo referencias locales, en términos prácticos, es como si los objetos estuvieran dentro de la misma instancia de la JVM.

Mensajería Asíncrona Con el uso de JMS es posible ejecutar procesos de manera asíncrona, los cuales además son tolerantes a fallas.

Servicio de Timers Permite coordinar la ejecución de procesos en tiempos futuros.

Espacio de Nombres Este servicio actúa como las páginas amarillas. En él se registran todos los servicios habilitados en el dominio, permitiendo acceder a ellos desde otros sistemas.

Seguridad Mediante el uso de la API de seguridad, se puede limitar la ejecución de algunos procesos según perfiles de usuario.

Existen 3 tipos de EJB:

Entity Beans Son los que encapsulan los datos en el lado del servidor y corresponden a la capa de persistencia.

Session Beans Son los que proveen las lógicas de negocio. Generalmente se utilizan para disponibilizar otros componentes cuya implementación está dentro del servidor. Existen de dos tipos:

Stateful En este tipo de bean, las variables de instancia se almacenan y se mantienen durante todo el tiempo, permitiendo modificar y posteriormente recuperar el estado de las variables hasta que el usuario termina la sesión. En caso que el usuario se desconecte involuntariamente, la instancia del bean es pasivada y puede ser activada cuando el usuario vuelve a conectarse, recuperando completamente el último estado de las variables.

Stateless En ellos, las variables de instancia no se mantienen, permitiendo el acceso concurrente a ellos. Sin embargo, no se garantiza el contenido de las variables de instancia entre llamadas al método, ya que son reutilizables por el contenedor de EJB.

Message-Driven Beans Estos beans se suscriben a las colas JMS o a un tema (o *topic*) y se instancian automáticamente cuando llega un mensaje, esto permite generar llamadas asíncronas dentro de los procesos.

2.3.5. JEE lógicas de persistencia de datos

Para implementar esta capa, se utilizaron dos interfaces distintas, JPA y JDBC.

JDBC (Java DataBase Connectivity) Es una API que permite la ejecución de operaciones sobre la base de datos desde el lenguaje Java, independiente del gestor de base de datos que se utilice. Si bien no se considera como una implementación de la API de persistencia, dentro de DOCODE existen algunas piezas que utilizan directamente JDBC como la capa que permite acceder a los datos y persistir los cambios. Esta forma de trabajo obliga a administrar el modelo de objetos como si fuera un modelo relacional y por lo tanto se pierde rendimiento en el uso de la base de datos y de la máquina virtual.

JPA (Java Persistence API) Esta API fue desarrollada especialmente para JEE y consisten en un framework que maneja datos relacionales dentro de las aplicaciones como si se tratara de un modelo de objetos. De esta forma, permite mantener las ventajas de la orientación a objetos al interactuar con una base de datos y permite usar objetos planos, conocidos como POJO (Plain Old Java Object).

Existen varias implementaciones de JPA, algunas de ellas:

- Hibernate
- TopLink
- EclipseLink
- OpenJPA
- Kodo

En JEE, estos objetos son llamados *Entity Beans*.

El uso de JPA, también ofrece la posibilidad de utilizar distintos lenguajes de consulta basados en objetos, lo que se diferencia de SQL (sigla de *Structured Query Language*) que trabaja sobre las relaciones del modelo.

EJBQL (EJB Query Language) Es un lenguaje basado en HQL (Hibernate Query Language) y fue modificado para usarse sobre EJB3 con una nueva definición llamada JPQL. Es similar a SQL, pero muy distinto en su modelo y objetivo, ya que utiliza un esquema abstracto orientado a objetos en vez del modelo relacional. Dicho de otra forma, las consultas en EJBQL no usan tablas ni sus componentes, en cambio usan EJBs, su estado de persistencia y las relaciones entre ellos. Así, el resultado de una consulta SQL es un grupo de registros de la base de datos, mientras que con EJBQL el resultado puede ser un objeto, una colección de objetos de un tipo dado, o un arreglo de datos.

JPQL (Java Persistence Query Language) Es usa para definir búsquedas contra los *Entity Beans* de forma independiente al mecanismo utilizado para guardar esas entidades. Así, JPQL es portable y está liberado de cualquier repositorio de datos. JPQL es una extensión de EJBQL, donde se agregan operaciones más complejas. Además las consultas en JPQL pueden ser declaradas en forma estática dentro de metadata o pueden construirse dinámicamente en el código.

2.3.6. JEE relación con los datos

JEE no implementa una capa de datos, ya que utiliza otros medios para este fin. Dependiendo del componente de DOCODE, es posible que se utilice un sistema de archivos (local o en red) o una base de datos.

Sistemas de archivos Está encargado de estructurar la información guardada en las unidades físicas (por ejemplo, disco duro) que luego es interpretada con el gestor de archivos. Para el caso de DOCODE, se hace uso del sistema de archivos mediante la creación de archivos temporales, los cuales están alojados dentro del file system de la misma máquina que se ejecuta o dentro del perteneciente a otra máquina configurada para ofrecer acceso en red.

Bases de datos Es el nombre utilizado para referirse generalmente a un gestor de base de datos (en inglés, Database Management System - DBMS) y a la base de datos misma, esto debido a que las bases de datos que se utilizan traen implementadas todas estas funcionalidades. El DBMS es la interfaz entre la base de datos, el usuario y las aplicaciones que lo utilizan, mientras que la base de datos es el conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su uso posterior.

2.4. Detección de citas bibliográficas usando herramientas informáticas

En primer lugar, Pouliquen et al. [33] desarrollaron un software que permite identificar citas directas de manera automática, en 11 idiomas diferentes provenientes de los cables de noticias. Trabajaron con un corpus compuesto por 35.000 noticias diarias. El método empleado consiste en rastrear, por un lado, marcas propias de cita que se encuentren cercanas a verbos de reporte (decir, declarar, señalar, entre otros) y, por otro, los nombres de persona conocidos que hayan mencionado lo citado. Dentro de una serie de posibilidades de citas directas, este software sólo es capaz de identificar aquellas en las que el nombre de la persona se encuentra inmediatamente antes o después de la cita. De esta manera, por ejemplo, no es capaz de identificar las citas que se relacionen a través de pronombres con su referente (manera anáforica). Los parámetros propuestos para identificar las citas son 6: marcas de cita, verbos de reporte, modificadores generales, determinantes, nombres de personas y entidades en general.

Posteriormente, de La Clergerie et al. [9] construyeron una plataforma de extracción de citas (denominada SAPIENS), tanto directas como indirectas, desde los cables de noticias (sólo en francés), que considera tanto al autor como a su contexto de aparición. El proceso de identificación y extracción de citas contempla tres etapas: la primera implica la detección de las entidades (nombres de personas, nombres de instituciones, URL, números) y la extracción de las citas literales; luego, se realiza un análisis sintáctico de los textos (a través de parsers); y, por último, una etapa que implica, tanto la extracción de citas, basadas en las citas literales y el resultado de los análisis sintáctico, como la resolución de anáforas. Junto con lo anterior, el equipo de investigadores desarrolló una página Web con el fin de poder mostrar al público las características de esta herramienta [14].

Con el objetivo de refinar los resultados de esta herramienta, Sagot et al. [38] elaboraron un lexicón de los verbos de cita (reported speech verbs) directa introducidos por una cláusula parentética. Una de las particularidades que se deduce de la investigación se relaciona con el método de búsqueda de las palabras, en la medida en que se presenta una mezcla entre el análisis sintáctico y el discursivo. Dicha decisión descansa en la certeza que poseen sobre el comportamiento léxico en el uso y no en la teoría. De acuerdo a las categorías propias de la gramática francesa, ciertos verbos de reporte son inherentemente transitivos, por lo que el análisis, a priori, supone la presencia obligatoria de un objeto directo. Sin embargo, gracias al análisis de una gran cantidad de noticias, concluyeron que no siempre, en la realidad del uso de la lengua por parte de los hablantes, algunos verbos, que se consideran como transitivos, se comportan como tales, en tanto los argumentos léxicos requeridos no

se presentan. Es más, proponen que todas las citas directas introducidas por una cláusula parentética más que construcciones sintácticas son construcciones discursivas. Desde esta misma perspectiva, Danlos et al. [7] realizaron un acabado análisis en el que determinaron y describieron 3 categorías en las que se dividen, discursivamente, los verbos de reporte en francés, con lo que mejorarían los resultados de detección y extracción de citas en los cables de noticia: verbos transitivos de estilo indirecto (decir, declarar), los verbos intransitivos (despotricar) y los verbos transitivos que no son de estilo indirecto (interrumpir, comentar, continuar).

Una de las tareas más complejas para detectar y extraer las citas se relaciona con identificar, fehacientemente, los verbos de reporte que introducen cita. Es así como, Krestel et al. [18] detallan las etapas básicas para el procesamiento de análisis del discurso referido y los informes sobre el rendimiento de una aplicación en forma de un recurso de GATE (entorno de procesamiento de lenguaje natural desarrollado por la Universidad de Sheffield). Su propuesta se sustenta en una concepción holística de lo que se entiende como una cita y la manera en que debe ser abordado este fenómeno (Ver Figura 2.4).

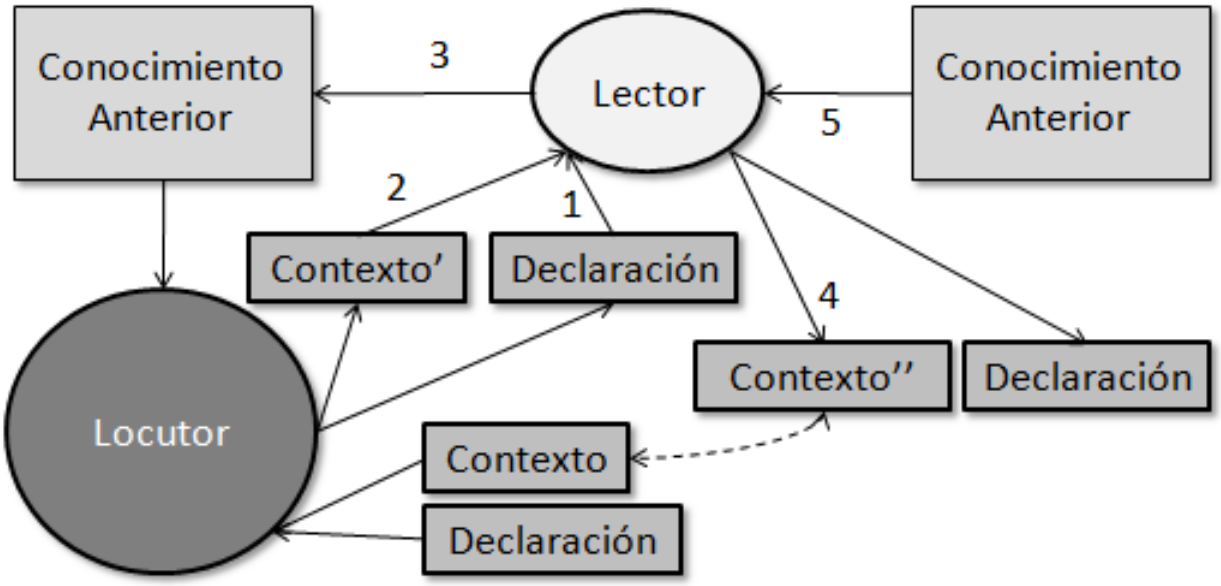

Figura 2.4: Etapa de análisis de verbos de reporte.
Fuente: Basado en Krestel et al. [18]

Esta implementación implicó la utilización de dos componentes. Por un lado, un marcador de verbos de reporte y, por otro, un detector de verbos de reporte. Los resultados obtenidos señalan que el recall alcanzó un valor de 0,79 mientras que el precisión value un 1,0, por lo que el F-measure fue de 0,88.

Uno de los aspectos que ha llamado más la atención de los investigadores en este campo es el del reconocimiento de las entidades (denominado como Named Entity Recognition) asociadas a las citas, en la medida en que su detección se constituye como un parámetro esencial para la identificación de citas. Leaman y Gonzalez [20] proponen BANNER, un sistema de código abierto para la detección de entidades en el área de la biomedicina. Está implementado en Java como un sistema de machine-learning basado en campos aleatorios condicionales (conditional random fields). Está diseñado para maximizar la independencia del dominio al no emplear rasgos semánticos (que pueden ser débiles) o basados en reglas, lo que la hace una herramienta utilizable, potencialmente, en cualquier dominio del conocimiento, con la sola condición de su especificación (Ver Figura 2.5).

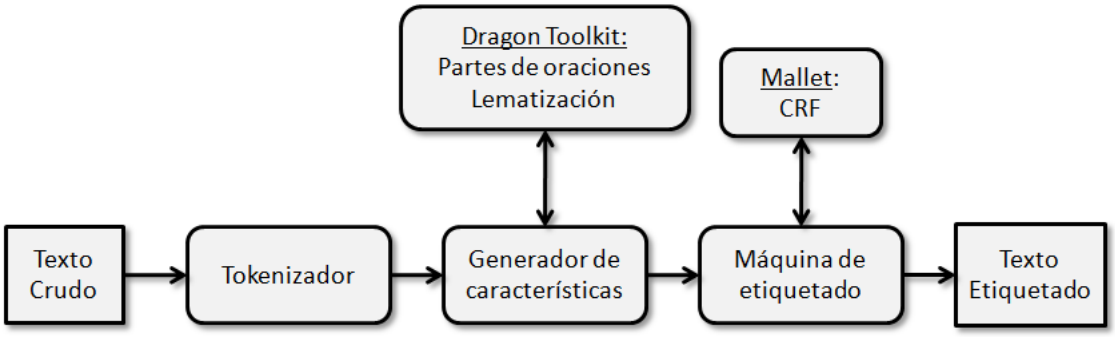

Figura 2.5: Arquitectura de BANNER.
Fuente: Gonzalez et al. [20]

Poulinquen et al. [34] presentan una herramienta que extrae los nombres de persona (nombres propios) desde largas colecciones de noticias en múltiples idiomas e iguala las variantes de nombres referidos a la misma persona del texto. Su principal preocupación es transliterar los nombres propios provenientes de lenguas que tienen diferentes alfabetos (árabe y latino, por ejemplo) (Ver Figura 2.6).

No obstante lo anterior, en muchas oportunidades, las entidades no se presentan como nombres sino que lo hacen por medio de estrategias correferenciales (por ejemplo, a través del uso de diversos pronombres). Bontcheva et al. [4] presentan una investigación en la que se discuten los métodos para la resolución de la correferencia poco profundas de las entidades nombradas y la construcción de las cadenas de correferencia, todo construido bajo la arquitectura de GATE. Los pasos propuestos para la identificación de la correferencia son 4:

1. Identificar el contexto del pronombre,

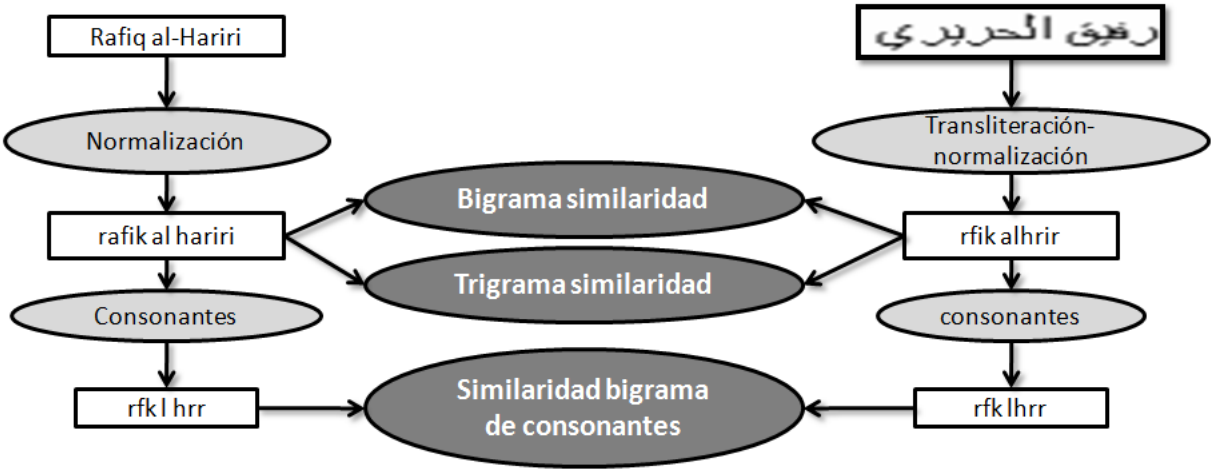

Figura 2.6: Proceso de normalización de nombres provenientes de la prensa árabe
Fuente: Steinberg et al. [34]

2. Inspeccionar el contexto de los antecedentes que satisfagan una serie de restricciones de consistencia,

3. Asignar valores a la prominencia cada antecedente sobre la base de un conjunto de reglas y de los factores

4. Elegir el candidato con el mejor valor de relevancia.

La puesta en práctica se basa únicamente en la información de parte de su discurso, denominado reconocimiento de entidades y la información de la correferencia ortográfica.

Dimitrov et al. [12] analizaron en su corpus las siguientes categorías de correferencia: personal (I, me, you, he, she, etc.), adjetivos posesivos (my, your, her, etc.), pronombres posesivos (mine, yours, hers, etc.), pronombres reflexivos (myself, yourself, herself, etc.) en relación con segmentos citados de texto. De esta manera, la construcción de la solución propuesta fue realizada a través de una estructura modular, siendo los más importantes el módulo de detección de citas y el módulo resolución de pronombres. El algoritmo propuesto se compone de 5 pasos:

1. Buscar la descripción del fragmento citado que contiene el pronombre. Si el pronombre no está contenido en cualquier fragmento, a continuación volver sin proponer un antecedente.

2. Inspeccionar el contexto del fragmento citado (como se define anteriormente) para determinar antecedentes candidatos.

3. Tratar de encontrar un candidato en el texto siguiente el fragmento citado (primer modelo). Si más de un candidato está presente, elegir el más cercano al final de la cita. Si un candidato se encuentra a continuación, elegir como un antecedente.

4. Tratar de encontrar candidato en el texto que precede el fragmento citado (tercer patrón). Elegir la más cercana al inicio de la cita. Si se encuentra, a continuación elija como antecedente.

5. Tratar de encontrar antecedentes en la parte sin comillas de la frase que precede a la frase en que la cita se inicia (segundo patrón). Dar preferencia a la pallabra más cercana al final de la cita (si lo hay) en la frase anterior o más cercana a la frase del principio.

Por último, desde un punto de vista computacional, algunas aproximaciones para la detección de citas han utilizado herramientas probabilísticas provenientes del área de la inteligencia artificial, como los Conditional Random Fields [37]. Estas aproximaciones, basadas en inferencia probabilística, permiten la extracción de citas directamente de las fuentes, modelando las oraciones y su similitud léxica. En una aproximación más reciente, se utiliza una variación de Markov Random Fields para la identificación de extractos citados dentro de un documento [35]. Estas técnicas son utilizadas frecuentemente para la generación automatizada de resúmenes de documentos y para la detección de plagio.

La revisión anteriormente planteada ratifica la necesidad de contar con un modulo de identificación y extracción de citas para refinar los resultados obtenidos en DOCODE. Además, plantea una problemática inmediata: no existen estudios relativos ni en el contexto de un detector de plagio (es decir, en su ámbito) ni para el idioma español (es decir, en su código). La gran parte de las investigaciones se ha llevado a cabo para el francés y para el inglés, lo que complejiza aún más la tarea, en la medida en que, por ejemplo, los sistemas pronominales (para efectuar la determinación de los correferentes) son completamente diferentes entre estos tres idiomas.

Capítulo 3

DOCODE 1.0 - Definición Original

El sistema DOCODE ha implementado varios algoritmos que permiten la detección de plagios [41], dentro de una arquitectura SOA [5], la que será rediseñada durante el presente trabajo de título, con el fin de que pueda cumplir con los requerimientos planteados. Esta implementación de DOCODE posee varias caracteristicas que se deben mejorar, de entre ellas: alto acoplamiento, diseño no escalable, problemas para usar esquemas de alta disponibilidad, alta dificultad para integrar nuevos algoritmos y la baja utilización de *web services* privilegiando el uso de *Servlets*, son las más críticas y que es necesario reparar para poder garantizar la calidad del servicio que presta DOCODE a sus usuarios.

3.1. Arquitectura original DOCODE

El proyecto DOCODE está distribuido en un conjunto de servidores, cada uno de los cuales permiten administrar las principales operaciones de DOCODE. La arquitectura tecnológica de DOCODE presenta una capa relacionada con el usuario final, una capa que permite soportar el procesamiento de los documentos, y otra capa que permite almacenar la información generada por el DOCODE Engine. Estos servidores están soportados por 2 unidades de almacenamiento y una UPS que permite mantener los servidores principales en operación continua ante imprevistos de la red eléctrica. La Figura 3.1 muestra la arquitectura física del sistema.

3.2. DOCODE SaaS

DOCODE Software as a Service (*DOCODE SaaS*) es el nombre del sistema que permite la interacción del usuario final con la herramienta de análisis de originalidad. Esta versión de la aplicación posee tres componentes principales:

- Una herramienta completamente desarrollada con el objetivo de proveer de un punto de interacción entre los distintos usuarios relacionados con DOCODE y el motor que permite buscar

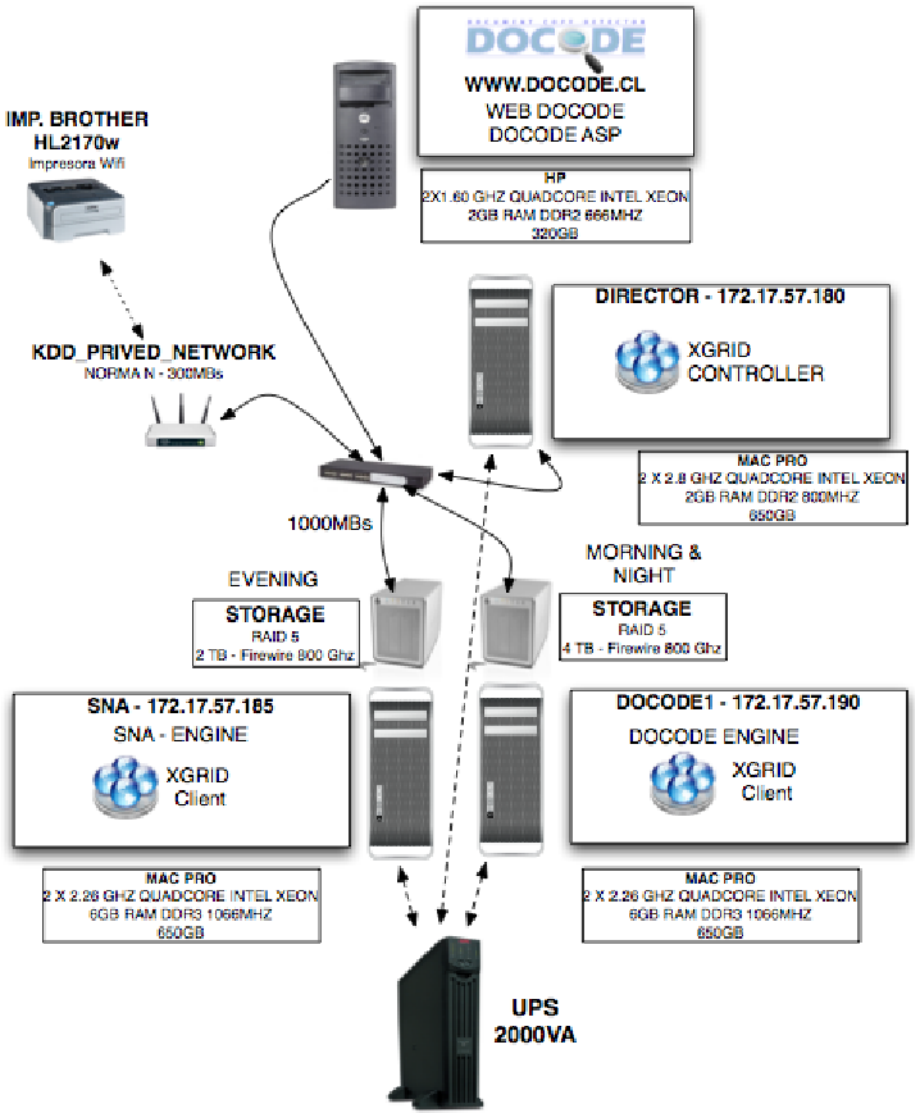

Figura 3.1: Arquitectura física DOCODE 1.0.
Fuente: Bravo et al. [5] (Uso autorizado por el autor)

la similitud entre los documentos, conocida también como *DOCODE Lite*.

- Un módulo para Moodle (herramienta de manejo de contenido escolar y universitario de e-Learning [22]). Mediante el cual es posible comunicarse con DOCODE Engine y entregar capacidad de análisis a los usuarios de esta plataforma.

- Una extensión de la plataforma U-Cursos [36]. Desarrollado por el Area de Infotecnologías de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile, la cuál es utilizada por un gran número de usuarios a lo largo de los distintos campus de la Universidad de Chile, y por otros establecimientos educacionales (e.g. colegios, institutos técnicos, etc.).

A continuación se presentan las principales funcionalidades que la herramienta DOCODE soporta para los usuarios de las distintas plataformas.

3.2.1. Procesos del sistema

DOCODE es un software orientado a centros educacionales y que permite la detección de copias en documentos tales como tareas, ensayos, resúmenes, etc. Para esto el software es capaz de desarrollar una serie de procedimientos agrupados en 2 proceso principales:

- **Proceso de administración** Corresponde a los procedimientos donde se realizan operaciones de mantenimiento y sustento operacional de los procesos y actores considerados en el software. Dentro de este proceso se encuentran la administración de roles y entidades y la recuperación de claves.

- **Proceso de generación/revisión de tareas** Corresponde a los procedimientos mediante los cuales se realiza generación y revisión de tareas haciendo el análisis de plagio. Al profesor se le permite la administración de tareas y asignación de estas a un grupo determinado, además de la posibilidad de descargar los archivos que representan las tareas, una vez cumplido el tiempo de entrega y un informe con el nivel de copia y plagio encontrado en el documento. Dentro de este proceso se encuentran la generación y asignación de tareas y la revisión de tareas.

Proceso de administración Este proceso es realizado por el super administrador y por el administrador del sistema, no siendo accesible para los perfiles de Profesores ni Alumnos. En este proceso se consideran los casos de uso de administración de roles y entidades en DOCODE y la recuperación de claves por parte de los usuarios del software. Se distinguen dos casos de uso en este proceso, el de "administración"y el de "recuperación de clave".

- **Administración de roles y entidades:** Los actores involucrados son el Administrador y el sistema, mediante este proceso son administrados tanto las instituciones como las personas asociadas a la institución administrada. La administración comprende la creación, actualización y eliminación de instituciones y personas en el sistema. En la Figura 3.2 se puede apreciar el diagrama BPMN del proceso de administración. Las principales opciones presentadas por el sistema son las de "Crear institución"y "Crear Persona", que son explicadas a continuación.

    - **Crear Institución:** Corresponde al procedimiento que permite la administración de las instituciones existentes en el sistema y la habilitación o deshabilitación de estas para su acceso a DOCODE, además de la creación de nuevas instituciones y la modificación de los datos de las instituciones ya creadas.
    - **Crear Persona:** Corresponde al procedimiento que permite la administración de personas dentro del sistema, creación, modificación y eliminación de los datos de personas.

- **Recuperación de Clave:** Los actores involucrados son el Administrador, el Profesor y el sistema. Mediante este proceso es posible restablecer una clave perdida. En la Figura 3.3 se puede apreciar el diagrama BPMN del proceso de recuperación de clave.

Proceso de generación y revisión de tarea Este proceso puede ser realizado por el super administrador, administrador y Profesor, y no es accesible a Alumnos, sin embargo los Alumnos participan en el proceso de manera indirecta con el sistema al recibir y contestar las tareas asignadas a las listas de cursos. El foco del proceso está puesto sobre el perfil de Profesor, ya que es él quien ejecuta el proceso. En el cual, se distinguen dos casos de uso distintos, el de "Generación y Asignación de tareas"y el de "Revisión de Tareas". A continuación se presentarán ambos casos utilizando diagramas BPMN.

- **Generación y Asignación de tareas:** Los actores involucrados son el profesor y el sistema. En este caso de uso se consideran las acciones concernientes a la confección de una tarea y a la asignación de esta a una lista de cursos específica. Cabe mencionar que en el profesor interactúa sólo con el sistema y en ningún momento directamente con el alumno. En la Figura 3.4 se presenta el diagrama de flujo del proceso de generación y administración de tareas de DOCODE. El sistema presenta como la opciones más relevantes: "Listas"y "Tareas", las que son explicadas a continuación.

    - **Listas:** Una lista es el equivalente a un curso, se representa como una lista con datos

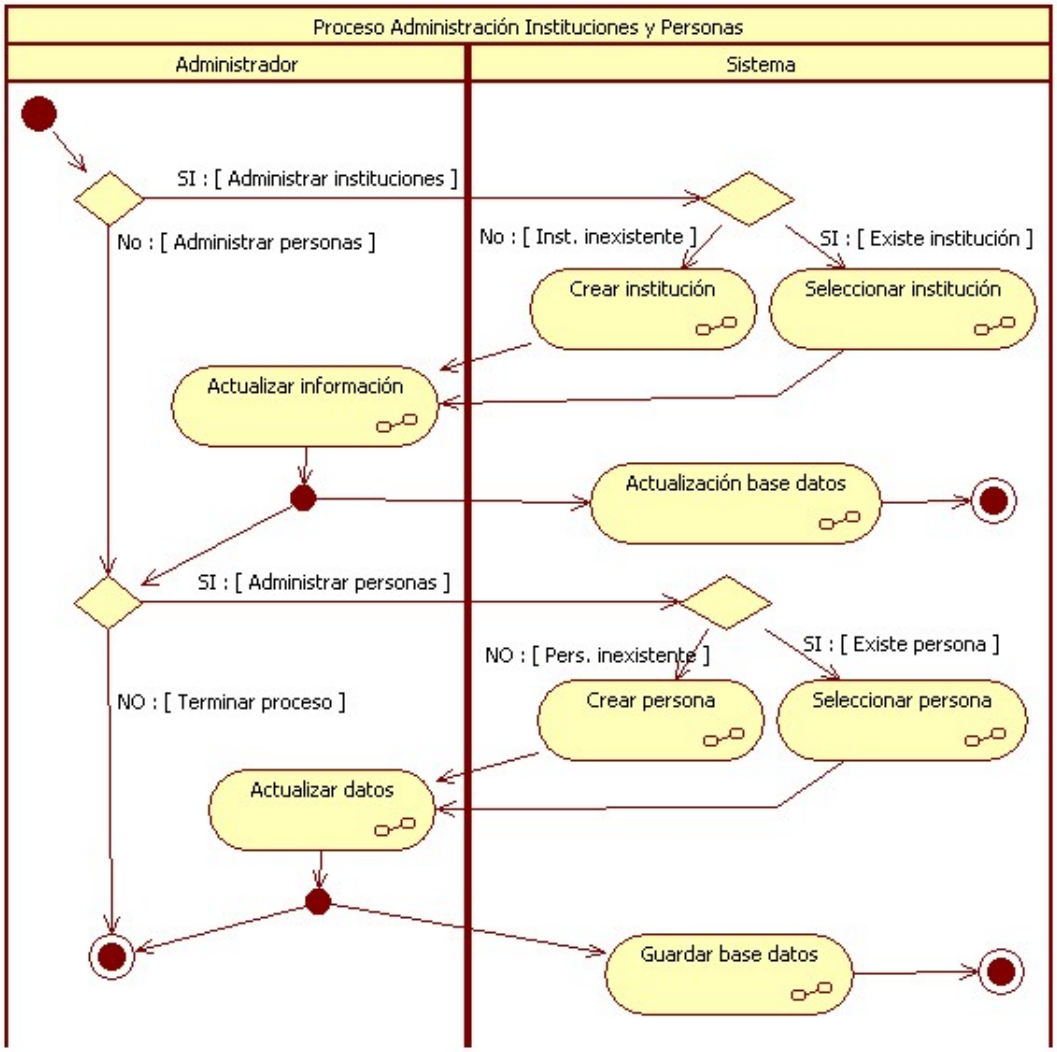

Figura 3.2: Diagrama BPMN del proceso de administración.
Fuente: Elaboración propia

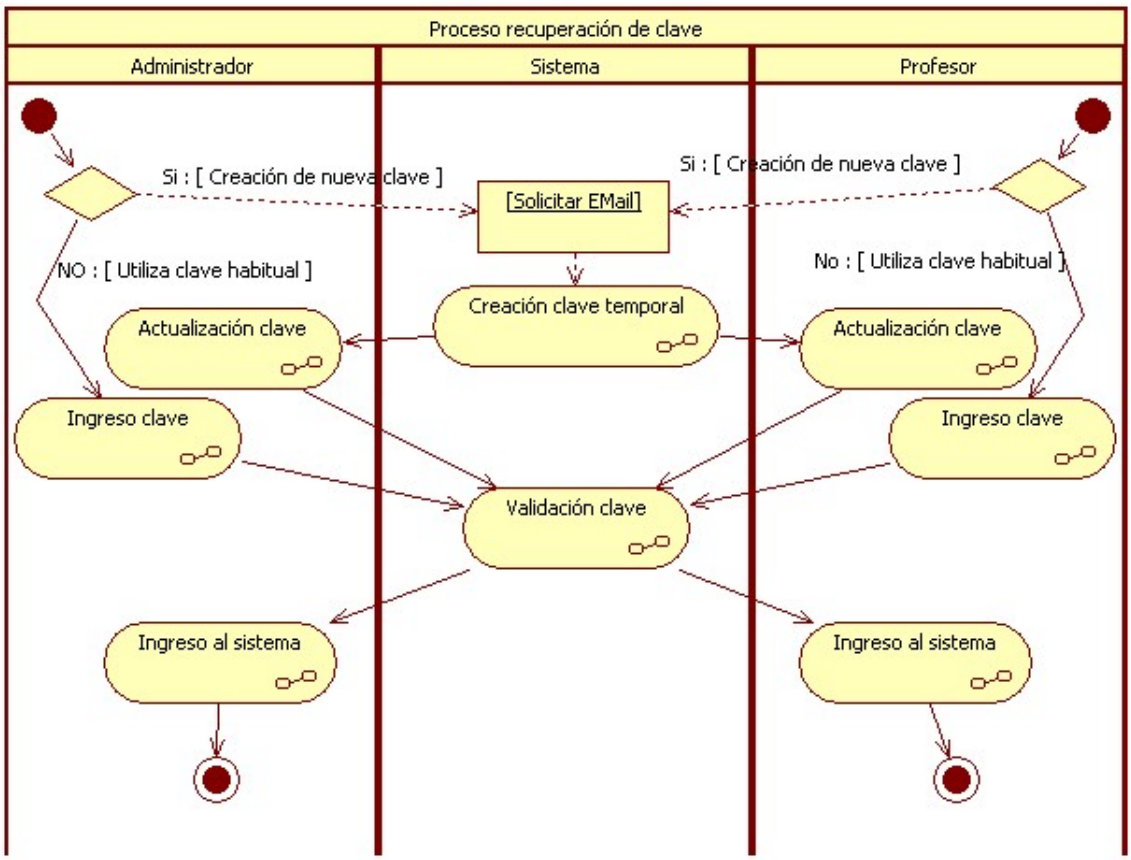

Figura 3.3: Diagrama BPMN del proceso de recuperación de clave.
Fuente: Elaboración propia

sobre el alumno (email, rut y nombres). Esta opción permite la creación de listas dentro del sistema, modificar datos y eliminar listas existentes.

- **Tareas:** Esta opción permite la creación de Tareas dentro del sistema, modificarlas, asignarlas a una lista y descargar las tareas ya entregadas.

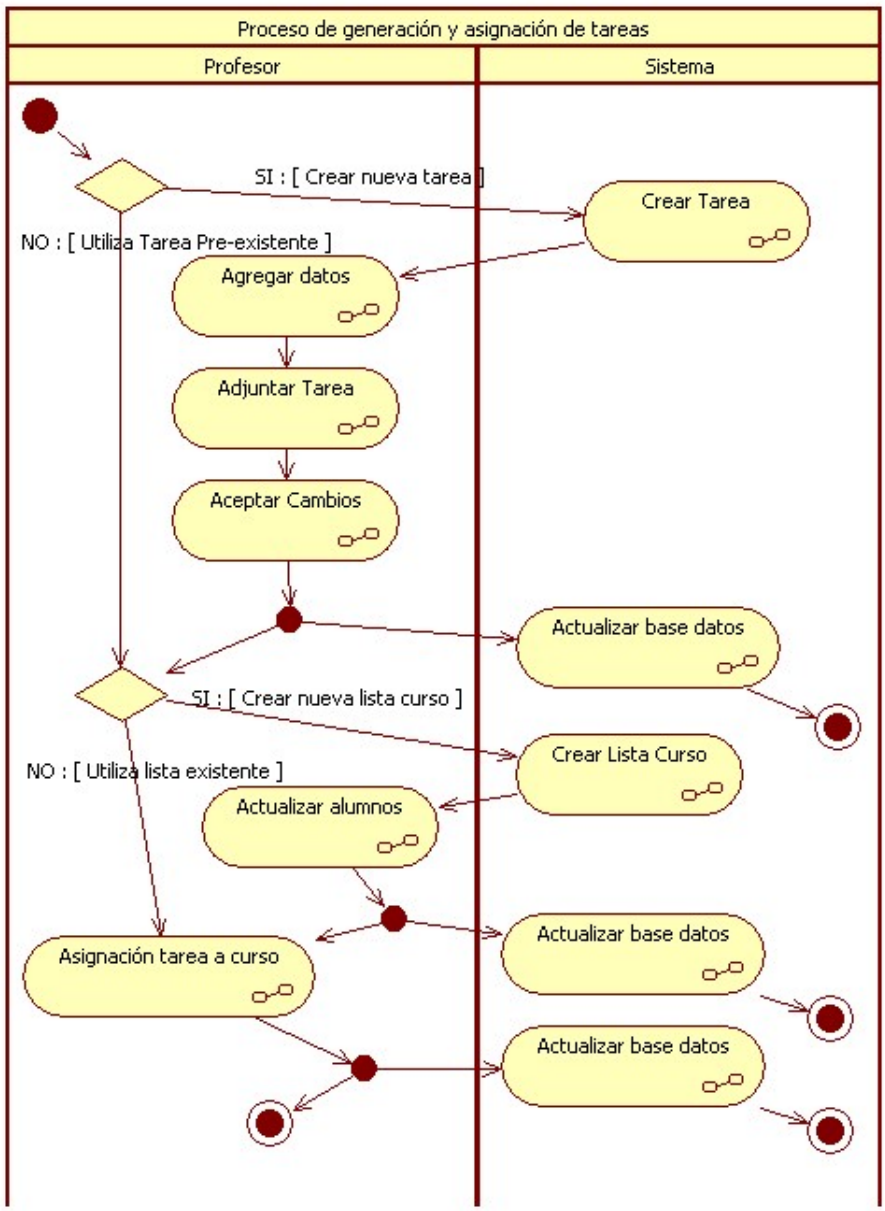

Figura 3.4: Diagrama BPMN del proceso de generación y administración de tareas.
Fuente: Elaboración propia

- **Revisión de tareas:** Los actores involucrados son el profesor, el alumno y el sistema. Se incluyen en este caso de uso las acciones involucradas en la revisión de tareas. En la Figura 3.5 se aprecia el diagrama BPMN con los procesos que realizan los diferentes actores del

proceso. El Alumno y el almacenamiento de tareas, mientras dura el período de entrega de tareas, se encuentran enmarcados. Esto para resaltar que este proceso puede realizarse cada vez que el alumno lo desee mientras se encuentra vigente el período de entrega de tareas, accediendo mediante el link en el correo que se le envía a la opción de entregas del sistema.

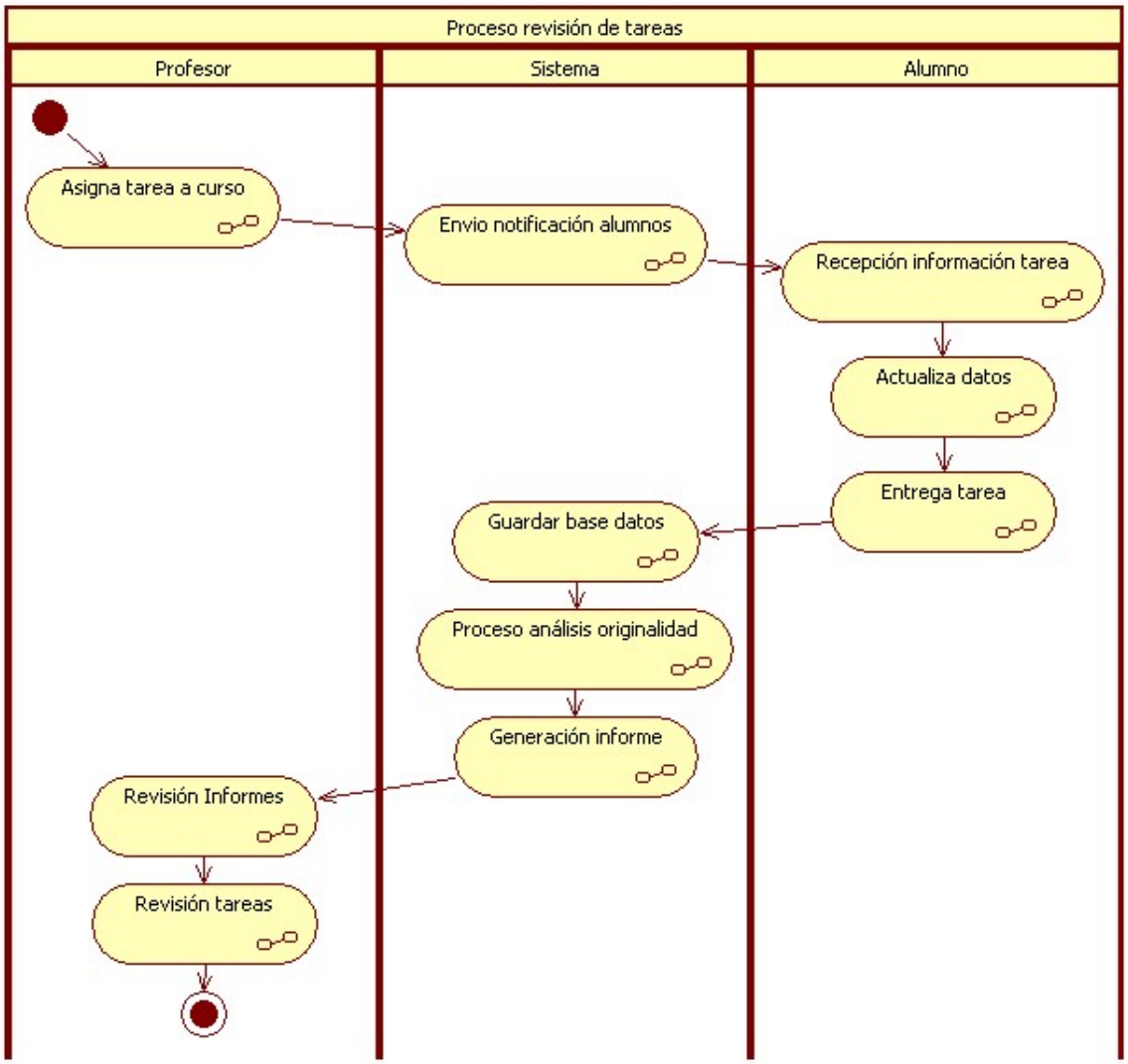

Figura 3.5: Diagrama BPMN del proceso de revisión de tareas.
Fuente: Elaboración propia

3.2.2. Roles del sistema

Los roles definidos en el sistema corresponden al de Super administrador, Administrador, Profesor y Alumno. A continuación se hace una breve descripción de cada uno, caracterizándolos en términos de sus atribuciones.

Super administrador: Tiene a cargo la administración de todos los perfiles existentes, además

de la supervisión de todos los procesos del mismo.

Administrador: Tiene a cargo las tareas del proceso de administración en DOCODE. A él corresponden las acciones de crear, modificar y eliminar usuarios e instituciones que tendrán acceso al sistema.

Profesor: Un usuario bajo este rol, pertenece siempre a la misma institución y puede acceder a la creación de listas de cursos y tareas, revisar la entrega y descargar las tareas entregadas. Además puede enviar las tareas a revisión al DOCODE Engine.

Alumno: Si bien no tiene acceso a la interfaz online de DOCODE donde se aprecian los resultados del análisis de originalidad, su rol es fundamental y se ejerce a través de la realización de las tareas. Su función básica es la de responder las tareas asignadas, todo esto a través de la utilización del email.

En el Cuadro 3.1, se hace una descripción de los roles definidos anteriormente en función de las actividades que pueden realizar.

ROL	Actividades	Observaciones
Super Administrador	▪ Creación de personas. ▪ Creación de instituciones. ▪ Creación de listas de curso. ▪ Creación de tareas.	
Administrador	▪ Creación de profesores. ▪ Recuperación de clave.	
Profesor	▪ Creación de listas de curso. ▪ Creación y asignación de tareas. ▪ Revisión de reportes de análisis de originalidad.	▪ El usuario bajo el rol de profesor sólo existe como parte de una institución. ▪ Si un mismo profesor existe en dos instituciones, será creado dos veces.
Alumno	▪ Recibir y subir tareas al sistema.	▪ El usuario alumno no existe como tal en el sistema, sino como parte de una lista curso.

Cuadro 3.1: Roles y actividades de los perfiles en DOCODE.

3.2.3. Descripción del código fuente

A continuación se presenta un resumen con lo necesario para la instalación de DOCODE SaaS, así como un esquema de la estructuración de la Base de Datos detrás de DOCODE SaaS.

Instalación Para el correcto funcionamiento de los componentes de DOCODE SaaS es necesario utilizar los siguientes programas:

- Mysql 5.0.X
- Apache 2.2.X
- PHP 5.1.X

Estructura de carpetas de instalación **Carpeta lib** Contiene las librerías externas utilizadas por la aplicación.

Carpeta include Contiene las librerías propias de la aplicación.

Carpeta template Contiene los templates de HTML utilizados por la aplicación.

Carpeta template/tmp Carpeta que contiene los compilados de los templates de la aplicación.

Carpeta php Contiene los scripts que implementan las distintas funcionalidades de la aplicación.

Carpeta web Contiene las imágenes, estilos y jsp utilizados por la aplicación. Es la única carpeta que debe quedar pública.

Modificaciones archivos configuración Para la instalación se requiere realizar los siguientes cambios en el archivo config.php dentro de la carpeta web:

- Modificar las siguientes variables para indicar las distintas rutas a la aplicación.

```
$ruta_web : ruta web de la aplicación.
$ruta_diseno: ruta de la carpeta de diseño.
$ruta_archivos: ruta de los archivos que se suben a la plataforma.
```

- Modificar los valores del siguiente arreglo para establecer la conexión a la Base de Datos:

```
$db = array(
   'user' => usuario,
   'pass' => password,
   'name' => Nombre de la Base de Datos,
   'host' => Servidor,
   'type' => Administrador de Base de datos,
);
```

Base de datos En la Figura 3.6 se presenta la estructura de la base de datos detrás del software DOCODE SaaS. Como se puede observar, las tablas se encuentran no relacionadas entre ellas a través de llaves foráneas sino que son un conjunto de tablas aisladas.

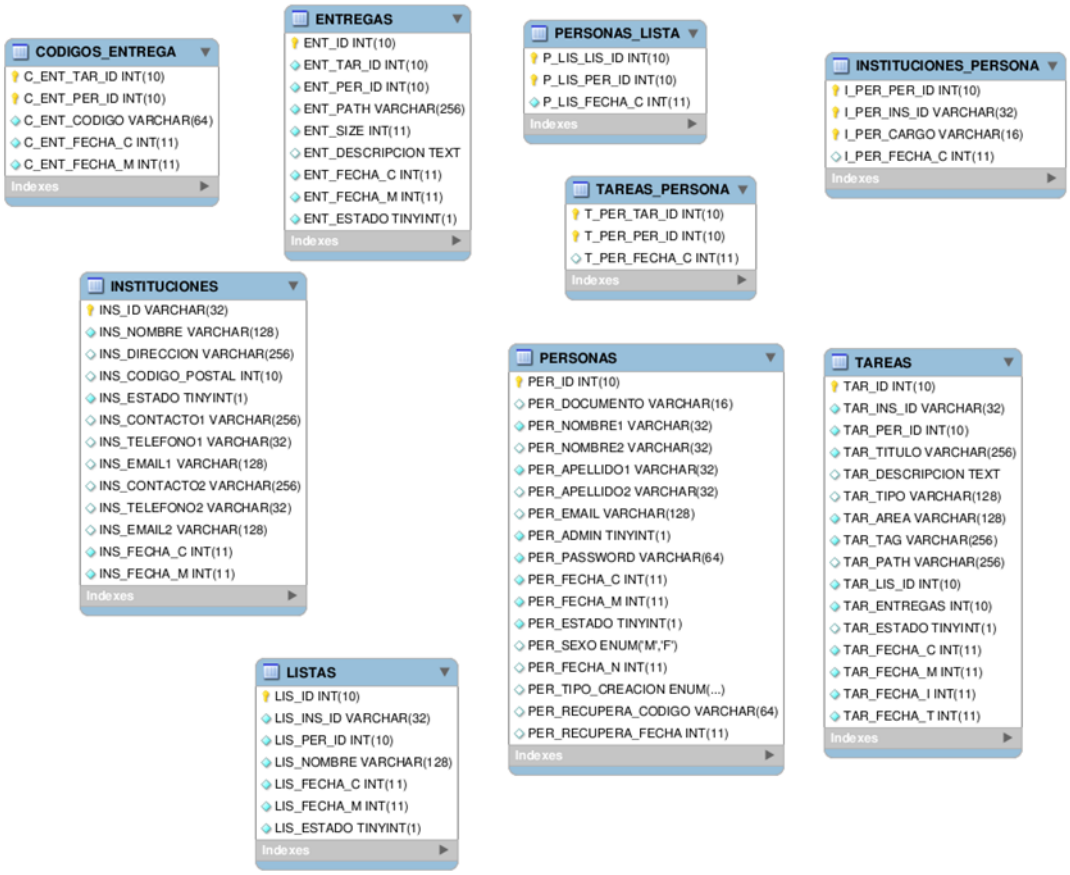

Figura 3.6: Modelo de la base de datos relacional.
Fuente: Elaboración propia

3.2.4. Funcionalidades de software

El Anexo A contiene la definición funcional de DOCODE mediante casos de uso. En ellos, se pueden ver las funciones, que pueden ser realizadas por los distintos perfiles de usuario definidos en el software, según se indica en cada cuadro.

Estas funcionalidades no serán cambiadas por la reestructuración de la solución de DOCODE, puesto que las mejoras están orientadas principalmente a la capa de servicios y estas definiciones afectan a la capa de vista.

3.2.5. Interfaz de usuario DOCODE SaaS

A continuación se presenta la documentación de la interfaz de usuario. Las funcionalidades presentadas en esta sección se replican para aquellos usuarios conectados vía u-cursos y moodle, sistemas para los cuales se han desarrollado las mismas funcionalidades.

Ingreso al Sistema

Para ingresar al sistema, es necesario utilizar el nombre de usuario y contraseña. La Figura 3.7, presenta la interfaz de usuario donde se ingresan estos datos. En ella, se solicitan 2 campos obligatorios (Correo Electrónico y Contraseña). En caso que se olvide la contraseña, se puede recuperar

Figura 3.7: Interfaz de autenticación de DOCODE SaaS.
Fuente: Elaboración propia

mediante el enlace "¿Olvidaste tu clave?". Esta opción solicita el correo electrónico previamente registrado en el sistema DOCODE ASP. La Figura 3.8, corresponde a la interfaz de recuperar contraseña, en ella se aprecia que también es necesario llenar un campos CAPTCHA, de modo de validar que el usuario que está solicitando la contraseña sea un usuario humano y no un robot. Si el usuario ejecuta correctamente estos pasos, puede recuperar su contraseña enviada por DOCODE al correo electrónico previamente indicado.

Figura 3.8: Formulario recuperación contraseña.
Fuente: Elaboración propia

Usuario super administrador

El usuario super administrador es quien puede crear, activar y desactivar instituciones, además de editar los contactos e información base de cada una de las instituciones. También puede modificar la información de cualquier usuario (tanto profesores registrados, cómo alumnos). La Figura 3.9, muestra el menú de inicio de este perfil, en donde se puede acceder a administrar Instituciones o Personas.

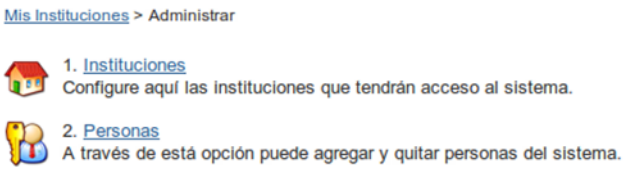

Figura 3.9: Opciones de administración del perfil Super Administrador.
Fuente: Elaboración propia

Administrar Instituciones Esta corresponde a una de las principales funcionalidades del super administrador, a continuación se despliegan figuras con las principales funcionalidades.

La Figura 3.10, muestra la interfaz donde se administran las instituciones, en ella se pueden ver las opciones de agregar nuevas instituciones, o editar las instituciones ya existentes.

La Figura 3.11, muestra la interfaz que se despliega al agregar una institución nueva o al editar una existente, donde es necesario completar los valores de cada campo con la información

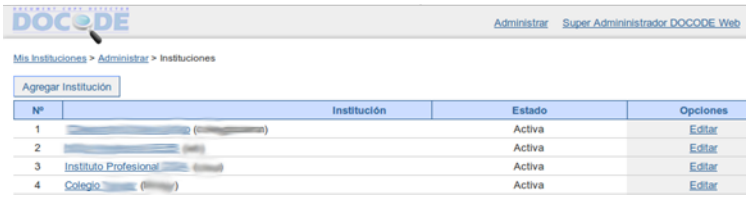

Figura 3.10: Opciones de administración de instituciones.
Fuente: Elaboración propia

relacionada a la institución que se está agregando o editando.

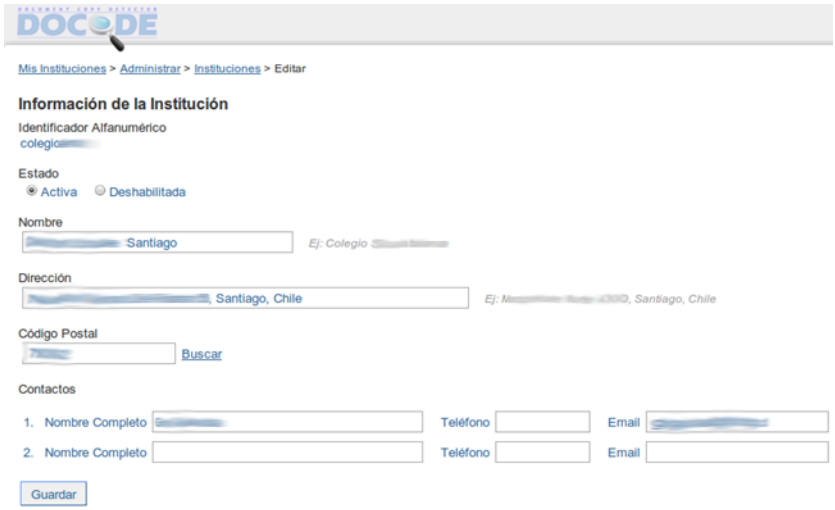

Figura 3.11: Formulario de edición de una institución.
Fuente: Elaboración propia

Administrar personas La Figura 3.12, muestra la interfaz donde se pueden hacer búsquedas de los usuarios que están disponibles en el sistema, listar a todas las personas en el sistema, además de tener la opción de agregar nuevas personas.

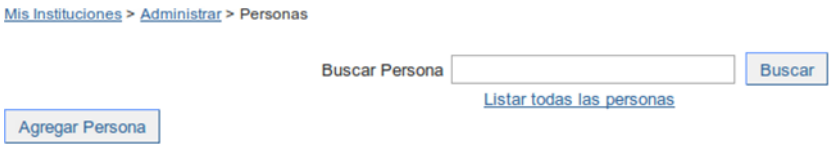

Figura 3.12: Herramienta de búsqueda de personas.
Fuente: Elaboración propia

La Figura 3.13, muestra la interfaz donde se puede agregar nuevos usuarios en el sistema. Hay que destacar la opción de asignarle permisos de super usuario, ya que esta característica sólo puede ser fijada por un super usuario.

Figura 3.13: Formulario de edición de personas desde el perfil de super administrador.
Fuente: Elaboración propia

La Figura 3.14, muestra como se listan las personas que son resultado de la búsqueda, la cual, puede realizarse por cualquiera de los campos ingresados en las personas presentados en la Figura 3.13. El listado resultado muestra nombre, correo electrónico, RUT e instituciones relacionadas. Hay que destacar que en el resultado, al costado del nombre de usuario, se tiene la opción de hacer "switch"con el perfil del usuario. Esto permite ingresar al sistema personificando al usuario y de esta forma, revisar las opciones y preferencias que maneja en un determinado momento. Esto es relevante para identificar problemas que puedan tener los usuarios, y revisar desde la línea de soporte directamente aquellas complicaciones que pueda tener un determinado usuario final con respecto al sistema.

Figura 3.14: Resultado de búsqueda de usuarios.
Fuente: Elaboración propia

Usuario Administrador

Puede realizar tres tipos de operaciones: administrar los profesores de una institución, administrar las listas de curso y administrar las tareas. La Figura 3.15, muestra el menú de inicio del perfil profesor, donde puede acceder a cada una de las opciones de perfil: Administrar profesores, administrar listas de cursos y tareas.

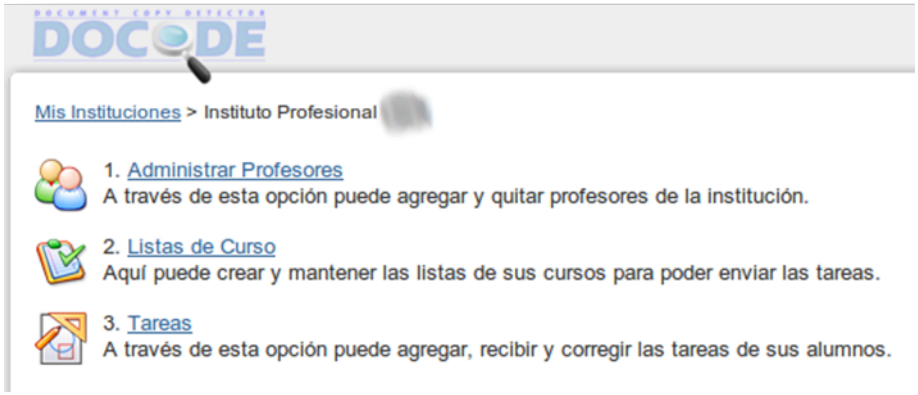

Figura 3.15: Opciones de administración del perfil Administrador.
Fuente: Elaboración propia

Administrar profesores Permite la administración de profesores. Esta operación es fundamental para el iniciar las actividades de una institución en DOCODE SaaS, ya que los profesores son quienes tienen la responsabilidad de crear las listas de usuarios, como también crear las tareas

que los usuarios alumnos deberán responder y entregar al sistema.

La Figura 3.16, muestra la interfaz con el resultado de la búsqueda de profesores que están disponibles en la institución seleccionada. Desde aquí se puede ir al formulario de edición y eliminación de profesores.

Figura 3.16: Resultado de búsqueda de profesor.
Fuente: Elaboración propia

La Figura 3.17 muestra el formulario de administración de los datos de un profesor. Al agregar un nuevo profesor, se debe ingresar su correo electrónico, RUT, nombres y apellidos, sexo y fecha de nacimiento.

Figura 3.17: Formulario para administrar datos de un profesor.
Fuente: Elaboración propia

Otras opciones de administración Las opciones de administrar listas de cursos y tareas se presentan en la sección del Usuario Profesor.

Usuario Profesor

Puede gestionar listas de cursos y tareas, además de revisar los reportes entregados por DOCODE para cada tarea creada por el profesor. La Figura 3.18 muestra las opciones del menú del perfil Profesor.

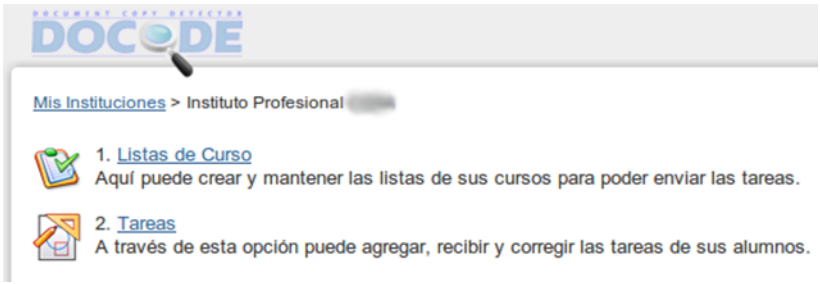

Figura 3.18: Opciones del menú del perfil Profesor.
Fuente: Elaboración propia

Administración listas de curso Lo primero que debe hacer un Profesor, es crear una lista de curso, para luego crear una tarea. Para crear la lista de curso, se selecciona la opción de *Listas de curso*. La Figura 3.19 muestra la interfaz donde se despliegan todas las listas de cursos de un profesor. También se pueden ver las opciones para agregar una nueva lista y editar las anteriores. Al seleccionar una lista de curso, se despliega la lista de alumnos que ella contiene.

Figura 3.19: Listas de cursos para el Usuario Profesor.
Fuente: Elaboración propia

La Figura 3.20, muestra la interfaz donde se puede administrar los alumnos de una lista de curso. Los alumnos se pueden cargar uno a uno o utilizando un template para carga masiva. Terminado este proceso, es indispensable presionar el botón *Guardar* ya que de otra forma, no se guardan los cambios dentro de la lista.

No es obligatorio para el profesor completar toda la información de los alumnos, dado que el alumno es quien puede completar toda la información que falte. Esto se explica en el Usuario Alumno. El campo más importante es el correo electrónico, dado que es la vía mediante la cual el sistema se conecta con el usuario alumno.

Administración de tareas Después de crear las listas de curso, el profesor esta en condiciones de crear tareas. Con este fin, selecciona la opción *Tareas* del menú del perfil Profesor. La Figura

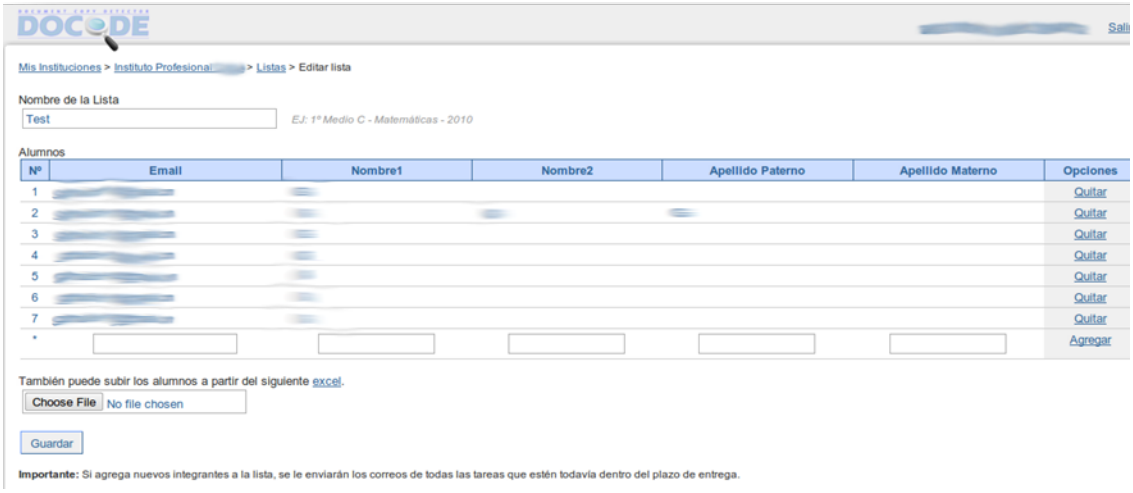

Figura 3.20: Listado de usuarios dentro de un curso.
Fuente: Elaboración propia

3.21, muestra la interfaz donde se despliegan todas las tareas de un profesor separadas según la lista de curso correspondiente. También se pueden ver las opciones para agregar una nueva tarea y editar las anteriores. Desde esta interfaz se accede al reporte de análisis de originalidad, presionando *Revisar* al costado de la tarea seleccionada.

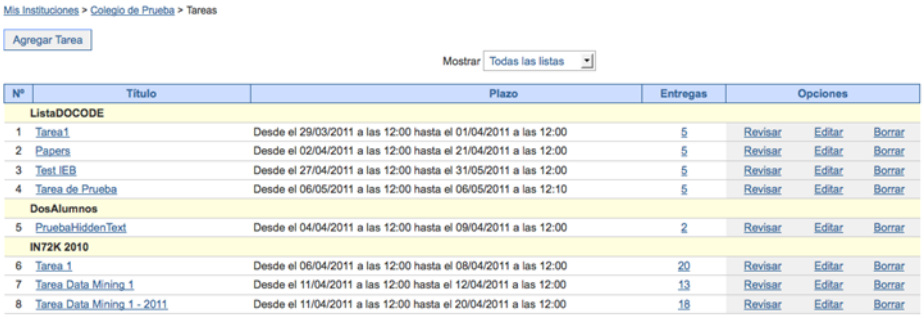

Figura 3.21: Listado de tareas de un profesor.
Fuente: Elaboración propia

Al agregar una nueva tarea, se debe indicar el curso sobre el cual se aplica, además de los parámetros correspondientes a la tarea. Un campo muy importante es el que permite subir un archivo, el cual es usado como parte del enunciado de la tarea. La Figura 3.22 muestra el formulario donde se administran los datos de una tarea, en ella se ingresan: Título, tipo, palabras clave (conceptos relevantes), área de trabajo, descripción y el período de validez de la tarea. Terminado este proceso, es indispensable presionar el botón *Guardar* ya que de otra forma, no se guardan los cambios dentro de la tarea.

Al seleccionar la opción *Revisar* dentro del listado de tareas, se puede ver el reporte del análisis de originalidad de la lista de curso. En ella, una vez que los alumnos responden y envían sus

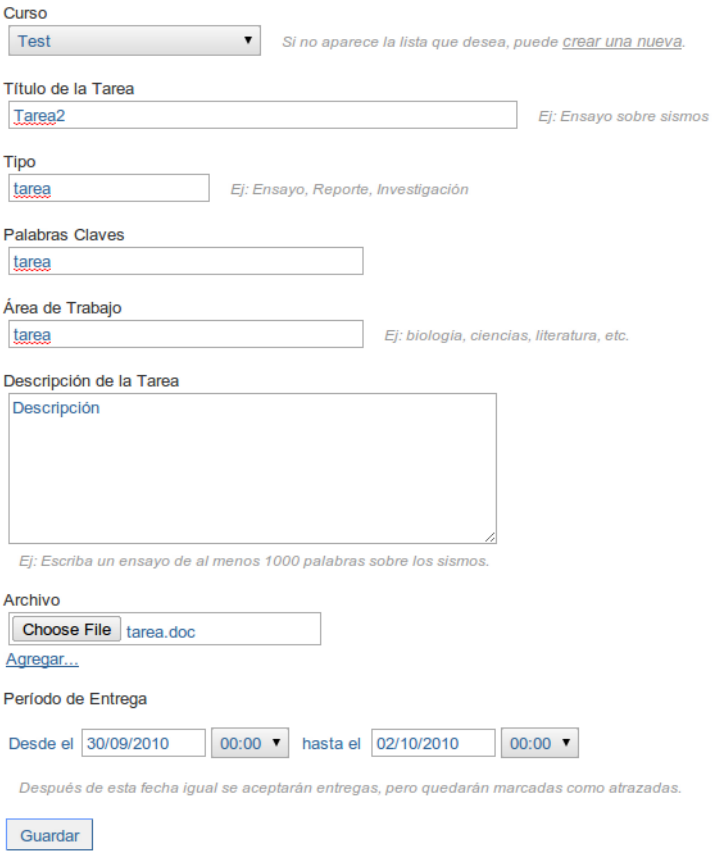

Figura 3.22: Formulario para administrar tareas.
Fuente: Elaboración propia

documentos, se puede visualizar la fecha que el alumno revisó la tarea, la fecha de entrega, y la información sobre el detalle ingresado por el alumno y la descarga del documento. En caso que algún alumno no haya respondido la tarea, y no haya revisado aún el correo por algún motivo, es posible reenviar la tarea para que el alumno vuelva a tener la opción de entregar su tarea. La Figura 3.23 muestra el reporte de resultados para la lista de curso. Se debe notar que este reporte se puede obtener sólo después de haber aplicado DOCODE sobre las tareas entregadas por los alumnos, para este fin, se debe seleccionar la opción *Solicitar DOCODE* o *Reaplicar DOCODE*, según si es la primera vez que se solicita el análisis o corresponde a una repetición del análisis. Otro detalle importante es el de los puntos suspensivos al final de la tarea del alumno, que representa gráficamente un posible plagio de texto oculto. Los

Figura 3.23: Reporte de resultados de DOCODE.
Fuente: Elaboración propia

resultados del reporte de análisis de originalidad, muestran varios puntajes por cada informe subido al sistema.

- Cambio de estilo interno del documento respecto de los distintos párrafos que contiene.
- Similitud del documento respecto del resto del curso.
- Cantidad de documentos rescatados desde la web que se compararon contra el documento.
- Marca de texto oculto, permite identificar que un documento podría estar usando esta técnica para evitar el análisis.
- Opciones administrativas: leer el informe, el detalle de la tarea y la descarga del archivo que entregó el alumno.

A continuación se presentan los reportes generados para visualizar las interacciones entre los documentos entregados por los alumnos.

Reporte de similitud dentro del curso Se estima un reporte de similaridad comparando cada archivo entregado por los alumnos con cada uno de los archivos entregados por

los otros alumnos. Después de evaluar la similitud entre los documentos del curso, se presentan los datos mediante un grafo. En la Figura 3.24 se puede ver el grafo que muestra el grado de similitud entre los documentos de un curso de 5 alumnos, donde cada documento es representado por un nodo en la red, y el puntaje en las aristas corresponde al nivel de similitud entre los documentos relacionados.

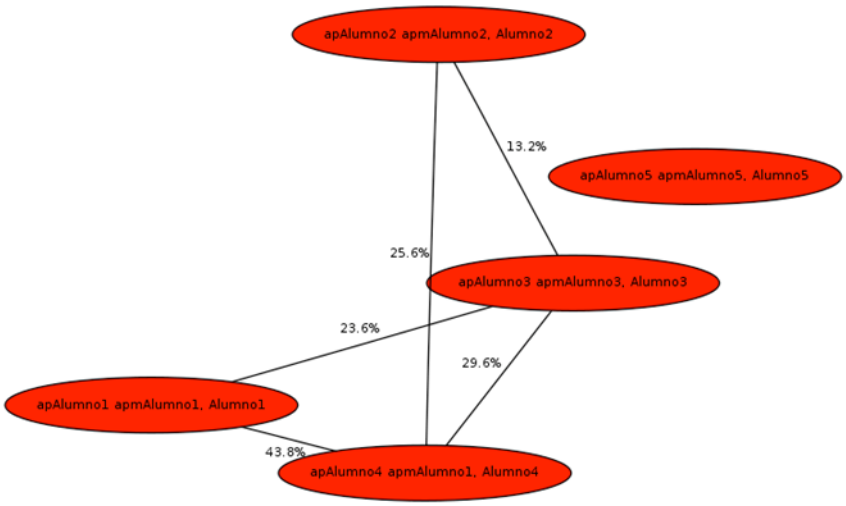

Figura 3.24: Grafo de la similitud de las tareas.
Fuente: Elaboración propia

Reporte de cercanía semántica entre documentos Este reporte muestra la distancia semántica entre los documentos de la tarea utilizando una métrica de *Ánalisis Semantico Latente (LSA)* [19], comparando cada archivo entregado por los alumnos con cada uno de los archivos entregados por los otros alumnos. En la Figura 3.25 se puede ver la representación gráfica de la cercanía semántica entre cinco tareas, donde una de las tareas se ve desconectada de las otras.

Reporte cercanía todas las tareas con la Web Como una forma de desplegar rápidamente la información de todo el curso, se muestran todas las tareas y las fuentes dentro del mismo grafo. En la Figura 3.26 se puede ver el grafo que es una representación de los distintos documentos entregados en un curso y su relación con los documentos encontrados en distintos sitios Web. En este caso, se puede ver que hay una variada gama de sitios Web de los cuales se lograron verificar algún grado de similitud.

Detalle Indice Similitud entre dos documentos de la misma tarea Se calcula un índice de copia entre un documento y cada documento entregado en la tarea. En la Figura 3.27 se pude ver este índice (al costado de la etiqueta *Detalle Similitud Curso*). Además seleccionando alguno de los otros documentos listados, es posible ver al costado izquierdo

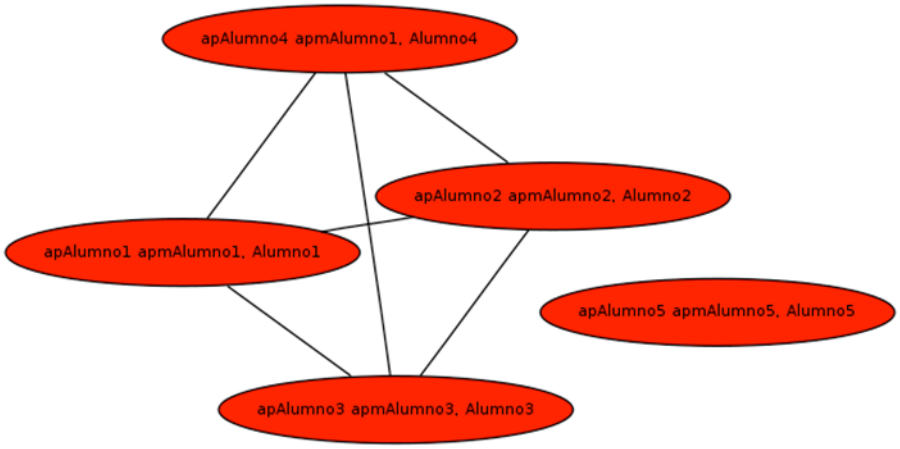

Figura 3.25: Grafo de la cercanía semántica de las tareas.
Fuente: Elaboración propia

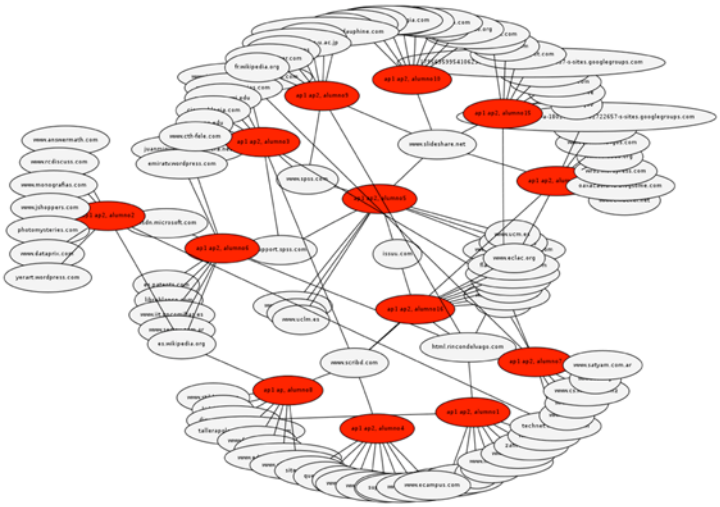

Figura 3.26: Grafo de la cercanía de todas las tareas con los sitios web.
Fuente: Elaboración propia

el documento original y al derecho el documento comparado, mostrando en color amarillo los textos que resultaron como posible copia.

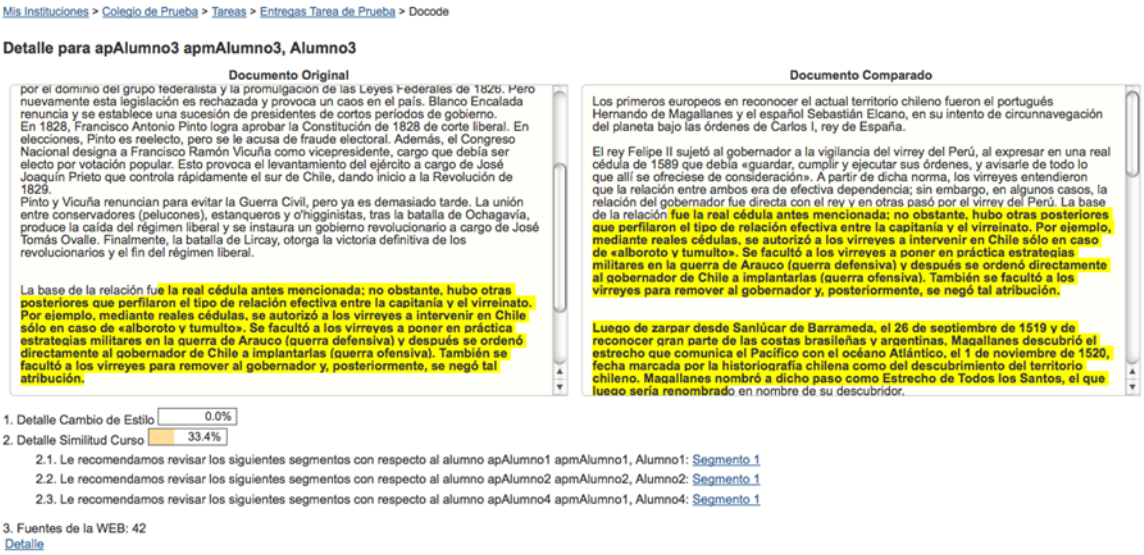

Figura 3.27: Comparación entre dos documentos de la misma tarea.
Fuente: Elaboración propia

Índice similitud entre un documento y la Web se calcula un índice de originalidad entre cada documento y las posibles fuentes web que se identificaron. En la Figura 3.28, se muestra a la izquierda el texto original y a la derecha, una lista con las posibles fuentes web, además de un índice de originalidad donde se evalúa el documento contra la fuente web indicada.

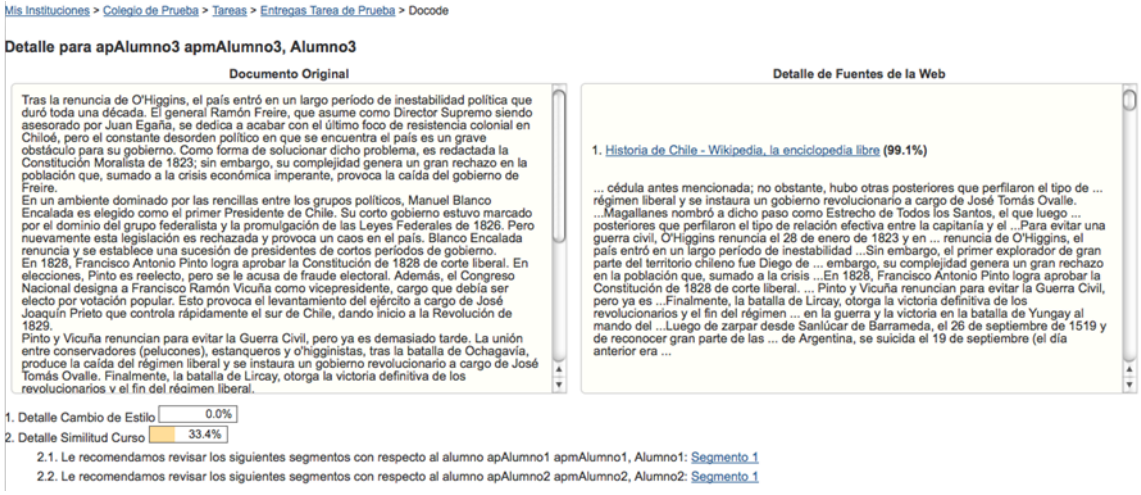

Figura 3.28: Comparación entre el documento y los resultados extraídos de la web.
Fuente: Elaboración propia

Reporte cercanía de las tareas con la Web Este reporte muestra una tarea relacionada con todas las posibles fuentes web. En la Figura 3.29 se puede ver un grafo, donde un

documento aparece relacionado con las posibles fuentes y en los arcos, aparece el valor obtenido en el índice de originalidad.

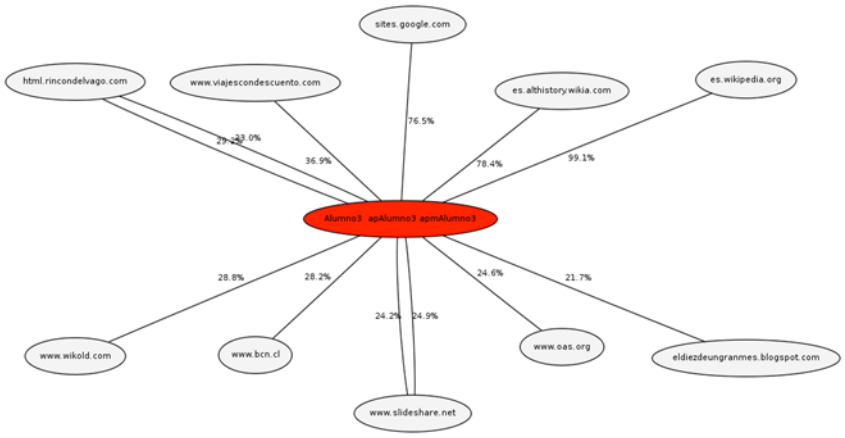

Figura 3.29: Grafo de la cercanía de una tarea con los sitios web.
Fuente: Elaboración propia

Usuario Alumno

Como se mencionó anteriormente, el perfil de Alumno trabaja principalmente por fuera del sistema. A continuación se presentan los pasos que debe ejecutar para ingresar una tarea en el sistema.

1. **Recepción correo electrónico de nueva tarea** Inicialmente, el Alumno recibe un correo electrónico informando que hay disponible una nueva tarea en el sistema y puede ingresar a revisar el detalle de la tarea en el link que está dentro del mensaje. La Figura 3.30 muestra el detalle del correo electrónico y del link que se envia al usuario para que pueda subir el documento de la tarea.

Figura 3.30: Correo electrónico enviado al alumno cuando se crea una tarea.
Fuente: Elaboración propia

2. **Procedimiento datos personales** Una vez el alumno haya terminado de realizar sus deberes, es necesario que suba el documento que lo respalda. Al intentar hacer esto, el sistema solicita al usuario que ingrese o actualice sus datos personales. La Figura 3.31 muestra el formulario que debe completar el alumno con sus datos personales.

Figura 3.31: Formulario donde el alumno ingresa datos personales.
Fuente: Elaboración propia

3. **Subir documento de tarea** Una vez que termina el proceso anterior, el sistema re-dirige al usuario hacia otro formulario, donde puede subir el documento de la tarea. La Figura 3.32 muestra el formulario de entrega de la tarea, donde se despliega el detalle de la tarea y se pueden agregar además del documento, algunos comentarios para cuando revise el profesor. En este formulario se pueden ver: el título, la descripción, el archivo asociado (o enunciado), los plazos definidos por el profesor, el input para subir la tarea y un comentario para la revisión del profesor, el cual es opcional. Si la tarea es entregada fuera del plazo, se recibe, pero es criterio del profesor revisarla.

4. **Validación entrega en el servidor** Una vez entregada la tarea, DOCODE presenta una pantalla que muestra alguna información relativa a la correcta carga del documento en el servidor. La Figura 3.33 muestra el mensaje del sistema con la entrega realizada correctamente por el usuario. Hay que destacar el número de operación, ya que es un identificador único de la entrega, permitiendo después recuperar información relevante de la entrega en caso de ser necesario.

5. **Correo electrónico de confirmación de tarea cargada** Además de la validación por

Figura 3.32: Formulario que permite subir la tarea.
Fuente: Elaboración propia

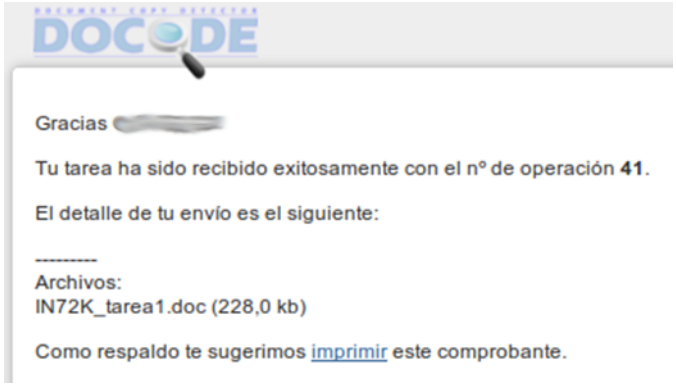

Figura 3.33: Validación de carga de la tarea en el sistema.
Fuente: Elaboración propia

pantalla, el sistema envía automáticamente un correo al alumno con la información desplegada por pantalla. De esta forma el usuario queda con un respaldo del envio. La Figura 3.34 muestra el correo que recibe el usuario, aquí se puede validar la consistencia entre los datos desplegados dentro del sistema con los recibidos en el correo.

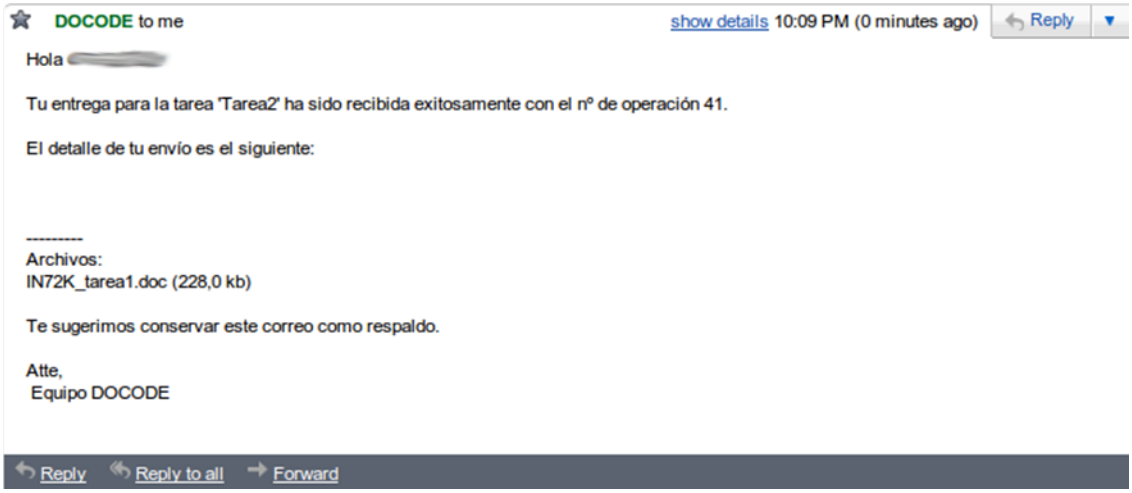

Figura 3.34: Correo electrónico de validación de la entrega de tareas en el sistema.
Fuente: Elaboración propia

3.3. DOCODE Engine

Esta sección presenta los principales casos de uso de DOCODE Engine, junto con un diagrama de flujo que pretende describir la interacción entre las distintas capas que soportan las operaciones de DOCODE. Junto con los diagramas descritos anteriormente, se presentan las principales componentes del Software y una breve descripción de cada una de ellas.

3.3.1. Casos de uso relacionados con el DOCODE Engine

DOCODE Engine es el nombre que recibe un componente de la plataforma DOCODE, donde se ejecutan los procesos de análisis de originalidad sobre los documentos ingresados. Las dos operaciones básicas que pueden ejecutarse desde sistemas externos son: cargar una lista de tareas y consultar el resultado de las tareas. La Figura 3.35 muestra el diagrama de casos de uso de DOCODE Engine.

Cargar una lista de tareas El caso de uso puede ser extendido en cualquier sistema externo que desee enviar archivos a analizar en DOCODE. Este caso de uso consiste en el envío de varios archivos que componen una tarea (mínimo 3). Por cada envío, el sistema retorna un número de atención (token), el cual es utilizado posteriormente para consultar el estado del envío por

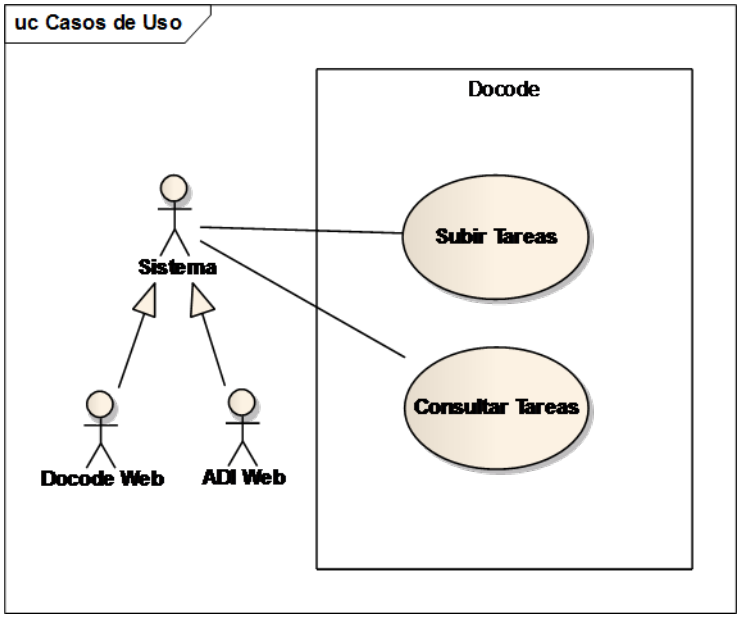

Figura 3.35: Diagrama casos de uso DOCODE Engine.
Fuente: Elaboración propia

el sistema externo.

Algunos detalles técnicos de la operación relacionada con cargar un conjunto de tareas:

- El envío de los archivos es mediante HTTP (POST, multipart/form-data).
- Cada archivo debe tener un nombre del tipo entregas[x], donde x es un número único para cada tarea.
- El token de respuesta es retornado como parte de un mensaje en formato XML.
- La tareas son procesadas en forma asincrónica, por lo que el sistema cliente tiene la responsabilidad de verificar cuando las tareas enviadas ya fueron procesadas (esto se hace con el siguiente caso de uso, consulta estado de tareas). Es importante destacar que las tareas son procesadas de forma asíncrona dado que el volumen de tareas enviadas por los distintos clientes y su tiempo de procesamiento respectivo no puede garantizarse, debido a que el tiempo de descargar las referencias web es muy variable. Por lo cuál no es recomendable tratar de mantener una conexión abierta entre el sistema cliente y el DOCODE Engine para retornar los resultados una vez listos, como si fuese una llamada síncrona.

Consultar el estado de tareas Consiste en consultar el estado de un envío previamente efectuado utilizando el token retornado en el caso de uso anterior. Si al consultar por un envío, aún esta siendo procesado, el estado asignado es *procesando*. Si el envío ya fue procesado,

se pueden verificar los resultados que los distintos algoritmos de análisis de originalidad del DOCODE Engine con respecto a los documentos enviados.
Algunos detalles técnicos con respecto al consultar el estado de las tareas:

- La consulta se efectúa mediante HTTP (POST).
- La respuesta es retornada en formato XML.
- Sólo se retorna los datos relacionados a los resultados de los distintos algoritmos de detección de copia de documentos una vez que se ha procesado completamente el conjunto de tareas. En caso contrario, se retorna un XML con un mensaje que señala que aún están siendo procesadas.

3.3.2. Diagrama de Componentes

En los casos de uso anteriores, se explican las funcionalidades de una interfaz utilizada por los sistemas externos, además de DOCODE SaaS. Esta interfaz de software, el sistema de encolamiento de las distintas tareas, todos los algoritmos de detección de plagio, así como el resto de los componentes fueron desarrollados en Java

Componentes principales DOCODE Engine La implementación de DOCODE Engine está en el lenguaje de programación Java, específicamente para ser desplegado en el servidor de aplicaciones Glassfish que cumple con las especificaciones de JEE. El motor de DOCODE Engine está implementado en base a un sistema que recolecta las tareas enviadas por los clientes, y luego mediante la API *Java Message Service (JMS)* se encolan las tareas con su respectivo token, las cuales son posteriormente procesadas por un *Message Driven Bean (MDB)* que tiene la responsabilidad de ejecutar una DocodeTask (que comprende los distintos algoritmos de detección de plagio) y almacenar los resultados en una base de datos, para luego pasar a la siguiente tarea esperando ser procesada. La Figura 3.36 muestra la arquitectura de componentes de DOCODE Engine. Aquí se muestra como las peticiones que llegan desde la interfaz provista son manejadas por un servicio web.

Flujo proceso DOCODE Engine Luego, las peticiones de procesamiento son encoladas en una cola JMS y posteriormente procesadas por el MDB. Este esquema de procesamiento asegura la integridad de la información, además de garantizar el proceso asíncrono de la información. El MDB que procesa las peticiones utiliza un conjunto de librerías para el procesamiento de las tareas, las cuales están desarrolladas en su mayoría dentro del proyecto DOCODE. Entre las librerías se pueden destacar las siguientes:

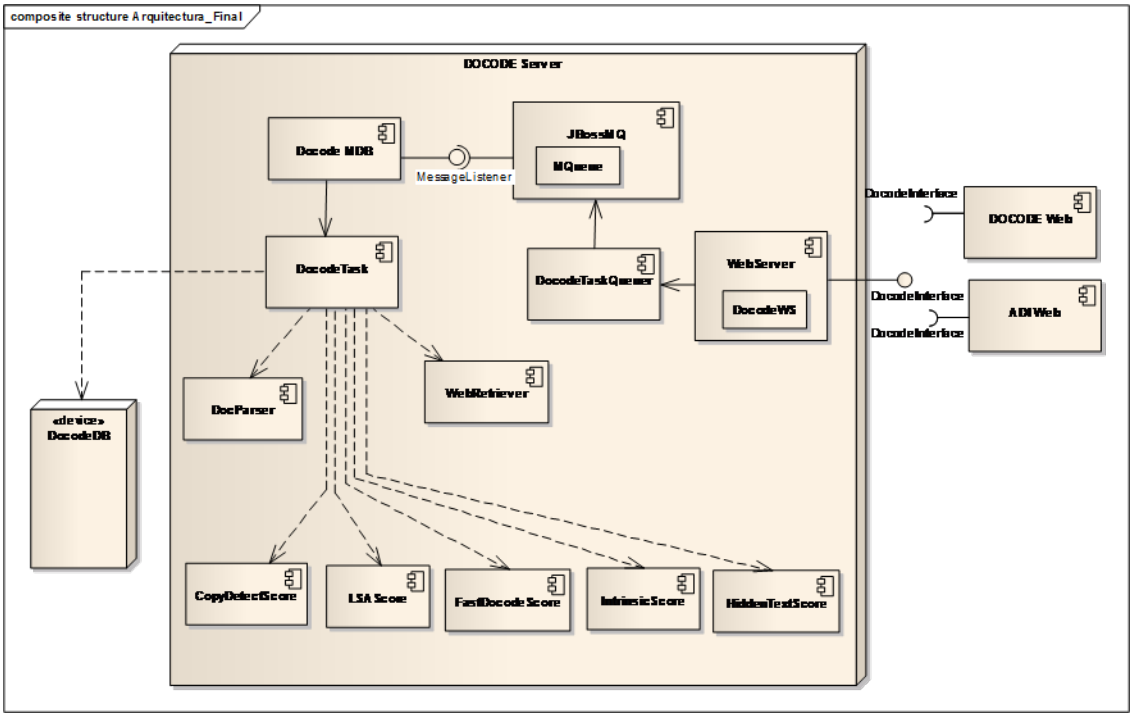

Figura 3.36: Diagrama de componentes de DOCODE Engine.
Fuente: Elaboración propia

DocParser El objetivo es extraer el contenido de las tareas de acuerdo al tipo de documento que sea (PDF, DOC, DOCX, Texto plano, HTML).

WebRetriever Permite la búsqueda de un conjunto de documentos Web similares a un documento de entrada.

CopyDetectScore El objetivo es ejecutar el algoritmo de detección de similitud entre dos documentos basado en distancias de edición.

FastDocodeScore Permite ejecutar el algoritmo para determinar puntajes de copias basado en FastDocode.

LSAScore Calcula la distancia basada en análisis semántico latente (LSA) entre dos documentos.

IntrinsicScore Permite ejecutar el algoritmo de detección de plagio intrínseco para un documento en particular.

HiddenTextScore Permite ejecutar el algoritmo para determinar si existe texto oculto en las tareas.

Almacenamiento DOCODE Engine El almacenamiento persistente de la información se hace a través de dos medios:

- Una base de datos Postgres. La cual cuenta con la información de todas las tareas y el estado de ellas (si se encuentran procesadas o no).
- Carpetas en el servidor para almacenar los archivos de las tareas físicamente, y para generar todos los archivos temporales necesarios para la ejecución de los algoritmos de análisis de originalidad.

3.3.3. Diagrama de Secuencia

Cuando llega una tarea la petición es tomada por el Servicio Web, el cual encola la tarea para su procesamiento asíncrono. De acuerdo a la estrategia planteada, el servidor va tomando las tareas encoladas y las va procesando con DocodeMDB. En la Figura 3.37 se muestra el diagrama de flujo de datos del DOCODE Engine. Se puede ver la interacción entre las capas DOCODE SaaS y DOCODE Engine, además de la interacción entre los componentes propios de DOCODE Engine.

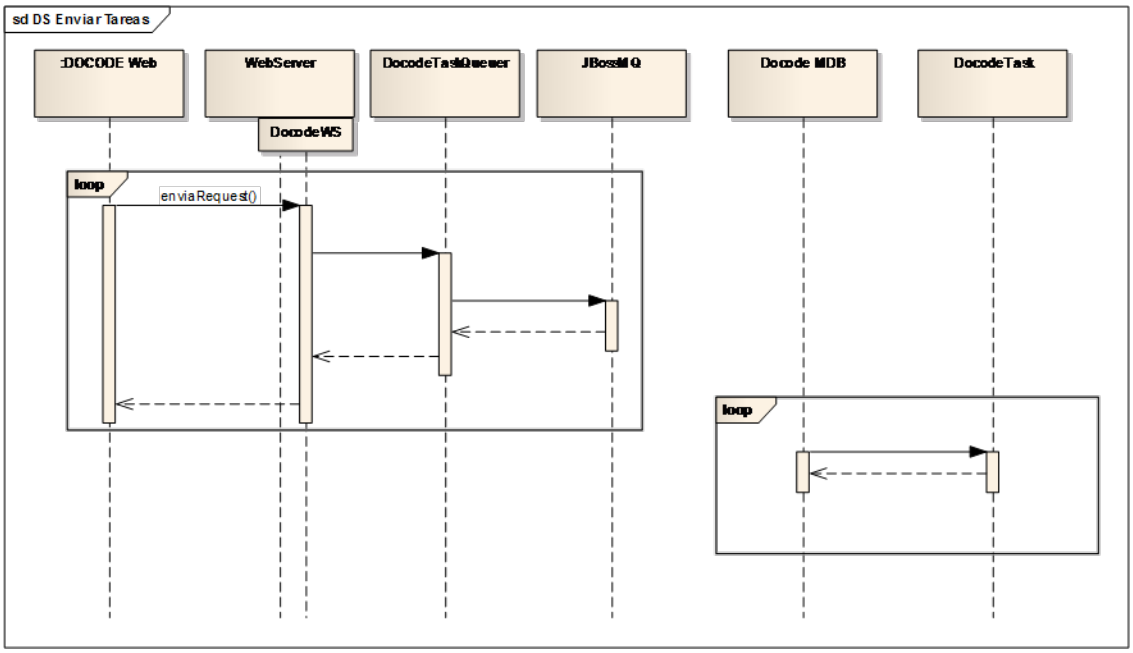

Figura 3.37: Diagrama de flujo de datos de DOCODE Engine.
Fuente: Elaboración propia

3.3.4. Detalles Técnicos

Sistema desarrollado en Java, bajo la arquitectura J2EE.

- Versión Java: 1.6.0 26.
- Herramienta de Desarrollo: Netbeans IDE 7.0 (build 201105291501).
- Archivo de Despliegue: DocodeEE.ear. Contenido:

- DocodeEE-war.war: contiene capa Web de la aplicación.
  - TareaServlet: Servlet que maneja las peticiones Web del Sistema.
- DocodeEE-ejb.jar: contiene capa de Negocio de la aplicación.
  - DocodeMDBBean: MDB que procesa los JMS.
  - DocodeTaskEJB: EJB que posee la lógica de procesamiento.
- docode-service.xml: descriptor XML de la cola de mensajes (JMS).

Dependencias de proyectos externos a DOCODE:

- Collection api
- Colt
- Common-fileupload
- Concurrent
- Dom4j
- Jama
- Jaxrpc
- Jdom
- Json
- Jung
- Log4j
- Lucene
- PDFBox
- Poi
- Postgres-jdbc
- Saaj
- Stax-api

- Wsdl4j
- Wstx-asl
- Xmlbeans

Dependencias de proyectos internos a DOCODE:

- DocodeAlgoritmos
- LSADocode
- Parser
- WebRetriever

Capítulo 4

DOCODE 2.0 - Rediseño plataforma

La arquitectura y funcionalidades definidas en el Capítulo 3 muestran el funcionamiento de la plataforma DOCODE antes de aplicar las mejoras. Las cuales fueron necesarias ya que dentro de los nuevos requerimientos para el sistema, está indicado que debe ser capaz de soportar una alta cantidad de usuarios concurrentes y una carga muy superior a la que ha tenido hasta el momento.

Según los últimos estudios realizados por el área comercial del proyecto, se estima un peak de carga de 20 procesos de análisis de originalidad ejecutados concurrentemente por un estimado de 50 usuarios conectados en la plataforma, con un crecimiento del 30 % a los seis meses del lanzamiento y un 100 % después del primer año de operación.

Tomando en cuenta los distintos requerimientos de la plataforma, se llegó a la siguiente definición de arquitectura. La cual trata de ajustarse a los estándares de implementación y a las mejores prácticas de JEE.

4.1. Definición arquitectura DOCODE 2.0

Inicialmente se define una estructura de capas que soporte toda la plataforma DOCODE. La Figura 4.1 muestra una definición de alto nivel de cada una de las capas que se explican en las siguientes subsecciones.

4.1.1. Capa cliente

Corresponde a los usuarios del sistema que interactúan con alguna de las interfaces que posee el sistema. Los clientes que se conectan a través del navegador se conectan a la capa web, mientras que los otros clientes se conectan directo al *web service*. Lo habitual es que los primeros sean personas y los segundos otros sistemas que están integrados al servicio DOCODE. El principal objetivo de crear los clientes del *web service*, es proveer al sistema DOCODE de una interfaz estandarizada que

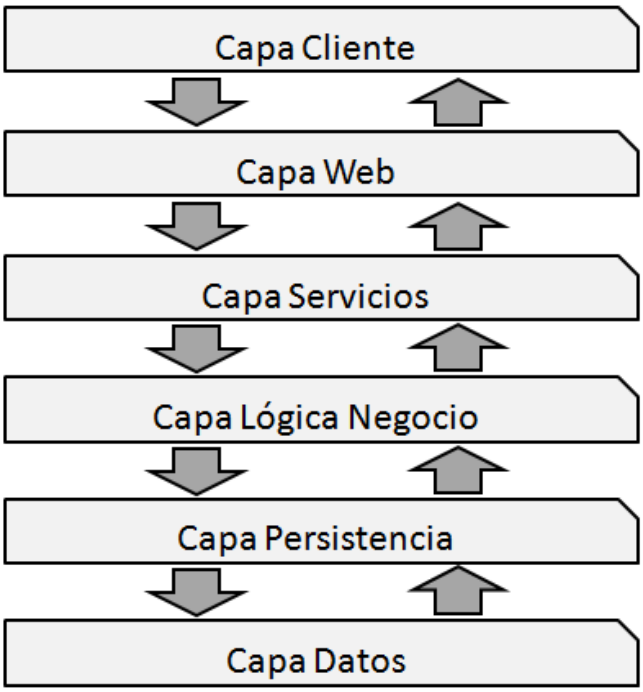

Figura 4.1: Diagrama de capas del sistema DOCODE.
Fuente: elaboración propia

permita conectar sistemas heterogéneos de una forma homogénea mediante la utilización de *web services* SOAP. Esta interfaz considera ejecutar los mismos comandos que el Servlet de la capa web y eventualmente disponibilizar servicios adicionales.

4.1.2. Capa Web

También conocida como *Capa de Presentación*, corresponde a la implementación de interfaces web para los usuarios del sistema que interactúan directamente con alguno de los sistemas que ya están implementados (DOCODE Lite o DOCODE ASP, los que se detallan más adelante). Además, internamente se implementa un Servlet para comunicar las interfaces con la capa de servicios. Esta implementación podría considerarse una capa adicional, sin embargo por sus características se decidió dejar en la capa Web.

4.1.3. Capa servicio

Dentro del esquema, esta capa es la encargada de disponibilizar los servicios hacia Internet. Representa una interfaz entre los clientes (personas y sistemas) y la capa donde están implementadas las lógicas de negocio.

4.1.4. Capa de lógica de negocio

En esta capa se implementan las acciones y procesos que son parte del negocio. Modelar los procesos de negocio es la parte más importante en desarrollar bien la arquitectura empresarial, ya que este proceso está directamente relacionado con la estrategia del negocio. El proceso de definición de las lógicas es el siguiente:

1. Capturar los requerimientos del negocio.
2. Analizar los requerimientos.
3. Definir la estrategia del negocio respecto de los requerimientos.
4. Modelar el proceso de negocio.

Los requerimientos son capturados y documentados. El paso siguiente es involucrar a los responsables, analizar los requerimientos y definir las estrategias para alcanzar los objetivos definidos en el documento de requisitos. Finalmente, el modelo del proceso de negocio está diseñado para dar una visión global del proceso de negocio completo. Puede ser modelado mediante herramientas de BPM. El proceso de negocio debe considerar todas las entidades relacionadas dentro de él [6].

4.1.5. Capa Persistencia

A grandes rasgos, permite administrar el modelo de datos relacional como si fuera un modelo de objetos. Esta capa es necesaria en la organización para proveer de información al negocio. Esta capa sirve como un puente entre los datos y los procesos de negocio. Para soportar los objetivos de negocio, los procesos recuperan información utilizando aplicaciones propietarias. Cada implementación de persistencia se ha desarrollado bajo su propia arquitectura de referencia. La cual provee una vista de los procesos definida durante la fase de desarrollo. Este proceso tiene una clara demarcación de sus actividades. Por ejemplo, el proceso de recuperar la data puede ser diferente del proceso de llevar la data al negocio [6].

4.1.6. Capa de Datos

Los datos son la parte crítica de la solución [6]. Constituye la mayor parte de la solución, en la cual se debe tener en consideración:

- Redundancia de datos.
- Reutilización de datos.

- Control de acceso.
- Respaldos regulares.

4.2. Implementación Arquitectura DOCODE 2.0

La Figura 4.2 resume las capas del sistema y en ella se puede ver las capas de la arquitectura abiertas a un nivel más de detalle, mostrando algunas de las tecnologías utilizadas.

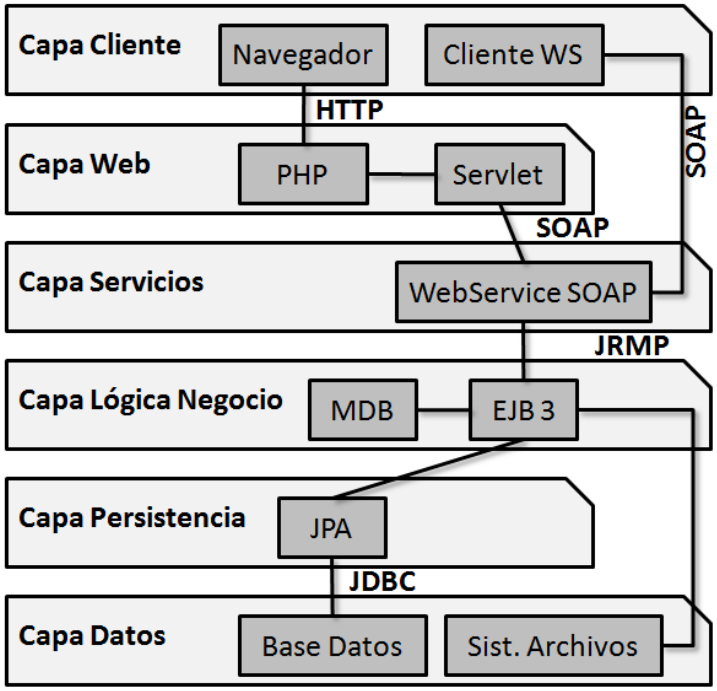

Figura 4.2: Diagrama de capas detallado del sistema DOCODE.
Fuente: elaboración propia

4.2.1. Herramientas utilizadas

Para la construcción se utilizaron herramientas de distribución libre en cada una de las capas.

PHP (PHP Hypertext Pre-processor) Para implementar las funcionalidades del sitio web.

Glassfish ESB (Enterprise Service BUS) Corresponde al BUS de servicios de Glassfish.

Glassfish 3.1.2 Es la versión libre del servidor de aplicaciones JEE implementada por Oracle [29].

PostreSQL Para implementar la base de datos del sistema.

4.2.2. Implementación Capa Cliente

No hay implementaciones en esta capa debido a que por definición consiste en un navegador o en un cliente para el servicio web, el cual debe implementarse en cada sistema externo que se integre a DOCODE.

4.2.3. Implementación Capa Web

Existen dos sistemas desarrollados en PHP para la capa de presentación: DOCODE ASP y DOCODE LITE. Si bien es posible crear otras interfaces para acceder al sistema, esas implementaciones no dependen del resto del proyecto. Y en su implementación se utilizará el servicio expuesto en la capa ESB. En el Capítulo 3 se explica largamente una implementación de esta capa. Las implementaciones actuales utilizan un Servlet para ejecutar las funciones del DOCODE Engine, el cual transforma los parámetros recibidos por el HTTP POST Request en SOAP e invoca el *web service*.

Servlet Como parte del legado del sistema, existe un Servlet que recibe parámetros por POST. Utilizando estos parámetros se puede pasar distintos comandos, los cuales responden con texto plano que tiene formato XML. Existen algunos sistemas que utilizan estas interfaces, constituyendo esa la razón fundamental para no descontinuar por el momento este servicio.

DOCODE ASP Sistema basado en Moodle [22], aborda uno de los problemas más importantes de la comercialización del servicio DOCODE ya que dispone de una interfaz que permite utilizar las funcionalidades del Core de DOCODE en instituciones que no poseen un sistema informático que pueda integrarse utilizando las interfaces especialmente construidas para ese fin. Este sistema cumple las mismas especificaciones que las mencionadas en el Capítulo 3 del sistema DOCODE SaaS. Algunas de las funcionalidades de este sistema son:

- Al ingresar como alumno, puede:
  - Actualizar su registro en el sistema.
  - Acceder a los contenidos de los cursos que tiene dentro de la plataforma.
  - Subir archivos de tareas.
- Al ingresar como profesor, puede:
  - Actualizar su registro en el sistema.
  - Acceder a los contenidos de los cursos que tiene dentro de la plataforma.

- Cargar listas de alumnos.
- Definir tareas con sus parámetros correspondientes.
- Enviar tareas para su análisis en el Core de DOCODE.
- Revisar los resultados del análisis.

- Al ingresar como administrador, puede:
  - Actualizar su registro en el sistema.
  - Crear nuevos cursos.
  - Crear profesores.
  - Administrar usuarios y roles.
  - Ejecutar labores administrativas.

La Figura 4.3 muestra la página de inicio del sistema DOCODE ASP.

Figura 4.3: Página inicio DOCODE ASP.
Fuente: http://asp.docode.cl

DOCODE Lite Este sistema está diseñado para usuarios que desean hacer pruebas en DOCODE sin estar comprometidos con un contrato. Tiene varias restricciones de uso, pero de todas formas el procesamiento de las tareas ocurre dentro del Core de DOCODE. Las funcionalidades que se tiene acceso son:

- Subir hasta tres archivos para revisar en cada envío.
- Ejecutar diariamente hasta 3 veces el servicio DOCODE.
- Revisar los resultados de DOCODE.

- Leer del funcionamiento de los algoritmos.

En la Figura 4.4, se muestra parte de la pantalla de inicio del sistema DOCODE Lite.

Figura 4.4: Página inicio DOCODE Lite.
Fuente: http://lite.docode.cl

4.2.4. Implementación Capa Servicios

El uso de la herramienta ESB ofrece una cantidad importante de funcionalidades que permiten obtener la mejor performance de los procesos y la correcta ejecución de los distintos servicios.

Definición del servicio

El primer paso es definir los objetos que serán parte de la comunicación. Para este fin se escriben los archivos XSD (*XML Schema Definition*) que contienen las definiciones de las estructuras que se envían dentro de los mensajes *SOAP*. En el Anexo B se puede ver el texto completo del XSD mencionado. Se destacan dentro de esta definición las siguientes estructuras:

TaskType Se utiliza para enviar los datos de la tarea, los que principalmente corresponden a identificadores utilizados dentro del sistema que envía los documentos a procesar (desde el origen de datos).

FileType Esta estructura es la que representa un archivo de la tarea y en ella se incluye realmente el contenido del archivo, utilizando el elemento *FileType.data* que es de tipo *hexBinary*.

A continuación, es necesario definir las operaciones que se pueden ejecutar utilizando el servicio. Para eso se define el archivo WSDL (*Web Service Description Language*), el cual incluye además de las estructuras definidas en el XSD, los métodos que se pueden ejecutar. En el Anexo C se puede ver el texto completo del WSDL mencionado. Se destacan las siguientes estructuras del formato:

types En la especificación del servicio este tag es el que contiene las definiciones de los tipos de datos que se aceptan como parte de las llamadas. Hay que destacar que debido a la construcción de este *web service*, dentro de este tag está contenido el archivo completo del XSD mencionado anteriormente. Esto podría ser distinto cuando se usan definiciones directamente dentro del WSDL o cuando se incluyen varios XSD utilizando la siguiente estructura:

```
<types>
   <xs:schema import='archivo.xsd'/>
   <xs:schema import='archivo2.xsd'/>
</types>
```

message Se utiliza este tag para indicar las entradas y salidas de los métodos expuestos. En el siguiente texto, se puede ver el uso del elemento *doQueryResponse* dentro de la definición de *QueryServiceWS doQueryResponse*.

```
<message name='QueryServiceWS_doQueryResponse'>
  <part element='tns:doQueryResponse' name='doQueryResponse'/>
</message>
```

portType Es el tag utilizado para definir los métodos expuestos mediante el servicio. El siguiente texto, se muestra la definicion del port *QueryServiceWS*, el cual es utilizado para definir la llamada mediante la operación llamada *doQuery*, que a su vez utiliza los mensajes *QueryServiceWS doQuery* y *QueryServiceWS doQueryResponse* como entrada y salida del método respectivamente.

```
<portType name='QueryServiceWS'>
 <operation name='doQuery' parameterOrder='doQuery'>
  <input message='tns:QueryServiceWS_doQuery'></input>
  <output message='tns:QueryServiceWS_doQueryResponse'></output>
 </operation>
</portType>
```

binding Este tag especifica los protocolos de comunicación usados. En este caso, no hay protocolos especiales, por lo tanto debería aceptar todas las llamadas.

```
<binding name='QueryServiceWSBinding' type='tns:QueryServiceWS'>
 <soap:binding style='document'
       transport='http://schemas.xmlsoap.org/soap/http'/>
 <operation name='doQuery'>
  <soap:operation soapAction=''/>
```

```
  <input>
   <soap:body use='literal'/>
  </input>
  <output>
   <soap:body use='literal'/>
  </output>
 </operation>
</binding>
```

service Este tag se utiliza para indicar donde está expuesto el servicio y por lo tanto cual es la máquina donde se va a ejecutar. El siguiente texto muestra como el servicio llamado *QueryServiceWS* está disponible en la ruta http://localhost:9090/docode/QueryServiceWS.

```
<service name='QueryServiceWS'>
 <port
      binding='tns:QueryServiceWSBinding'
      name='QueryServiceWSPort'>
  <soap:address
        location='http://localhost:9090/docode/QueryServiceWS'/>
 </port>
</service>
```

Hay que destacar que el uso de un WSDL no indica cuál es la función que realiza un servicio. Sólo ofrece la información necesaria para poder trabajar con él (funciones, mensajes, protocolos, parámetros).

Implementación del servicio

Una vez construído el WSDL, se procede a su implementación utilizando las herramientas para este fin. De las múltiples opciones para implementar el servicio, se decidió utilizar *Java API for XML Web Services (JAX-WS)* [1], ya que esta tecnología está completamente incluída dentro de la plataforma Glassfish. A continuación, se muestran algunos aspectos importantes de la implementación, los que se ven facilitados con la plataforma seleccionada.

Encabezado del servicio para implementar el servicio, basta con agregar algunos *annotations* dentro del código. La anotación *@WebService* permite indicar que esta clase implementa un servicio, mientras que el parámetro *serviceName* indica el nombre con el cual será expuesto el servicio. La anotación *@PermitAll* indica que el servicio puede ser consumido por cualquier cliente que se conecte.

[1] Para más antecedentes, revisar sitio oficial [28].

```
@WebService(serviceName = "QueryServiceWS")
@PermitAll
public class QueryServiceWS {
...
}
```

Métodos del servicio Cualquier método implementado dentro del objeto puede ser expuesto dentro del *web service*. Para ese fin, se deben agregar las siguientes anotaciones:

@WebMethod Indica que el método debe quedar expuesto en el servicio. Además el parámetro *operationName* permite modificar el nombre del método que será expuesto.

@javax.jws.WebResult Utilizando esta anotación junto con el parámetro *name*, permite modificar el nombre que tiene el parámetro de salida dentro del servicio expuesto.

@WebParam Utilizando esta anotación junto con el parámetro *name*, permite modificar el nombre que tiene el parámetro de entrada dentro del servicio expuesto.

A continuación, un ejemplo de cómo debe ser el encabezado de un método llamado *doQuery*, que recibe un parámetro llamado *queryRequest* de tipo *[package].QueryRequestType* y con un parámetro de salida llamado *queryResponse* de tipo *[package].QueryResponseType*. En ambos casos, *[package]* corresponde a *xmlws.docode.cl.service*.

```
@WebMethod(operationName = "doQuery")
@javax.jws.WebResult(name="queryResponse")
public xmlws.docode.cl.service.QueryResponseType
        doQuery(
           @WebParam(name = "queryRequest")
           xmlws.docode.cl.service.QueryRequestType request)
{
...
}
```

Lógica expuesta en el servicio El servicio expuesto por DOCODE 2.0 permite enviar archivos desde cualquier sistema externo al servicio de análisis de originalidad mediante el uso de mensajería SOAP. Este servicio valida la integridad de la llamada y luego compone varias lógicas de negocio dentro de un sólo servicio. De este modo, se aprovechan varias de las ventajas del uso del ESB, con el fin de mantener los servicios con el mayor *timeup* dentro de toda la arquitectura. Se espera que gracias a la composición de servicios que representan las lógicas de negocio, a futuro se puedan desarrollar servicios ad-hoc según las caracteristicas que necesiten los clientes.

4.2.5. Implementación Capa de Lógicas de Negocio

Como ya se comentó anteriormente, la capa de lógicas de negocio implementa los procesos y procedimientos que se necesitan llevar a cabo para poder desarrollar el negocio. En este caso, las lógicas son principalmente los distintos algoritmos que permiten analizar aspectos relevantes para el cálculo del índice de originalidad y por otro lado estan las lógicas que son parte del proceso asociado a la ejecución de esos algoritmos.

Modelo proceso de negocio. Subir archivos.

Existe un proceso principal, el cual está esquematizado en la Figura 4.5. En ella se puede ver las distintas capas interactuando entre sí.

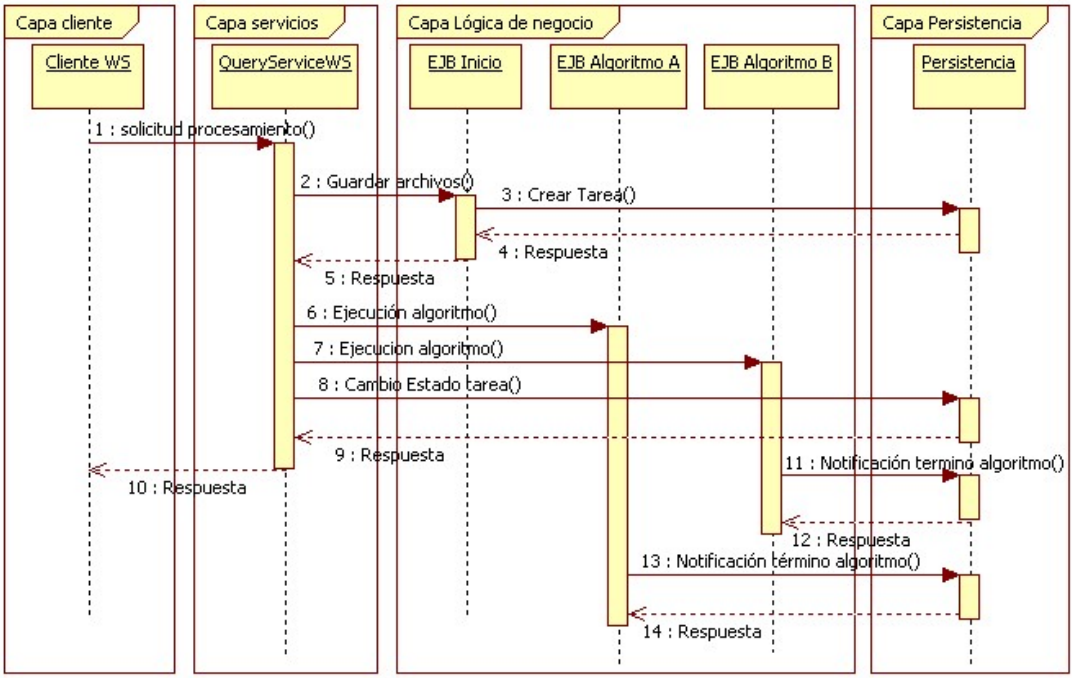

Figura 4.5: Proceso subida de archivos a DOCODE 2.0.
Fuente: elaboración propia

Actores del proceso

Cliente WS Corresponde a una implementación de la capa cliente que consume el *web service*. Desde aquí se gatilla el proceso de subir archivos.

QueryServiceWS Corresponde a la implementación del servicio expuesta en la capa del ESB y está encargada de componer las lógicas de negocio.

EJB Inicio Es parte de la capa de lógica de negocio. Corresponde a un EJB que ejecuta el inicio del proceso, el cual es necesario para poder ejecutar el resto de los algoritmos.

EJB Algoritmo A y B Dentro del proceso, se llaman así a dos algoritmos genéricos, que representan a los distintos algoritmos implementados que calculan los índices de originalidad.

Persistencia Es el módulo que corresponde a la implementación de la capa de persistencia.

Pasos del proceso

1. Solicitud de procesamiento: Representa el momento en que desde algún sistema, se envían los archivos para ser analizados por DOCODE 2.0.

2. Guardar archivos: Es la primera etapa de composición de lógicas mediante la capa de servicios, ya que la implementación del proceso está completamente contenida en la capa de negocio.

3. Crear tabla: El EJB Inicio es capaz de comunicarse con la capa de persistencia para obtener una representación de una tarea dentro de la capa de persistencia.

4. Respuesta: Es un mensaje que viene desde la capa de persistencia indicando que se logró generar la entidad tarea. Al obtener este mensaje, se ejecuta parte de la lógica que consiste en guardar los archivos dentro del *File System*.

5. Respuesta: Es el mensaje que va desde la capa de lógica hacia la de servicios, indicando que el proceso inicial ha finalizado. En este momento, se ejecuta la siguiente parte de la composición de lógicas, ya que el servicio ejecuta en paralelo llamadas para iniciar todos los algoritmos que estén configurados. En este ejemplo, los llamados Algoritmo A y B.

6. Ejecución algoritmo: Es la llamada que inicia la ejecución del algoritmo A, cuya implementación está contenida dentro de un EJB. Hay que hacer notar que el servicio no espera respuesta de esta ejecución, llamando al resto de los algoritmos en paralelo.

7. Ejecución algoritmo: Es la llamada que inicia la ejecución del algoritmo B, cuya implementación está contenida dentro de un EJB. Hay que hacer notar que el servicio no espera respuesta de esta ejecución, llamando al resto de los algoritmos en paralelo.

8. Cambio estado tarea: Una vez ha terminado de ejecutar todos los algoritmos, cambia el estado de la tarea dentro de la capa de persistencia. Esto permite saber que la tarea todavía no ha sido procesada.

9. Respuesta: La capa de persistencia responde favorablemente al cambio de estado. En este momento, el servicio puede terminar las transacciones pendientes y genera el mensaje de respuesta hacia el cliente.

10. Respuesta: Este mensaje representa el término de la ejecución del servicio propiamente tal, sin embargo las lógicas de negocio quedan funcionando en *background*.

11. Notificación de término de algoritmo: Al finalizar el procesamiento del algoritmo B, este informa que ha finalizado. Esto permite sincronizar el proceso de finalización.

12. Respuesta: La capa de persistencia responde favorablemente al cambio de estado del algoritmo. En este momento termina la ejecución del algoritmo B.

13. Notificación de término de algoritmo: Al finalizar el procesamiento del algoritmo A, este informa que ha finalizado. Esto permite sincronizar el proceso de finalización.

14. Respuesta: La capa de persistencia responde favorablemente al cambio de estado del algoritmo. En este momento termina la ejecución del algoritmo A.

Observaciones

- La separación de capas permite que las lógicas de negocio puedan ser reutilizadas desde distintos servicios.
- La flexibilidad y atomicidad de la capa de lógica de negocio permite componer nuevos servicios a partir de los ya existentes.
- La capa de servicios permite hacer ejecuciones síncronas y asíncronas. De esta forma es posible paralelizar la ejecución de los algoritmos de análisis, minimizando el tiempo de respuesta para el cliente que envia las tareas.
- La ejecución de los algoritmos A y B es completamente arbitraria y podrían considerarse más algoritmos con otros procesos más complejos.
- Se espera que en una etapa posterior de desarrollo, se pueda integrar mediante un motor de BPM.

Modelo proceso de negocio. Cerrar proceso análisis.

El cierre del proceso de análisis es completamente asíncrono respecto de la solicitud de análisis. Por eso se analiza como un proceso aparte. El objetivo prncipal de este proceso es componer los

resultados generados por cada análisis de manera independiente a fin de facilitar la posterior recuperación de resultados. La Figura 4.6 muestra el proceso de cierre que busca componer la respuesta que se entregará a los clientes.

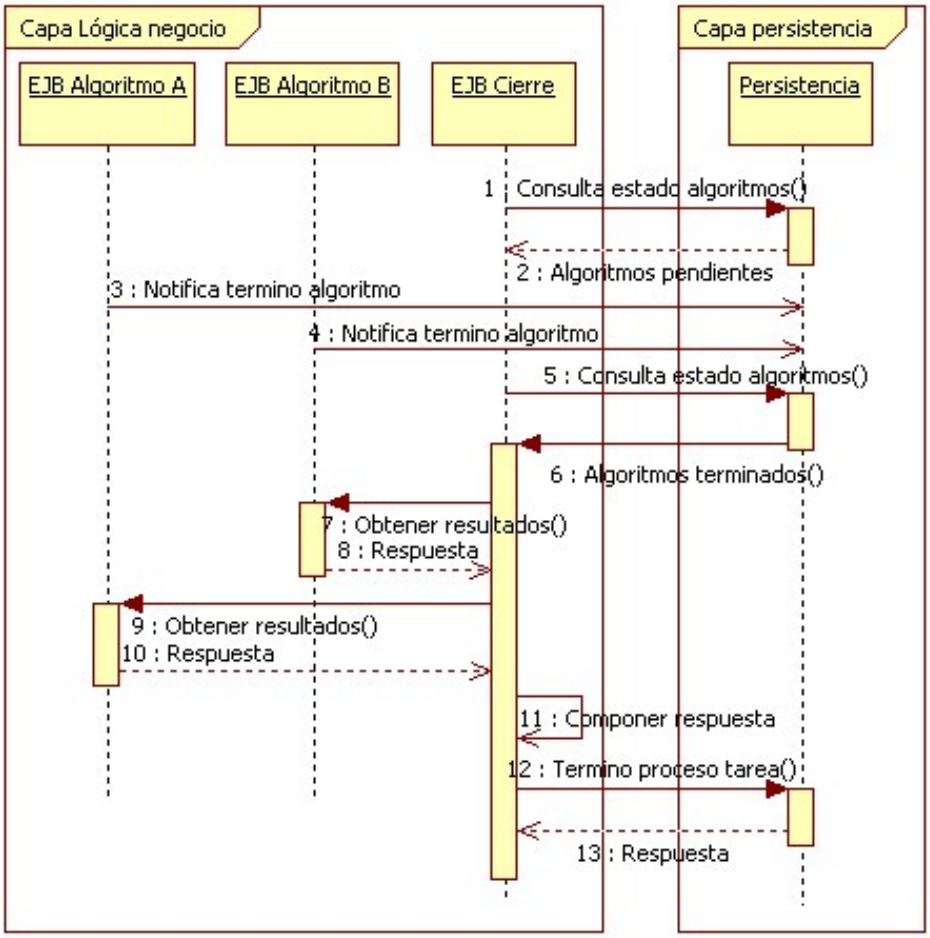

Figura 4.6: Proceso cierre análisis de documentos de DOCODE 2.0.
Fuente: elaboración propia

Actores del proceso

EJB Algoritmo A y B Dentro del proceso, se llaman así a dos algoritmos genéricos que representan a los distintos algoritmos implementados que calculan los índices de originalidad. Su rol es este proceso es el de informar a la capa de persistencia que terminaron de procesar y que los resultados de cada tarea están listos.

EJB Cierre Es parte de la capa de lógica de negocio, corresponde a un EJB que ejecuta el cierre del proceso, el cual está consultando cada cierto tiempo por las tareas pendientes y de ellas,

el estado de sus algoritmos. Tiene la capacidad de componer los archivos de resultados y consolidarlos en uno sólo.

Persistencia Es el módulo que corresponde a la implementación de la capa de persistencia.

Pasos del proceso

1. Consulta estado algortimos: Esta llamada representa una solicitud que no compone resultado.
2. Algoritmos pendientes: La capa de persistencia informa que todavía quedan algoritmos por terminar, por lo tanto el proceso finaliza.
3. Notifica término de algoritmo: El proceso del *EJB algoritmo A* notifica a la capa de persistencia que ha finalizado.
4. Notifica término de algoritmo: El proceso del *EJB algoritmo B* notifica a la capa de persistencia que ha finalizado.
5. Consulta estado algortimos: Esta llamada representa una solicitud que si obtendrá los resultados que gatillan la generación de la respuesta.
6. Algoritmos terminados: Es la respuesta que envía la capa de persistencia indicando que los algoritmos relacionados ya terminaron su ejecución.
7. Obtener resultados: El proceso de cierre chequea la respuesta generada por el Algoritmo B.
8. Respuesta: El Algoritmo B responde con los resultados.
9. Obtener resultados: El proceso de cierre chequea la respuesta generada por el Algoritmo A.
10. Respuesta: El Algoritmo A responde con los resultados.
11. Componer respuesta: Es un proceso interno, que arma la respuesta que será entregada después al cliente, se encarga de ajustar el formato y dejar una respuesta procesada.
12. Término proceso tarea: Una vez que término la composición, se notifica a la capa de persistencia que el proceso ha finalizado.
13. Respuesta: la capa de persistencia informa que se guardaron los cambios y por lo tanto el proceso puede finalizar.

Observaciones

- Los pasos 1 y 2 pretenden mostrar que el *EJB Cierre* está ejecutandose todo el tiempo a la espera de que finalicen los algoritmos de alguna tarea pendiente.
- Los pasos 3 y 4 sólo buscan representar el término de la ejecución de los algoritmos, y no está garantizado que ocurran en ese orden.
- Hay que hacer notar que en este proceso sólo actúan las capas de lógica de negocio y persistencia. Esto permite que la ejecución sea transparente para los clientes y los demás servicios.

Modelo proceso de negocio. Recuperar resultados análisis.

El proceso de recuperación de resultados del análisis es gatillado por los clientes, ya que por definición, no hay una interfaz que permita responderles de manera transparente. La Figura 4.7 muestra la interacción que ocurre entre los distintos actores para rescatar los resultados del análisis de originalidad.

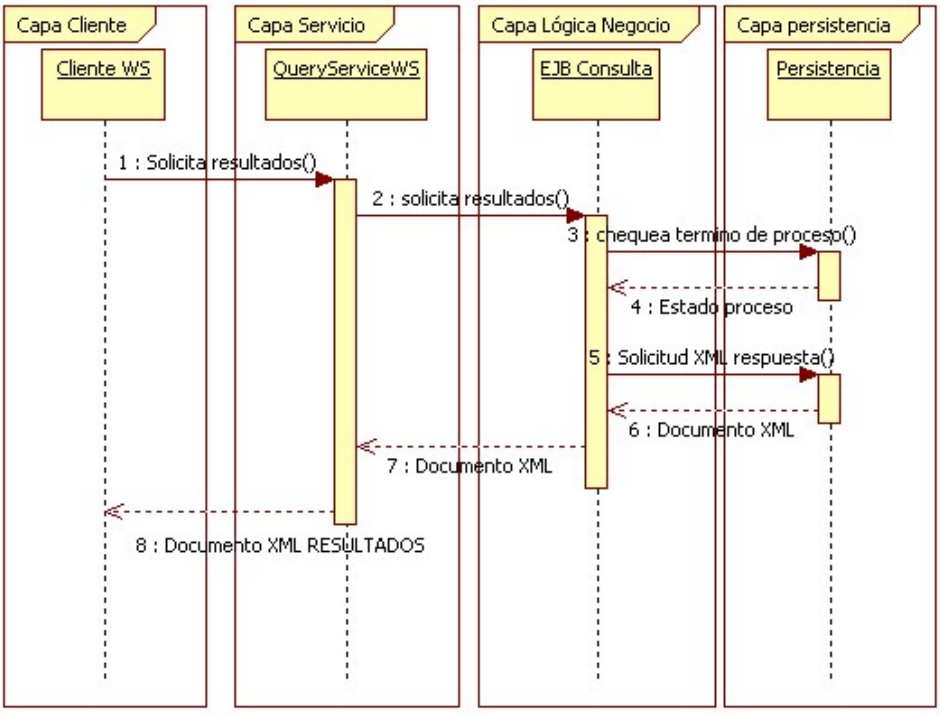

Figura 4.7: Proceso consulta de resultados de análisis de originalidad.
Fuente: elaboración propia

Actores del proceso

Cliente WS Corresponde a una implementación de la capa cliente que consume el *webservice*, desde aquí se gatilla el proceso de solicitar resultados.

QueryServiceWS Corresponde a la implementación del servicio expuesto en la capa del ESB y encargado de componer las lógicas de negocio.

EJB Consulta Es la implementación del EJB que permite rescatar los datos procesados desde la capa de persistencia.

Persistencia Es el módulo que corresponde a la implementación de la capa de persistencia.

Pasos del proceso

1. Solicitud de resultados: Es el primer paso. Representa el momento en que desde algún sistema externo se solicita el resultado del análisis de originalidad de los archivos analizados por DOCODE 2.0.
2. Solicita resultados: Desde la capa de servicios, se ejecuta una llamada hacia la capa de lógica solicitando los resultados del análisis.
3. Chequea término proceso: Se valida contra la capa de persistencia si la tarea ha terminado.
4. Estado proceso: Es un mensaje que viene desde la capa de persistencia indicando que se ha terminado de procesar la tarea. En caso de una respuesta positiva, se continua el proceso.
5. Solicitud XML respuesta: Si el proceso principal ha concluído con la tarea, entonces se solicita a la capa de persistencia los resultados obtenidos.
6. Documento XML: Corresponde a la respuesta que genera la capa de persistencia hacia la de lógica de negocio.
7. Documento XML: Una vez obtenida la respuesta de la capa de persistencia, se modifica su estructura para ajustarla a la respuesta de la capa de servicios.
8. Documento XML RESULTADOS: Es la respuesta que obtiene el cliente como respuesta a la solicitud de análisis de originalidad.

Observaciones

- Es posible que al solicitar la respuesta, el paso 4 responda que no ha finalizado el procesamiento de la tarea, en cuyo caso no se ejecuta el paso 5 y el EJB Consulta genera un XML indicando que no ha finalizado el proceso.

- El *EJB consulta* debe responder lo más rápidamente posible. Es por esta razón que toda la composicón de resultados debe quedar finalizada en algún paso previo.

Proyecto DocodeLibrary

Existe dentro de DOCODE 2.0 un proyecto especial. Corresponde a una librería, en donde se juntan los desarrollos comunes a toda la capa de lógicas de negocio. La Figura 4.8 muestra los packages que componen la librería.

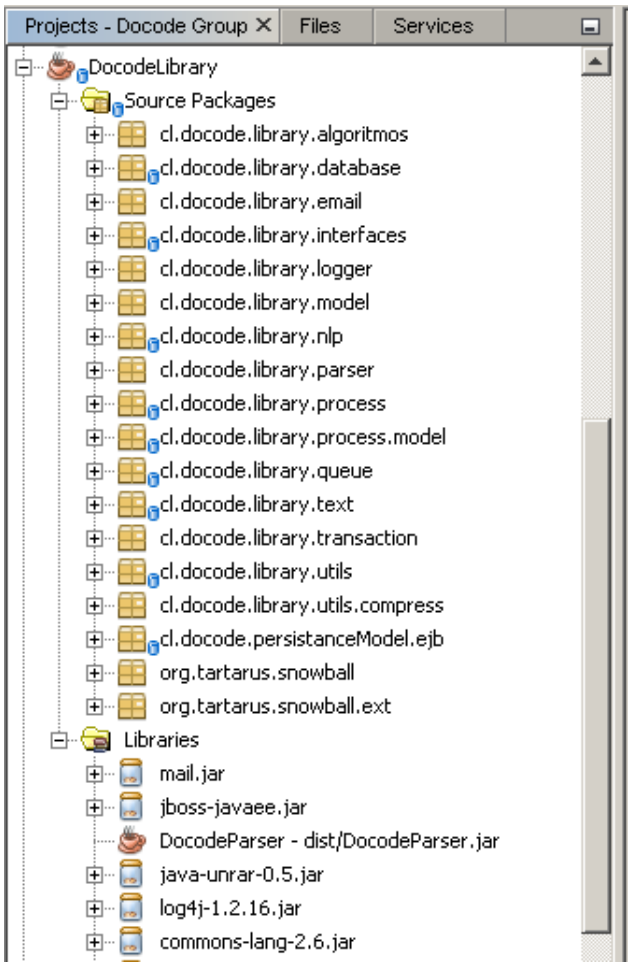

Figura 4.8: Estructura la libraría de DOCODE 2.0.
Fuente: elaboración propia

Se debe comentar los diferentes packages.

cl.docode.library.algoritmos Algunos objetos comunes a la implementación de varios algoritmos.

cl.docode.library.database Implementa una interfaz de comunicación con la base de datos permitiendo integrar consultas fácilmente dentro del código.

cl.docode.library.email Una librería que permite enviar correos electrónicos.

cl.docode.library.interfaces Especifica las interfaces que deben cumplir los algoritmos de análisis para ser incluídos dentro de la plataforma.

cl.docode.library.logger Administración de logs de la aplicación.

cl.docode.library.model Algunos objetos comunes que modelan estructuras en algunos de los algoritmos.

cl.docode.library.nlp Una interfaz para utilizar las funciones de OpenNLP.

cl.docode.library.parser Permite extraer como texto plano la información de documentos con formato como: doc, pdf y xml.

cl.docode.library.process.* Actualmente se alojan varios algoritmos para administrar la comunicación entre algoritmos y algunas clases que modelan esas lógicas.

cl.docode.library.queue Librería para enviar mensajes a las colas y Topics JMS.

cl.docode.library.text Pequeña libraría para determinar distancias entre dos cadenas de texto.

cl.docode.library.transaction Librería deprecada, se mantiene por razones de compatibilidad.

cl.docode.library.utils Varias funciones de adminsitración de fechas y de Strings.

cl.docode.library.utils.compress Clases para administrar archivos ZIP y RAR.

cl.docode.persistanceModel.ejb Contiene una interfaz remota para utilizar el EJB del proyecto DocodePersistenceModel, el cual centraliza la comunicación mediante la capa de persistencia hacia la base de datos.

org.targus.snowball.* Pertenece a una librería de *stemming* llamada Snowball, se debe integrar directamente desde el código fuente.

Algoritmos DOCODE

Para implementar los algoritmos que constituyen la lógica del negocio de DOCODE, se utilizan principalmente: técnicas probabilisticas, de entrenamiento asistido, otros no asistidos y de Procesamiento de Lenguaje Natural (en inglés, NLP). Con el objetivo de lograr efectivamente la ejecución de estas heurísticas y algoritmos, se evaluaron diferentes *frameworks*. De todos ellos, finalmente, se utiliza la librería distribuida por la Apache Software Foundation, en el proyecto OpenNLP, la que consiste, básicamente, en una máquina de aprendizaje para procesamiento de lenguaje natural. Ofrece las tareas más comunes de NLP: tokenización, segmentación, marcas de discurso, extracción

de entidades, etc. Otra ventaja de esta herramienta es que por estar escrita en Java se integra en forma natural al resto del sistema.

Estructura de los algoritmos Cada algoritmo fue encapsulado como si fuera parte de una librería, la cual se integró dentro de la plataforma JEE utilizando la especificación EJB3. Cada proyecto presenta una estructura como en la Figura 4.9. En ella se aprecia las diferentes carpetas del proyecto FastDocode (Fuentes, Librerías, EJBs, Recursos varios). El resto de los algoritmos tiene básicamente la misma estructura, es por esta razón que para la explicación se utiliza sólo este proyecto como ejemplo.

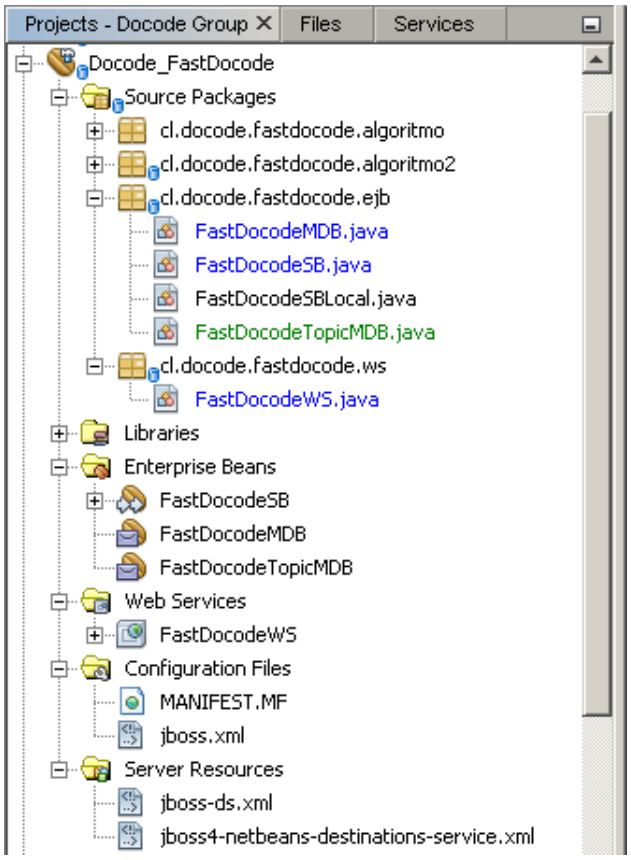

Figura 4.9: Estructura de los proyectos de algoritmos en DOCODE 2.0.
Fuente: elaboración propia

Observaciones Hay varios aspectos importantes, sin embargo, se destacan los siguientes:

FastDocodeMDB Corresponde a la implementación del MDB, el que está configurado para responder a los mensajes de la cola llamada *jms/docode/FastDocodeQueue*. En el Anexo D , se puede ver el código que se ejecuta al recibir un mensaje y ejecuta la llamada al EJB. Este código tiene muchas partes importantes:

Anotaciones Como por ejemplo @EJB, la cual instancia la interfaz local del EJB y la deja disponible para su utilización. También las de @MessageDriven, que generan todo lo necesario para vincular el MDB con la cola JMS respetiva.

onMessage Es el método que se ejecuta siempre que llega un mensaje a la cola JMS. Dentro de él está la llamada al EJB llamado FastDocodeSB.

FastDocodeSB Corresponde a la implementación del EJB, el cual está definido como *Stateless*. En el Anexo E se puede ver un fragmento del código fuente utilizado para integrar cada algoritmo dentro del contenedor de aplicaciones. Cada una de estas integraciones utiliza dos métodos.

execute Es el método que queda expuesto del EJB mediante la interfaz local para ser ejecutado por el MDB. Dentro de él, está la llamada a la función que llama la implementación integrada del código original. El código correspondiente es el siguiente:

```
    @Override
    public void execute(String token) {
...
        executeFastDocode(
             token,
             path_fd_tareas,
             path_fd_fuentes,
             3);
    }
```

executeFastDocode Es el método interno del EJB (no queda expuesto) que ejecuta realmente la llamada a la librería que contiene el algoritmo. Estos dos métodos son necesarios, porque permiten tener una interfaz común en todos los algoritmos y una llamada interna ajustada a los parámetros de cada algoritmo. El siguiente código es un ejemplo de este método:

```
    public boolean executeFastDocode(
                       String token,
                       String path_fd_tareas,
                       String path_fd_fuentes,
                       int idAlgoritmo) {
        ...
        try {
        //llamada a la librería
          cl.docode.fastdocode.algoritmo2.fastdocode.main(
               new String[]{
                    path_fd_tareas,
                    path_fd_fuentes});
```

```
        } catch (Exception ex) {
            System.err.println(
                "Error en Fast Docode..."+ex);
        } finally {
        ...
        }
        return true;
    }
```

FastDocodeTopicMDB Corresponde a la implementación de un MDB, conectado a un TOPIC llamado *jms/docode/SuscripcionTopic*, el cual mediante un patrón de publicador y subscriptor, es capaz de hacer que todos los algoritmos inscritos reciban y ejecuten el mismo mensaje, dejando esta administración al contenedor de aplicaciones.

WebService Se implementó una interfaz para ejecutar un algoritmo en forma aislada. Esto con el fin de poder hacer reprocesos y para poder generar servicios personalizados con la ejecución de un sólo algoritmo. Este servicio despliega dos implementaciones:

Síncrono permite ejecutar directamente el EJB y de esta forma esperar la respuesta del algoritmo.

Asíncrono al invocar este servicio, se deja un mensaje en la cola de ejecución correspondiente, generando una llamada al EJB en el futuro, cuando se atienda el mensaje.

La Figura 4.10 muestra un esquema de como queda encapsulado cada algoritmo de DOCODE dentro de la plataforma JEE. Aquí se puede ver la interacción entre los distintos MDB y el EJB que desacopla finalmente la implementación con una interfaz homogénea. Además se puede apreciar la interfaz que quedó disponible en dos *web services*, uno síncrono que invoca directamente el EJB y otro asíncrono que deja un mensaje en la cola del algoritmo para su posterior ejecución.

4.2.6. Implementación Capa Persistencia

La capa de persistencia se implementa de forma tal que se debe hacer el modelo orientado a objetos utilizando Entity Beans, los que después son persistidos en la capa de datos mediante las librerías diseñadas para ese fin.

Entity Bean Como se mencionó anteriormente, los entity beans se encargan de representar el modelo de datos relacional como si fuera un modelo de objetos. El Anexo F contiene la implementación del entity llamado DocArchivo, el cual es utiliza como ejemplo para mostrar las partes más importantes de esta implementación.

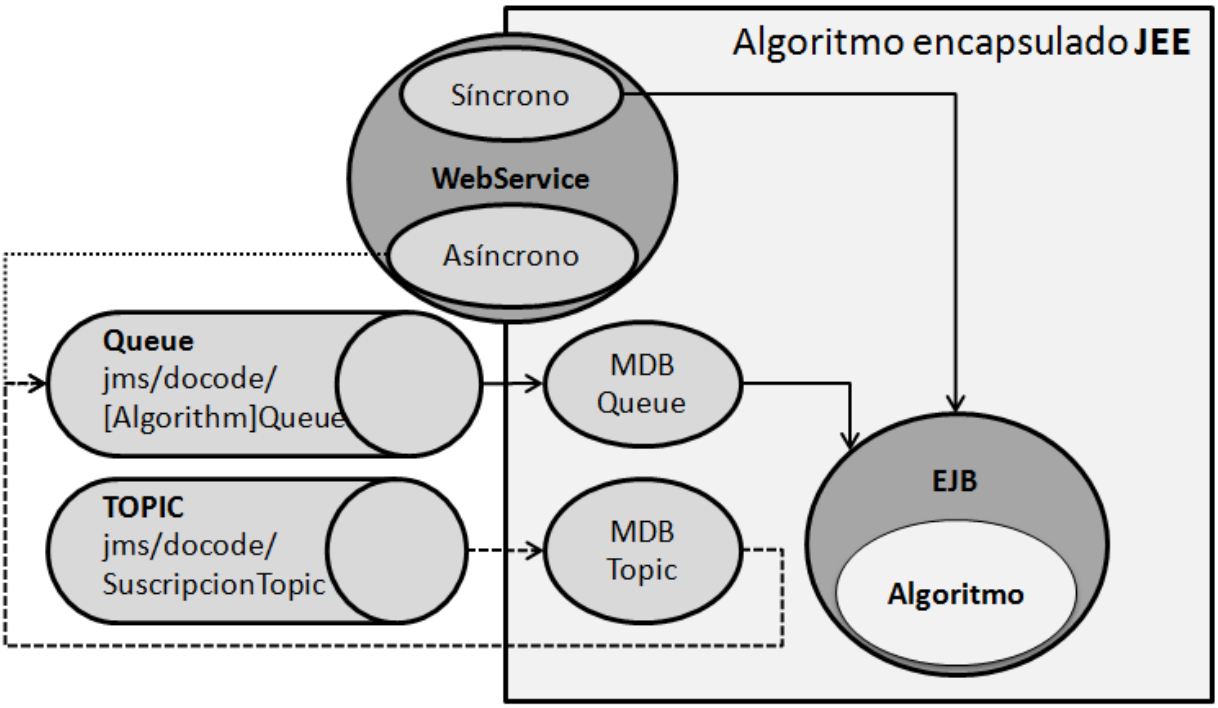

Figura 4.10: Figura con encapsulamiento de algoritmos DOCODE.
Fuente: elaboración propia

Anotaciones Una vez más las anotaciones en el código se vuelven muy relevantes, ya que permiten realizar directamente el mapeo del objeto con la base de datos. Además permiten generar algunas restricciones sobre el modelo a fin de garantizar la integridad de la información. Las anotaciones más importantes son:

- @Entity: Se usa para indicar que esta clase corresponde a un entity bean.
- @Table: Se usa para indicar la referencia a la tabla mediante el parametro name. En caso que no esté explicitamente, se asume que el nombre de la clase es el mismo que la tabla.
- @NamedQueries: Permite agrupar todas las consultas a la base de datos que están predefinidas.
- @NamedQuery: Permite escribir una consulta en EJBQL y asignarle un nombre.
- @Id: Para indicar la llave primaria de la tabla.
- @Column: Permite mapear una columna de la base de datos con un atributo de la clase.
- @OneToMany y @ManyToOne: Permiten administrar las relaciones 1 a N y N a 1 del modelo de datos relacional.

A modo de ejemplo, se muestra un fragmento del código mencionado.

```
@Entity
@Table(name = "doc_archivos")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "DocArchivos.findAll",
                query = "SELECT d FROM DocArchivos d")})

public class DocArchivos implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @Basic(optional = false)
    @Column(name = "arc_id")
    private BigDecimal arcId;

    /* Anotaciones, definiciones, getters y setters*/
    ...
    }
```

persistence.xml Es el archivo donde se configura toda la capa de persistencia. Aquí se indican los parámetros de los objetos que persisten y como lo hacen. El Anexo G muestra un ejemplo del código XML de un archivo persistence.xml, desde el cual se destacan los siguientes campos:

- persistence-unit: Aquí se declara la unidad de persistencia, el parametro más importante es *name*, ya que declara el nombre de la unidad al cual se le agregan los entities.
- provider: Indica cual es el proveedor de persistencia
- jta-data-source: Dice cual es el nombre del datasource que administra la conexión a la base de datos registrado en el JNDI.
- class: Permite identificar los entity que están sometidos a este contexto de persistencia.
- exclude-unlisted-classes: Es un parámetro que indica si todos los entities del proyecto son persistidos o sólo los que están explicitamente agregados usando el parámetro *class*.

A continuación un ejemplo del archivo persistence.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<persistence version="1.0"
  xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xsi:schemaLocation=
          "http://java.sun.com/xml/ns/persistence
           http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">

  <persistence-unit
       name="Docode_PersistenceModelPU"
       transaction-type="JTA">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>java:/docode_ds</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
  </persistence-unit>

</persistence>
```

4.2.7. Implementación Capa Datos

Como se mencionó anteriormente, la capa de datos provee de implementaciones que son completamente independientes del desarrollo del proyecto, pero que las utiliza por que son parte de los estándares de uso en las plataformas JEE. El Anexo H muestra un ejemplo de un archivo de configuración de datasources, lo que resulta en definitiva lo más importante a mencionar dentro de esta capa, además de la base de datos relacional.

DataSources Para configurar correctamente los datasources, es necesario poseer todos los parámetros de configuración para acceder a la base de datos. Los parámetros obligatorios se enumeran a continuación:

- connection-url: Se utiliza para indicar la forma que tiene el string de conexión a la base de datos.
- driver-class: Se necesita para decir la clase que implementa el driver que se utiliza para la conexión.
- user-name: El nombre de usuario que tiene acceso a la instancia de la base de datos.
- password: La clave para acceder a la instancia con el usuario indicado.

Existen otros parámetros que también son importantes, pero corresponden a la configuración del datasource y no de la conexión a la base de datos, estos son:

- Tipo Datasource: Se indica explicitamente con el tipo de tag que se utiliza, que puede ser:
  - local-tx-datasource: Para usar una conexión transaccional con la base de datos.

    - no-tx-datasource: Para una simple conexión no administrada.
    - xa-datasource: Para una conexión donde se utiliza un driver XA, que permite transacciones distribuídas.

- min-pool-size y max-pool-size: Permiten indicar la cantidad mínima y máxima de conexiones que acepta el datasource respectivamente.
- idle-timeout-minutes: Permite indicar un tiempo de espera para mantener la conexión abierta.

Un ejemplo de lo indicado anteriormente queda explicitado en el siguiente fragmento de código:

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>

    <local-tx-datasource>
        <jndi-name>docode_tx_ds</jndi-name>
        <connection-url>jdbc:postgresql://localhost/engine</connection-url>
        <driver-class>org.postgresql.Driver</driver-class>
        <user-name>usuario</user-name>
        <password>password</password>
        <min-pool-size>5</min-pool-size>
        <max-pool-size>50</max-pool-size>
        <idle-timeout-minutes>10</idle-timeout-minutes>
    </local-tx-datasource>

</datasources>
```

Modelo de datos Es importante mostrar el modelo de datos y hacer algunos alcances al mismo. El Anexo I muestra el script de creación de la base de datos.

Hay que indicar un punto importante y es que el modelo de datos actual es completamente funcional y está creado de forma ajustada a la necesidad de DOCODE 2.0 de administrar pocos datos. Es por esta razón que el modelo es bastante pequeño en comparación con los de otros procesos de negocio más complejos, sin embargo es suficiente para las tareas operativas actuales.

En una siguiente etapa se pretende modificar el modelo a fin de agregar mejor soporte y gestión de los datos que permitan idealmente utilizar algún esquema de mejora continua.

Capítulo 5

Módulo de detección de citas bibliográficas

Como se presentó anteriormente, el sistema DOCODE tiene la necesidad de un módulo detector de citas bibliográficas, debido a que cuando estas no representan plagio, sino muy por el contrario, enaltece al creador original y destaca su trabajo como algo relevante. También se definió en el Capítulo 2 una forma de clasificar las citas. Esa misma caracterización se utiliza para presentar este capítulo.

5.1. Librerías del módulo

Se implementaron algunas partes que se utilizan en todo el módulo, estas piezas de software se utilizan como librerías y permiten ahorrar trabajo dentro de otras piezas a desarrollar.

5.1.1. Librerías Parser de Documentos

Se desarrollo una interfaz para recuperar como texto plano, el contenido de documentos con formato. Esta interfaz es genérica y simplifica la codificación aislando el uso de librerías externas que permiten leer documentos con formato. El siguiente código muestra la interfaz del *Interprete*.

```
package cl.docode.quotation.documentParser;
import java.io.File;

public abstract class Interprete {
    protected File file;
    protected String document_text;

    public abstract String getText();

    public void setFile(File file){
        this.file=file;
        this.document_text = "";
    }
```

```
}
```

Para facilitar el uso de la interfaz, se implementó una *Factory* llamada *InterpreteFactory*, cuyo código se muestra a continuación.

```
package cl.docode.quotation.documentParser;

import java.io.File;

public class InterpreteFactory {
    public static Interprete newInstance(File file){
        Interprete interprete = null;
        if(file.getName().endsWith(".doc")){
            interprete= new InterpreteDOC();
        }
        if(file.getName().endsWith(".docx")){
            interprete= new InterpreteDOCX();
        }
        if(file.getName().endsWith(".txt")){
            interprete= new InterpreteTXT();
        }
        if(file.getName().endsWith(".pdf")){
            interprete= new InterpretePDF();
        }
        interprete.setFile(file);
        return interprete;
    }
}
```

Mediante esta factoría, utilizando las librerías POI y PDFBox, se implementaron distintos parser que cumplen con recuperar los textos desde los archivos con formato PDF, DOC y DOCX. El Anexo K contiene la implementación que permite extraer el texto desde un archivo DOC, se incluyen tres métodos de extracción para dejar la respuesta con algunas marcas que permitan recuperar las marcas estilográficas.

La Figura 5.1 muestra el diagrama de clases del package *cl.docode.quotation.documentParser*. Aquí se ve el factory que se utiliza para obtener las instancias del *Interprete* que está preparado para extraer la información de documentos con formato: *DOC*, *DOCX*, *TXT* y *PDF*. La clase *MarcaInterprete* permite administrar las marcas tipográficas.

5.1.2. Librerías Procesamiento Lenguaje Natural

Se desarrollo una pequeña interfaz que permite utilizar las librerías del proyecto OpenNLP. Esto detrás de una interfaz genérica que puede ser implementada por otras liberías de manejo de lenguaje natural. El siguiente código muestra la interfaz de *DocodeNLP*.

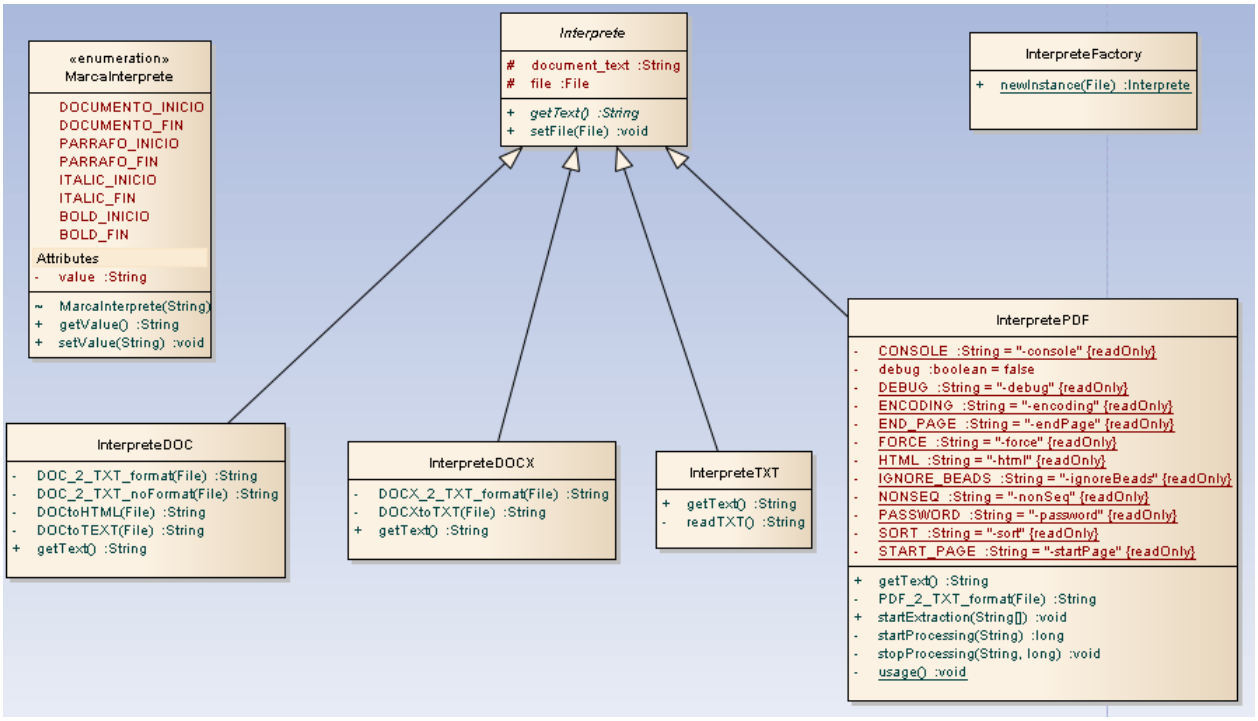

Figura 5.1: Diagrama de clases del package *cl.docode.quotation.documentParser*.
Fuente: Elaboración propia

```
package cl.docode.library.nlp;
import java.util.ArrayList;

public abstract class DocodeNLP {

    /**
     * Funcion para tokenizar un texto.
     * @param texto El texto que se quiere tokenizar
     * @return Arreglo con todos los tokens (cada palabra)
     */
    public abstract ArrayList<String> wordTokenizer(String texto);

    /**
     * Funcion para separar oraciones.
     * @param texto Texto completo que se va a separar
     * @return Arreglo con oraciones del texto
     */
    public abstract ArrayList<String> sentenceTokenizer(String texto);

    /**
     * Funcion para marcar POSTag
     * @param sentences Arreglo con las palabras de una oracion que se desea analizar
     * @return Arreglo con las marcas
     */
    public abstract ArrayList<String> posTag(String sentences[]);
```

```
    /**
     * Funcion para hacer stem
     * @param palabra La palabra que se quiere cortar
     * @return el stem de la palabra
     */
    public abstract String stem(String palabra) ;
}
```

En el Anexo L se muestra una implementación de la interfaz utilizando OpenNLP. En esa implementación se dejó pendiente el uso de stem, ya que no está soportado por OpenNLP. Sin embargo, despliega un mensaje que recomienda utilizar otra implementación para este caso.

Esta interfaz permitió además ampliar las funcionalidades del módulo agregando soporte para stem mediante el uso de la librería abierta *Snowball* [1] y la recuperación del código implementado dentro de otro de los algoritmos de DOCODE. También se agregó una implementación que utiliza expresiones regulares que cubre las funciones de tokenización.

5.1.3. Librería interna

Se desarrolló un package que es usado como librería, el cual contiene funciones utilitarias que permiten administrar las marcas sobre el texto como si fueran coordenadas.

En la Figura 5.2, se muestra el diagrama de clases del package *cl.docode.quotation.utils*. En él, se puede ver que cada coordenada es de uno de los tipos disponibles dentro de la enumeración *TipoCoordenada*. Además se ve la clase *ReportVerbs* que es usada para administrar la lista de verbos de reporte y la clase *Documento* que contiene el texto extraído junto con una lista de coordenadas.

Tipos de Coordenadas Dentro de este módulo, al analizar el texto, se generan marcas llamadas *Coordenadas*, las que pueden ser de los siguientes tipos:

Entidad Corresponde al sujeto del cual se habla. Generalmente es el nombre de la persona de la cual se extrae la cita.

Verbo de reporte Corresponde a un verbo que esta definido como parte de la estructura de una cita. El Anexo J muestra la lista de verbos de reporte considerados como parte de una cita bibliográfica.

Cita Es un texto generalmente entre comillas o con alguna marca tipográfica, como negritas, cursivas o subrayado.

[1] Para más referencias revisar [13].

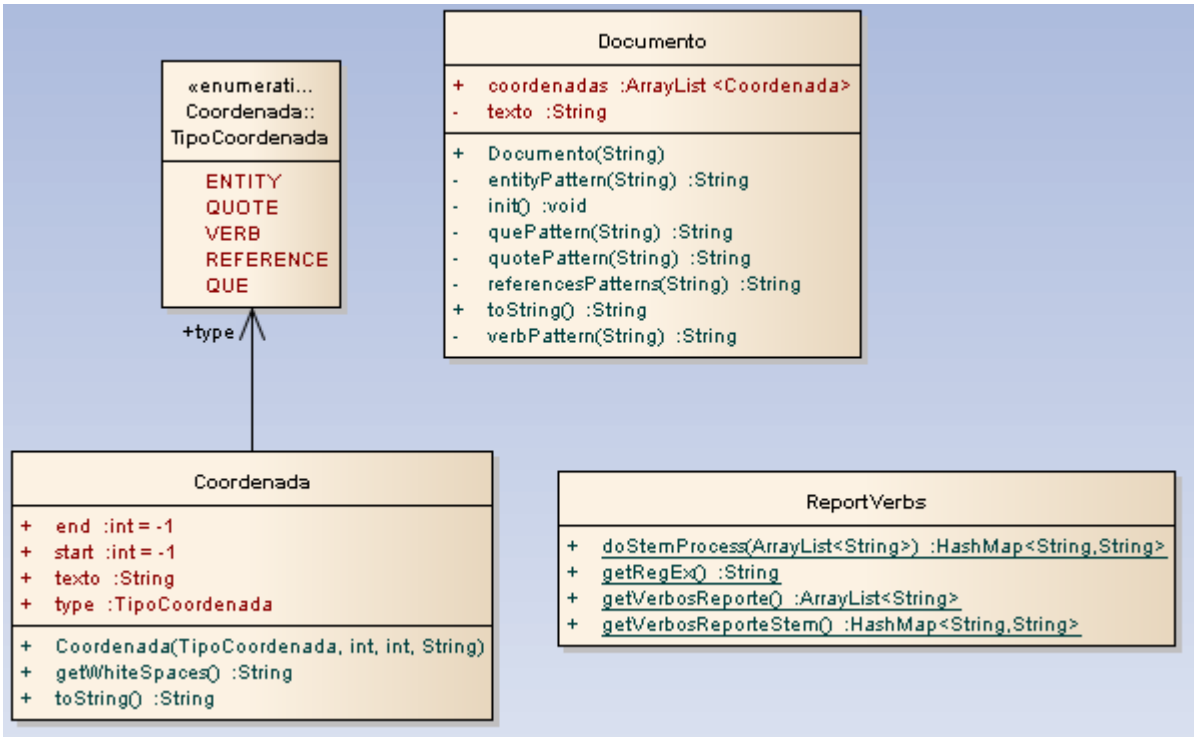

Figura 5.2: Diagrama de clases del package *cl.docode.quotation.utils*.
Fuente: Elaboración propia

Referencia Corresponde a una marca explicita de referencia dentro del texto hacia un documento o autor específico. Se distingue por su estructura y por su uso para identificar citas.

Que La palabra *QUE* pasa a tener una gran relevancia dentro de la expresividad de las citas, ya que actúa como un conector entre la entidad y la cita bibliográfica.

Estas coordenadas son las que se analizan posteriormente para encontrar los patrones que permiten detectar y clasificar las citas bibliográficas.

5.2. Implementación del módulo

En términos generales, cada algoritmo que detecta un tipo de cita específico corresponde a una especificación de un algoritmo general. Se implementó el package *cl.docode.quotation.algorithms* con los patrones de búsqueda para cada tipo de cita. Sin embargo, existe una clase principal (*QuoteDetection*), la cual es extendida por el resto de las implementaciones que refinan el funcionamiento.

La Figura 5.3 muestra el diagrama de clases del package *cl.docode.quotation.algorithms*, en donde se puede ver que existe un Factory para obtener una instancia del detector de citas que se necesite. Además se puede ver que cada instancia del detector extiende a la clase *QuoteDetection*.

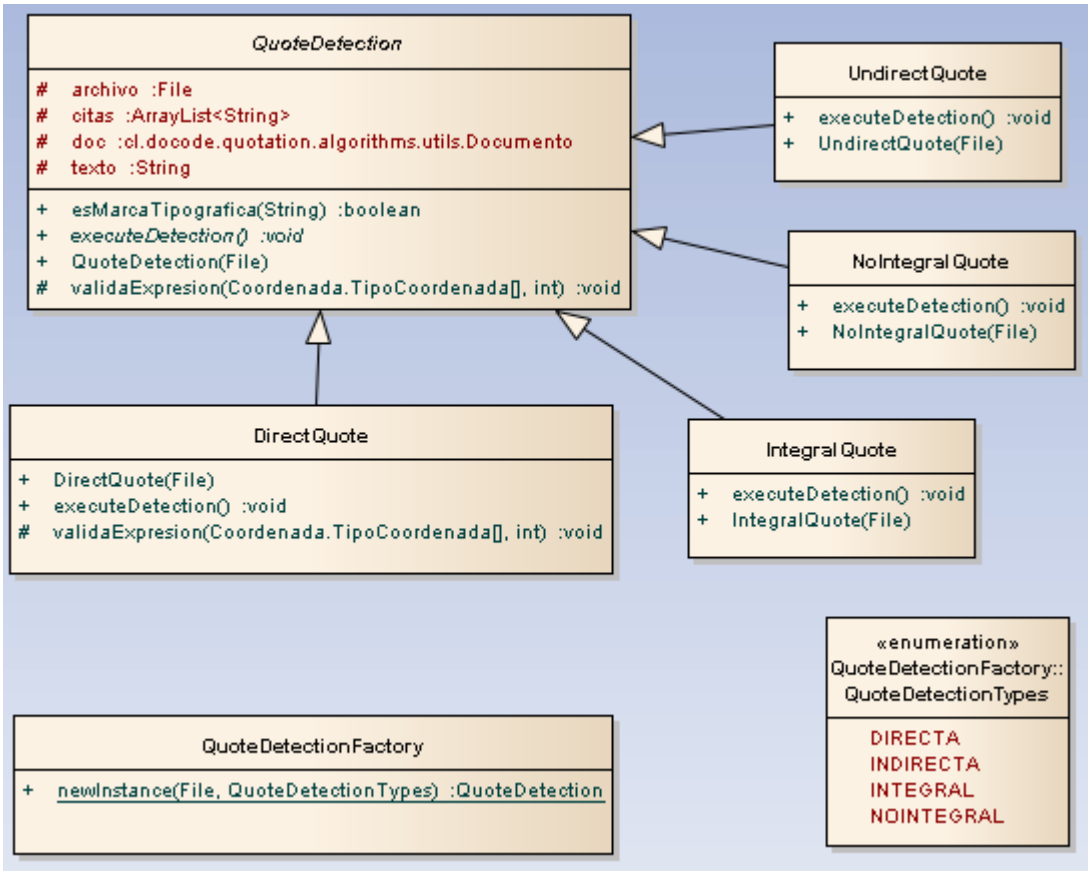

Figura 5.3: Diagrama de clases del package *cl.docode.quotation.algorithms*.
Fuente: Elaboración propia

5.2.1. Expresiones Regulares

Debido a la estructura que tienen las citas, se consideró una implementación utilizando expresiones regulares. A pesar que las expresiones regulares generalmente se utilizan dentro del contexto de análisis de texto y no del de detección de entidades, esta solución cumple con lo esperado. La única expresión regular que tiene un trabajo adicional es la que se genera para los verbos de reporte, ya que se genera primero un stem de los verbos.

Como se comentó anteriormente, la implementación del módulo está basada en el uso de expresiones regulares, es por esta razón que se explican algunos detalles relevantes al proyecto.

El siguiente código muestra el uso de una expresión regular dentro del contexto del módulo. Esta expresión busca comillas y si entre 2 comillas hay más de 4 palabras, considera que es una cita, y guarda un objeto de tipo *Coordenada.TipoCoordenada.QUOTE* con la información de la cita.

```
private String quotePattern(String text){
    java.util.regex.Pattern pattern = java.util.regex.Pattern.compile("\"[^\"]+\"");
    java.util.regex.Matcher matcher = pattern.matcher(text);

    while(matcher.find()){
        Coordenada c=new Coordenada(
                    Coordenada.TipoCoordenada.QUOTE,
                    matcher.start(),
                    matcher.end(),
                    matcher.group());
        text=text.substring(0,c.start)+c.getWhiteSpaces()+text.substring(c.end);
        StringTokenizer tok=new StringTokenizer(matcher.group());
        if(tok.countTokens()>4){
            coordenadas.add(c);
        }
    }

    return text;
}
```

Algunas Expresiones Regulares utilizadas

Durante el desarrollo de esta memoria, se han descrito diferentes expresiones regulares, que permiten encontrar algunas estructuras, a continuación se muestran algunas de ellas.

Quotes Expresión regular para identificar citas marcadas:

```
\"[^\"]+\"
```

Que Expresión regular para identificar el vocablo *que*:

```
(Q|q)u(e|é)
```

Entity 1 Expresión regular para identificar entidades (versión 1):

```
[A-ZÁÉÍÓÚÑ][a-záéíóúñ]{3,}
```

Entity 2 Expresión regular para identificar entidades (versión 2):

```
([A-ZÁÉÍÓÚÑ]\\.{0,1}){3,8}
```

Verbo Esta expresión se contruye usando la lista de verbos de citación [2] y concatenando el *stem* de cada verbo en una única expresión regular.

5.2.2. Formatos de Referencias

Existe un libro llamado *The Chicago Manual of Style* [25], el cual constituye una guía de estilos para escritores, editores y publicadores. Este libro define varios formatos para escribir referencias, los cuales fueron revisados y agrupados, obteniendo los siguientes patrones:

- [#]
- (#)
- *(#)*
- ***(#)***
- #
- (Apellido, año)
- (Apellido año)
- (Apellido)
- Nota al pie completa bibliográfica

Donde # representa un numero de referencia bibliográfica.

[2] Ver lista de verbos en Anexo J .

Algunos de estos patrones pueden ser fácilmente identificables utilizando expresiones regulares, sin embargo, otros patrones, se confunden fácilmente con la información del documento.
Por ejemplo:

- (Velásquez, 2012) corresponde a un patrón simple de identificar.
- 2012, es un patrón muy dificil de identificar como una referencia bibliográfica.

La información contenida en las referencias presenta marcas reconocibles que son buenos puntos para buscar citas bibliográficas. Es por esto que se incluye dentro de las estructuras que se analizan en el texto.

A continuación se muestran algunas de las expresiones regulares diseñadas para determinar los formatos de referencia descritos anteriormente, además de otros formatos detectados que son relevantes:

Formato de año Expresión regular para identificar formatos de año:

```
\\[|\\([0-9]{2,4}\\]|\\)
Ejemplo: [1989]
```

Formato Apellido y año 1 Expresión regular para identificar formatos que contienen un apellido seguido de un año, a esta expresión se agregó el patrón *et al.*:

```
[A-ZÁÉÍÓÚÑ][a-záéíóúñ]{3,}( et al\\.)?[,]?.[0-9]{4}[a-z]?(\\))?
Ejemplos:
    Contreras et al., 2012a
    (Martínez, 2009)
```

Formato Apellido y año 2 Al igual que la expresión regular anterior, también busca un apellido, pero el año va como una referencia adjunta y entre parentesis:

```
[A-ZÁÉÍÓÚÑ][a-záéíóúñ]{3,}( et al\\.)?.\\([0-9]{4}[a-z]?
Ejemplos:
    Neira (2010)
    Zuñiga et al. (1989)
```

Formato Sigla y año Expresión regular para identificar siglas escritas en mayúscula seguida de un año:

```
[A-Z]{3,}, [0-9]{4}(\\))?
Ejemplos:
    (NMG, 1994)
    ONEMI, 2010
```

5.2.3. Procedimiento detección

Utilizando las técnicas descritas anteriormente, es posible identificar las partes de la oración que tienen relevancia para el análisis de citas. De esta forma, se generan diferentes marcas dentro del texto (*Coordenadas*) las cuales al ser analizadas en su tipo y ubicación, permiten determinar si están los patrones que identifican a cada tipo de cita bibliográfica.

El algoritmo, descrito en prosa, que permite encontrar las citas es el siguiente:

1. Procesamiento del documento para extraer texto (se pasa a texto sin formato).
2. Marcar el texto con las cordenadas donde las expresiones regulares detectan coincidencias.
3. Buscar patrones dentro del listado de coordenadas, estos patrones corresponden a cada clasificación de cita.
4. Existe una tolerancia para definir cuando un patrón es detectado, la cual se debe ajustar.
5. Cuando se cumple el patrón con la tolerancia aceptada, se dice que hay una cita.

Patrones de citas directas

Para caracterizar las citas directas, se utilizó una definición general, la que fue refinada a lo largo de la investigación. La primera definición que se manejó fue: *se reproducen de manera textual las palabras del autor apoyándose en el uso de marcas tipográficas* [23]. Sin embargo, esta definición se debe ajustar con la gramática y sintaxis del idioma.

Esta categoría ya fue descrita en el Capítulo 2 y se puede identificar mediante los siguientes patrones:

- Entidad + verbo de reporte + cita
- Según + entidad + cita
- Para + entidad + cita
- De acuerdo con + entidad + cita

- En palabras de + entidad + cita
- En relación con lo planteado por + entidad + cita

Patrones de citas indirectas

Para caracterizar las citas indirectas, se utilizó una definición general, la que fue refinada a lo largo de la investigación. La primera definición que se manejó fue: *habitualmente se parafrasean las palabras del autor, obviando el uso de marcas tipográficas.*

Esta categoría ya fue descrita en el Capítulo 2 y se puede identificar mediante los siguientes patrones:

- Entidad + verbo de reporte + que + cita (sin marca tipográfica)
- Según + entidad + verbo de reporte + que + cita (sin marca tipográfica)
- Para + entidad + verbo de reporte + que + cita (sin marca tipográfica)
- De acuerdo con + entidad + verbo de reporte + que + cita (sin marca tipográfica)
- En palabras de + entidad + verbo de reporte + que + cita (sin marca tipográfica)
- En relación con lo planteado por + entidad + verbo de reporte + que + cita (sin marca tipográfica)

Patrones de citas integrales

Para caracterizar las citas integrales, se utilizó la siguiente definición a lo largo de la investigación: *se realiza una mención del autor de la información presentada dentro del mismo enunciado.* Esto significa que el autor es parte activa de la oración que contiene la cita.

Esta categoría ya fue descrita en el Capítulo 2 y se puede identificar con varios patrones, algunos de ellos, se muestran a continuación:

- Entidad + verbo de reporte + cita (independiente de la marca tipográfica)
- Según + entidad + verbo de reporte + cita (independiente de la marca tipográfica)
- Para + entidad + verbo de reporte + cita (independiente de la marca tipográfica)
- De acuerdo con + entidad + verbo de reporte + que + cita (independiente de la marca tipográfica)

- En palabras de + entidad + verbo de reporte + que + cita (independiente de la marca tipográfica)
- En relación con lo planteado por + entidad + verbo de reporte + que + cita (independiente de la marca tipográfica)

Patrones de citas no integrales

Para caracterizar las citas no integrales, se utilizó la siguiente definición a lo largo de la investigación: *el autor aparece mencionado fuera del enunciado, ya sea mediante una nota al pie de página o entre paréntesis al finalizar la cita.* Esto significa que el autor deja de ser parte activa de la oración que contiene la cita, pero de todas formas se le nombra.

Esta categoría ya fue descrita en el Capítulo 2 y se puede identificar con varios patrones. Algunos de ellos, se muestran a continuación:

- Verbo de reporte + Cita (independiente de la marca tipográfica) + referencia
- Cita (independiente de la marca tipográfica) + referencia
- Se dice que + cita (independiente de la marca tipográfica) + referencia
- De acuerdo con + cita (independiente de la marca tipográfica) + referencia tipográfica

Patrones de citas indeferenciados

Como parte de los análisis que se llevan a cabo, es necesario considerar algunos patrones que no han sido marcados dentro de las definiciones anteriores. Estos patrones han sido llamados indeferenciados, ya que al ejecutarse, podrían abarcar todo tipo de citas, incluso las que sean de difícil diferenciación para un experto en la materia.

Esta categoría no ha sido descrita, ya que es parte de la implementación de la librería más que las definiciones formales y será explorada dentro de los análisis ya que asume en si misma la dificultad de catalogar una cita. Algunos de los patrones que serán explorados:

- Referencia (utilizando todos los patrones de referencia implementados)
- Entidad + verbo + referencia
- Entidad + referencia

5.3. Framework de ejecución

Como parte de las implementaciones, se hizo necesario desarrollar un framework de ejecución mediante el cual se puedan automatizar algunos de los procesos de prueba y eventualmente este framework podrá usarse como parte de la plataforma administrativa de DOCODE. Es por esta razón que la implementación obedece al diseño de un sitio Web, donde se pueden ejecutar diferentes funciones.

5.3.1. Interfaz de usuario

Se considera que esta interfaz puede ser útil para posteriormente ampliar la funcionalidad y desarrollar una aplicación administrativa. Es por esto que actualmente ya incluye algunas de esas interfaces.

Interfaz ingreso a la aplicación

Lo primero es que la aplicación considera el uso de usuarios registrados en el sistema, razón por la cual se agregó una interfaz para autenticación, donde el usuario debe indicar sus credenciales. La Figura 5.4 muestra esta interfaz, utilizando el logotipo actual de DOCODE.

Figura 5.4: Interfaz de autenticación del framework.
Fuente: Elaboración propia

Detalle menu de usuario

Como corresponde a la aplicación, ésta cuenta con un menú de usuario que permite acceder a las distintas funciones que están implementadas. Actualmente estan disponibles tres menus principales:

- Ejecutar DOCODE, que permite inyectar una tarea en el flujo de proceso de análisis de originalidad.
- JMS, que permite ingresar un ticket para ser reprocesado completamente o por algún algoritmo específico.
- Ejecutar Algoritmos, aquí se concentran las pruebas que se realizarán sobre los algoritmos.

La Figura 5.5 muestra el banner de la aplicación, además de un cuadro donde se muestra el nombre del usuario conectado y el menú principal, desde donde se accede a las funciones ya mencionadas.

Figura 5.5: Detalle del banner y menu general.
Fuente: Elaboración propia

Opción Ejecutar Docode

Desde aquí se puede ejecutar la funcionalidad de análisis de originalidad, sin necesidad de usar las interfaces DODOCE ASP ni DOCODE Lite. Esta interfaz es especialmente útil para apoyar el proceso de integración de nuevos algoritmos, además de permitir otra forma de probar la conectividad del servicio.

La Figura 5.6 muestra la interfaz de ejecución para DOCODE Lite, la cual es análoga a la que se usa para ejecutar directamente el servicio sobre DOCODE Core. En ella, se puede ver también los datos necesarios para procesar esta información como si se tratara de una tarea. En particular, el campo mostrado como *Archivos* permite subir los archivos a ser analizados.

Opción JMS

Utilizando esta opción, se puede dejar un mensaje equivalente a un número de ticket, el cual identifica univocamente un proceso de análisis para que ese proceso sea retomado por los algoritmos.

Esta opción ofrece 2 variantes, *Topic* y *Queue*. *Topic* se utiliza para enviar el mismo mensaje de forma simultanea a todos los algoritmos, mientras que *Queue* permite enviar una señal de procesamiento hacia un sólo algoritmo, permitiendo así una ejecución aislada.

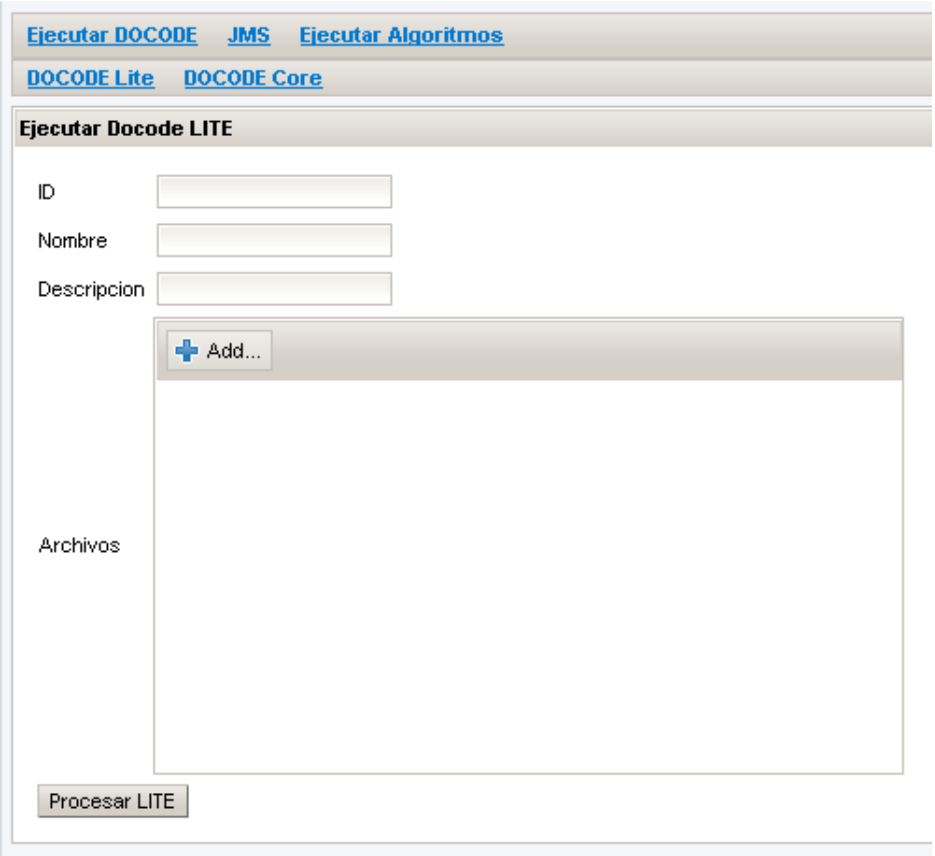

Figura 5.6: Interfaz de ejecución del proceso de análisis de originalidad.
Fuente: Elaboración propia

La Figura 5.7 muestra la interfaz que permite ingresar un número de ticket para ser procesado por todos los algoritmos mediante una integración utilizando un *Topic*. Esta interfaz es similar a la de insertar en una *Queue*, con la diferencia que en ella se indica también la *Queue* de destino del mensaje.

Figura 5.7: Interfaz que permite utilizar mensajería JMS.
Fuente: Elaboración propia

Opción de ejecutar algoritmos

Dentro de este menú, por ahora está solamente la opción de ejecutar el algoritmo de citas, pero contiene 2 variantes para el proceso de ejecución: Procesamiento de Citas y Procesamiento Masivo de citas.

SubMenú Citas Desde aqui se puede ejecutar el proceso de análisis de citas sobre un archivo especificado dentro de los parámetros por la llave *archivo*. Además se puede identificar el archivo que contiene las citas a ser identificadas en el parámetro *validación*. Los otros parámetros corresponden a valores especificos que regulan la acción del algoritmo. Dentro del campo *Entidades*, se pueden seleccionar los tipos de coordenadas que se utilizarán como patrón para determinar una cita. La Figura 5.8 muestra la interfaz donde se ingresan los datos para ejecutar el algoritmo, según la descripción indicada.

SubMenú Cita Masiva La interfaz es análoga a la anterior, sin embargo, en ella no se especifican archivos, sino carpetas, donde estan todos los archivos que serán procesados. De esta forma, la llave *Archivo* cambia por *carpetaTexts* para indicar donde estan los archivos a procesar y la llave *validación* cambia por *carpetaSolve* para indicar donde estan los archivos con las citas marcadas. Estos archivos, se llaman igual que los analizados, pero se agrega el prefijo *SOLVE*.

En ambas interfaces, se calculan los valores de *Precision*, *Recall* y *F-Measure*, los cuales permiten evaluar los resultados obtenidos por los distintos patrones sobre los distintos algoritmos y que se

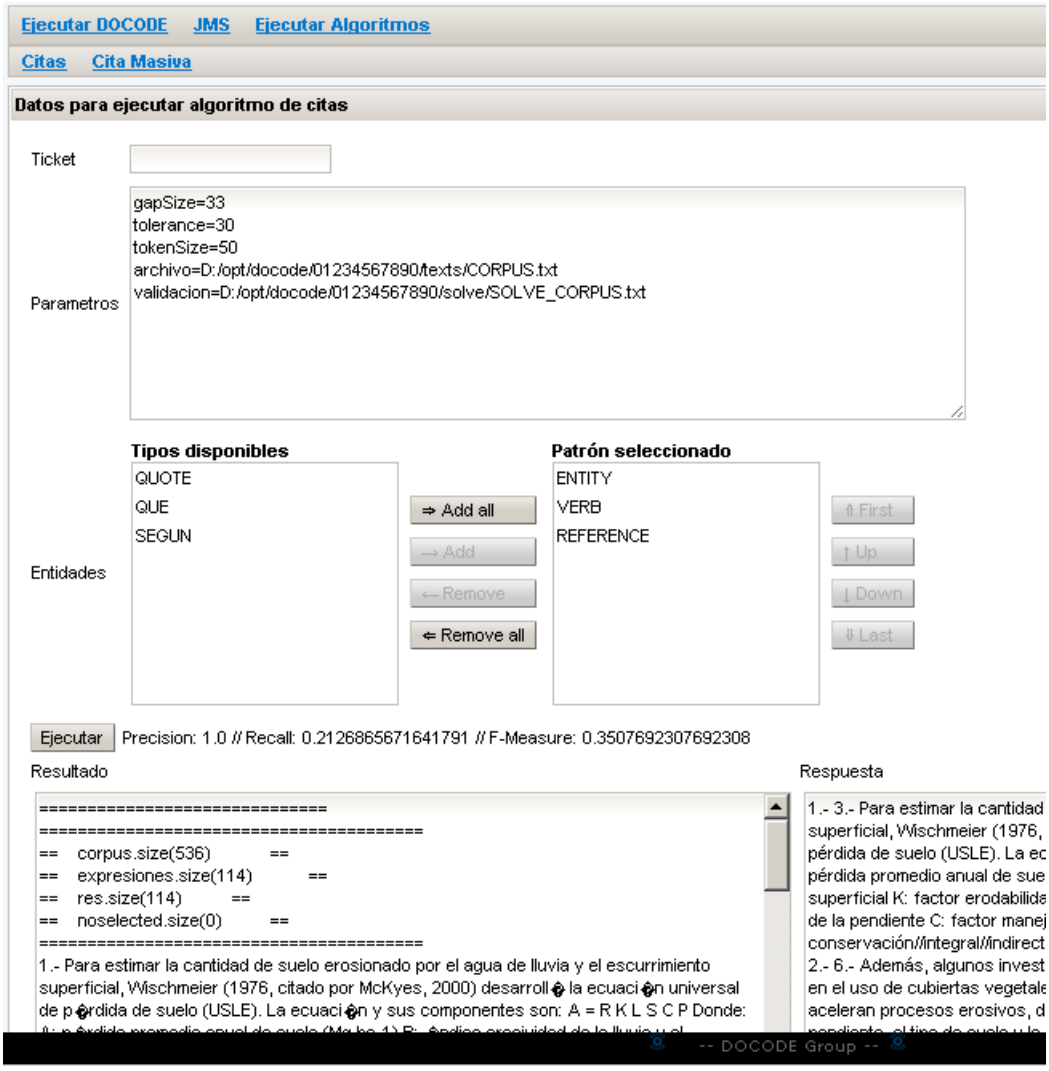

Figura 5.8: Interfaz para ejecutar algoritmo de Citas.
Fuente: Elaboración propia

utilizarán en la etapa de experimentación.

5.3.2. Modelamiento de capas

Esta aplicación, implementa el modelo de capas que está definido para la nueva arquitectura de DOCODE. Se integra dentro de cada capa de la siguiente manera:

- Capa Web: utilizando JSF junto con RichFaces para las interfaces de usuario.
- Capa Servicios: hasta el momento no disponibiliza servicios directamente, sólo los utiliza como cliente.
- Capa Lógica Negocio: las lógicas propias de los clientes donde utiliza JAX-WS para integrarse con los diferentes algoritmos mediante llamadas a los *webservices*
- Capa Persistencia: se conecta a la capa de persistencia utilizando JPA. Con esto administra la persistencia de los objetos de negocio.
- Capa Datos: todo el modelo de datos queda alojado en una base de datos configurada para este fin.

5.4. Corpus de citas bibiográficas

Se trabajó en conjunto con el equipo de desarrollo de DOCODE, para hacer un corpus de citas bibliográficas. Este trabajo esta descrito por Araya et al. en *Mecanismos de citación en tesis de pregrado de la Universidad de Chile* [1]. A continuación se presentan algunos de los resultados de ese informe.

5.4.1. Resumen Memorias de Agronomía

A continuación se presentan 2 tablas con el resumen de los tipos de citas detectados en la memorias de Agronomía.
El Cuadro 5.1 muestra el resumen de los resultados obtenidos del análisis de las memorias de agronomía respecto de las citas directas e indirectas.
El Cuadro 5.2 muestra el resumen de los resultados obtenidos del análisis de las memorias de agronomía respecto de las citas integrales y no integrales.

Archivo	Citas directas	%	Citas indirectas	%	Total
AGR01	0	0	11	100	11
AGR02	28	34	55	66	83
AGR03	0	0	32	100	32
AG304	0	0	17	100	17
AGR05	3	38	5	62	8
AGR06	0	0	9	100	9
AGR07	0	0	28	100	28
AGR08	1	11	8	89	9
AGR09	0	0	37	100	37
AGR10	0	0	35	100	35
AGR11	0	0	34	100	34
AGR12	0	0	32	100	32
AGR13	0	0	11	100	11
AGR14	0	0	16	100	16
AGR15	0	0	30	100	30
AGR16	0	0	15	100	15
AGR17	0	0	47	100	47
AGR18	0	0	50	100	50
AGR19	0	0	18	100	18
AGR20	4	36	7	64	11
Total	36	7	497	93	533

Cuadro 5.1: Resumen de citas directas e indirectas en memorias de agronomía.

Archivo	Integrales	%	No Integrales	%	Total
AGR01	7	64	4	36	11
AGR02	50	61	32	39	82
AGR03	26	81	6	19	32
AG304	12	71	5	29	17
AGR05	6	75	2	25	8
AGR06	0	0	9	100	9
AGR07	25	89	3	11	28
AGR08	5	56	4	44	9
AGR09	29	78	8	22	37
AGR10	27	77	8	23	35
AGR11	30	88	4	12	34
AGR12	29	91	3	9,4	32
AGR13	9	82	2	18	11
AGR14	11	69	5	31	16
AGR15	27	90	3	10	30
AGR16	6	40	9	60	15
AGR17	39	83	8	17	47
AGR18	37	74	13	26	50
AGR19	13	72	5	28	18
AGR20	8	73	3	27	11
Total	396	74	136	26	532

Cuadro 5.2: Resumen de citas integrales y no integrales en memorias de agronomía.

5.4.2. Resumen Memorias de Ingeniería Industrial

A continuación se presentan 2 tablas con el resumen de los tipos de citas detectados en la memorias de Ingeniería Industrial.

El Cuadro 5.3 muestra el resumen de los resultados obtenidos del análisis de las memorias de industrias respecto de las citas directas e indirectas.

El Cuadro 5.4 muestra el resumen de los resultados obtenidos del análisis de las memorias de industrias respecto de las citas integrales y no integrales.

Archivo	Citas directas	%	Citas indirectas	%	Total
IND02	3	23	10	77	13
IND03	2	50	2	50	4
IND04	0	0	4	100	4
IND05	3	43	4	57	7
IND06	0	0	28	100	28
INDO7	3	21	11	79	14
IND08	1	20	4	80	5
IND10	1	25	3	75	4
IND11	11	20	44	80	55
IND12	1	14	6	86	7
IND13	1	4	22	96	23
IND14	5	21	19	79	24
IND15	0	0	10	100	10
IND16	10	83	2	17	12
IND17	3	13	20	87	23
IND18	3	21	11	79	14
IND20	0	0	6	100	6
Total	47	19	206	81	253

Cuadro 5.3: Resumen de citas directas e indirectas en memorias de industrias.

5.4.3. Aumento del volumen del corpus

Una vez obtenido el corpus de citas correctamente clasificadas y categorizadas, se debe generar un corpus de mayor tamaño utilizando textos que no corresponden a citas y sobre este nuevo corpus, realizar las pruebas de búsqueda.

Para este fin, se descargaron 484 (cuatrocientos ochenta y cuatro) libros de dominio público, que están disponibles en el sitio de dominio público del gobierno de Brasil [3]. Una vez descargados los libros, se seleccionaron en total 5.890 (cinco mil ochocientos noventa) frases adicionales, las que debían cumplir con dos condiciones básicas: Empezar con una mayúscula y tener más de 110

[3] Para más información o descargar los libros revise [8]

Archivo	Integrales	%	No Integrales	%	Total
IND02	11	85	2	15	13
IND03	2	50	2	50	4
IND04	3	75	1	25	4
IND05	5	71	2	29	7
IND06	23	82	5	18	28
INDO7	13	93	1	7	14
IND08	4	80	1	20	5
IND10	4	100	0	0	4
IND11	38	69	17	31	55
IND12	6	86	1	14	7
IND13	19	83	4	17	23
IND14	23	96	1	4	1
IND15	4	40	6	60	10
IND16	6	50	6	50	12
IND17	17	74	6	26	23
IND18	11	79	3	21	14
IND20	4	67	2	33	6
Total	193	84	60	26	230

Cuadro 5.4: Resumen de citas integrales y no integrales en memorias de industrias.

caracteres. Si bien estas medidas son completamente arbitrarias, permitieron elegir una muestra que asimila un muestreo aleatorio.

Las pruebas se realizaron sobre el corpus de las memorias de agronomía, que al mezclar junto con las oraciones extraídas de los libros, se obtiene un total de 6.401 (seis mil cuatrocientos uno) oraciones sobre las cuales trabajar.

El Cuadro 5.5 muestra un resumen de las cantidades de oraciones que se utilizan para ejecutar el experimento.

Cantidad de citas reales en el corpus	536
Cantidad de oraciones agregadas	5.890
Cantidad total de oraciones del corpus	6.401

Cuadro 5.5: Cantidad de oraciones en el corpus.

5.5. Experimento de laboratorio

A continuación, se explicará la metodología utilizada para ejecutar los algoritmos, además de mostrar el proceso de definición y creación del corpus sobre el cual se realizaron las pruebas.

5.5.1. Parámetros de la medición

Precisión

Se define como la fracción de documentos recuperados que son relevantes para la necesidad de información del usuario. La Ecuación 5.1 muestra la fórmula.

$$\text{precision} = \frac{|\{\text{oraciones relevantes}\} \cap \{\text{oraciones recuperadas}\}|}{|\{\text{oraciones recuperadas}\}|} \tag{5.1}$$

El peor valor de este indicador es cero y corresponde al caso cuando se seleccionan sólo oraciones que no pertenecen a las citas reales. Por otro lado, el mejor valor es uno y corresponde al caso cuando se lograron recuperar sólo citas reales.

Recall

Se define como la fracción de documentos relevantes para una consulta que fueron recuperados. La Ecuación 5.2 muestra la fórmula.

$$\text{recall} = \frac{|\{\text{oraciones relevantes}\} \cap \{\text{oraciones recuperadas}\}|}{|\{\text{oraciones relevantes}\}|} \tag{5.2}$$

El peor valor para este indicador es cero y corresponde al caso en que no se lograron obtener citas reales. Mientras que el mejor valor es uno y corresponde al caso en que se lograron recuperar todas las citas reales.

F-Measure

Es un indicador que permite concentrar y balancear los valores de *Presicion* y *Recall*. Aquí se utiliza la evaluación que pondera de igual forma ambos índices. La Ecuación 5.3 muestra la fórmula.

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{5.3}$$

Este indicador es cero cuando sólo uno de los índices lo es y vale uno cuando ambos índices son iguales a uno. Hay que agregar que este indicador se indetermina cuando ambos indicces son iguales a cero. Este caso es posible y corresponde a ejecuciones con muy malos resultados.

Clasificación de resultados

En general, para las labores de clasificación se utilizan los términos *verdadero positivo*, *falso positivo*, *verdadero negativo* y *falso negativo*. Los términos *verdadero* y *negativo* se refieren a la predicción (comportamiento esperado) y los términos *verdadero* y *falso* corresponden al juicio externo (comportamiento observado).

En particular para el análisis de citas, los significados utilizados son los siguientes:

Verdadero positivo Este caso ocurre cuando el algoritmo determinó que una oración es una cita bibliográfica y efectivamente lo es. Este es un resultado correcto.

Falso positivo Cuando el algoritmo indica que una oración es una cita bibliográfica, sin embargo la oración no está dentro del corpus de citas. Este resultado es inesperado ya que son marcas que no deberían estar.

Falso negativo Esto sucede cuando una cita biliográfica que pertenece al corpus no ha sido detectada por el algoritmo. Dicho de otro modo es un resultado extraviado.

Verdadero negativo Las oraciones que el algoritmo no marcó como cita bibliográfica y que efectivamente no lo son. Este resultado corresponde al de una ausencia esperada.

El Cuadro 5.6 es un resumen de los estados posibles para clasificar las oraciones entregadas como resultado del algoritmo.

Comportamiento	**Observado**	
Esperado	**vp** (verdadero positivo) Resultado correcto	**fp** (falso positivo) Resultado inesperado
	fn (falso negativo) Resultado extraviado	**vn** (verdadero negativo) Ausencia esperada

Cuadro 5.6: Resumen de clasificaciones para los resultados generados por el algoritmo.

Basándose en las clasificaciones anteriores es posible redefinir las ecuaciones que se usan para calcular *precisión* y *recall*. Las Ecuaciones 5.4 y 5.5 muestran la forma alternativa para las Ecuaciones 5.1 y 5.2 respectivamente. Hay que destacar que el valor de *vp+fp* corresponde al total de oraciones que el algoritmo indicó como válidas, mientras que el valor de *vp+fn* corresponde al total de citas bibliográficas contenidas en el corpus.

$$\text{precision} = \frac{vp}{vp + fp} \tag{5.4}$$

$$\text{recall} = \frac{vp}{vp + fn} \tag{5.5}$$

5.5.2. Algoritmos de distancias de texto

Para determinar cuando una oración seleccionada por el algoritmo está dentro del corpus, se procede a evaluar un índice de similitud basado en algoritmos que permiten medir la similitud de cadenas de caracteres. Para este fin se implementaron 3 alternativas.

Distancia de Levenshtein También conocida como distancia de edición, o distancia entre palabras, corresponde al número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Se entiende por operación a una inserción, eliminación o sustitución de un carácter.

Needleman Wunsch Sirve para realizar alineamientos globales de dos secuencias. El algoritmo funciona del mismo modo independientemente de la complejidad o longitud de las secuencias y garantiza la obtención del mejor alineamiento.

Smith Waterman Corresponde a una reconocida estrategia para realizar alineamiento local de secuencias biológicas (ADN, ARN o proteínas). Es decir, que determina regiones similares entre un par de secuencias.

Al utilizar los tres algoritmos, se pudo determinar que la mejor condición para saber cuando dos oraciones se parecen depende de los valores obtenidos de los índices de *SmithWaterman* y *Levenshtein*. A continuación se muestra la evaluación y el valor retornado.

```
if(SmithWaterman.index > Levenshtein.index)
  then
      return Levenshtein.index
  else
      return SmithWaterman.index
```

Cuando ese valor es superior al 90 %, se determina que estamos en presencia de dos oraciones lo suficientemente iguales. En algunos casos es posible ajustar este valor hasta el 95 % con una pérdida de menos del 20 % en los resultados.

5.5.3. Ejecución

Para evaluar los resultados entregados por el algoritmo, se definen los índices según lo especificado en las Ecuaciones 5.6 y 5.7.

$$\text{precision} = \frac{|\{\text{oraciones recuperadas que son citas reales}\}|}{|\{\text{oraciones recuperadas}\}|} \tag{5.6}$$

$$\text{recall} = \frac{|\{\text{oraciones recuperadas que son citas reales}\}|}{|\{\text{citas reales}\}|} \tag{5.7}$$

Para esto, se implementó un algoritmo que al ejecutarse, revisa las citas de un texto y lo compara con el resultado que ya ha sido procesado en una etapa anterior como parte de un corpus. De esa forma se puede medir los parámetros de precisión y recall para las citas.

Adicionalmente, se consideran dos opciones para analizar los patrones de citas. Uno es respetando el orden del patrón ingresado y la otra opción es sólo de pertenencia, o sea que el patrón exista completamente dentro de la oración analizada.

5.5.4. Resultados

El experimento consiste en evaluar distintos patrones de los que fueron clasificados como citas directas, indirectas, integrales y no integrales, para ver los índices asociados a cada uno de ellos.

Patrón ordenado

Las primeras pruebas se realizaron considerando el patrón ordenado, además entre las coordenadas no debe haber una distancia mayor a cinco coordenadas. Se realizaron pruebas con otros patrones, pero los resultados fueron más bajos, por lo que quedaron fuera de los resultados mostrables. El Cuadro 5.7 muestra los resultados obtenidos al evaluar esos patrones sobre un total de 536 citas posibles (la tabla está ordenada según el valor de F-Measure).

Patrón	vp	vp+fp	vp+fn	Precisión	Recall	F-Measure
Entidad+Verbo+Cita	4	4	536	1,0000	0,0075	0,0148
Entidad+Cita	10	10	536	1,0000	0,0187	0,0366
Entidad+Verbo	170	1195	536	0,1423	0,3172	0,1964
Entidad+Verbo+QUE	126	668	536	0,1886	0,2351	0,2093
Entidad+Verbo+Referencia	114	114	536	1,0000	0,2127	0,3508
Promedios	84,8	398,2	536	0,6662	0,1582	0,1616

Cuadro 5.7: Resultados obtenidos considerando el patrón ordenado.

Patrón desordenado

Estas pruebas se realizaron considerando el patrón desordenado. Esto significa que el patrón puede estar respetando el orden indicado o no, pero siempre se considera la completitud del patrón dentro de la oración analizada para indicar una oración valida. El Cuadro 5.8 muestra los resultados obtenidos al evaluar esos patrones en las condiciones descritas sobre un total de 536 citas posibles (la tabla está ordenada según el valor de F-Measure).

Patrón	vp	vp+fp	vp+fn	Precisión	Recall	F-Measure
Entidad+Verbo+QUE	407	2372	536	0,1716	0,7593	0,2799
Entidad+Verbo	423	1892	536	0,2236	0,7892	0,3484
Según+Entidad+Verbo+QUE	334	1169	536	0,2857	0,6231	0,3918
Entidad+Verbo+QUE+Cita	335	1170	536	0,2863	0,6250	0,3927
Según+Entidad+Verbo	323	686	536	0,4708	0,6026	0,5286
Entidad+Verbo+Cita	327	690	536	0,4739	0,6101	0,5334
Entidad+Verbo+referencia	424	787	536	0,5388	0,7910	0,6410
Promedios	367,6	1252,3	536	0,3501	0,6858	0,4451

Cuadro 5.8: Resultados obtenidos considerando el patrón desordenado.

Pipe and filter

Adicionalmente a la búsqueda de patrones, se agregó una estrategia relacionada a un patrón de integración llamado *pipe and filter*, el cual consiste en definir una entrada, que se filtra para obtener una salida. Este mismo proceso se hace usando varios filtros para construir el analizador. Esta forma de revisión es análoga a la de patrón desordenado, sin embargo los resultados se ven mejor para esta estrategia. El Cuadro 5.9 muestra los resultados de la evaluación en estas condiciones sobre un total de 536 citas posibles (la tabla está ordenada según el valor de F-Measure).

Patrón	vp	vp+fp	vp+fn	Precisión	Recall	F-Measure
Entidad+Verbo+QUE+Cita	9	9	536	1,0000	0,0168	0,0330
Verbo+Cita	12	12	536	1,0000	0,0224	0,0438
Entidad+Verbo	502	1731	536	0,2900	0,9366	0,4429
Entidad+Verbo+Referencia	480	482	536	0,9959	0,8955	0,9430
Verbo+Referencia	487	490	536	0,9939	0,9086	0,9493
Promedios	298,0	544,8	536	0,8559	0,5560	0,4824

Cuadro 5.9: Resultados obtenidos mediante *pipe and filter*.

Patrón no identificado

Además de probar los patrones definidos, se investigó la aplicación de otros patrones no clasificados como citas directas, indirectas, integrales o no integrales. Estas pruebas al ser patrones atómicos, son independientes del orden o desorden de ellos dentro de la oración. El Cuadro 5.10 muestra los resultados obtenidos al evaluar esos patrones en las condiciones descritas sobre un máximo de 536 citas biliográficas posibles (la tabla esta ordenada según el valor de F-Measure).

5.5.5. Análisis de resultados

El análisis de los resultados se hará en el mismo orden en que se presentaron los resultados del experimento. Esto es, análizando con el patrón ordenado, desordenado y sin patrón definido.

Patrón	vp	vp+fp	vp+fn	Precisión	Recall	F-Measure
Cita	16	16	536	1,0000	0,0299	0,0580
Verbo	445	3061	536	0,1454	0,8302	0,2474
Referencia	508	511	536	0,9941	0,9478	0,9704
Promedios	323	1196	536	0,7132	0,6026	0,4253

Cuadro 5.10: Resultados obtenidos considerando patrones no definidos.

Patrón ordenado

Una de las estructuras más importantes para definir las citas, debía ser la que contiene una entidad seguida de un verbo de reporte. Este patrón se repite constantemente en todas las definiciones de los tipos de citas. Sin embargo, el mejor valor obtenido fue al mezclarlo con la referencia. De modo que al buscar el patrón *Entidad Verbo Referencia*, se obtienen los valores: 1,0000 (Precision), 0,2127 (Recall) y 0,3508 (F-Measure).

Un punto a destacar es que cuando se considera dentro del patrón una cita (con marca tipográfica), el valor del indice Precision se mantiene en uno.

Patrón desordenado

Una vez más, el análisis se centra en la estructura de Entidad y Verbo, sólo que al considerar el patrón sin importar el orden, se obtuvo un valor mejor para esa combinación subiendo el F-Measure desde 0,1964 a 0,3438 lo que todavía sigue siendo bastante deficiente a pesar de haber detectado más del doble de citas correctas que en el primer caso.

Una vez más vemos que la mejor combinación para Entidad y Verbo, es combinarlo con Referencia, pasando de un valor de 0,3508 a 0,6410 para F-Measure, pero una pérdida importante en el valor de Precision que cayó desde 1 a 0,5388 mostrando una baja muy grande.

Si bien en terminos generales los valores son mejores, hay una importante baja en los valores de precision, que también se ve reflejado en el descenso de los valores promedio. Esto impacta en que la calidad del algoritmo no es buena ya que genera muchos falsos positivos.

Pipe and filter

El uso de este patrón, permite mejorar los resultados anteriores, por ejemplo para el patrón *Entidad + Verbo*, se mejora el F-Measure desde 0,3483 a 0,4429; otro ejemplo es para *Entidad + verbo + referencia*, se obtuvo una mejora para F-Measure desde 0,6410 a 0,9430.

Patrón no identificado

Al buscar patrones que son independientes del orden (se podría hablar de un *patrón singleton*), aparece el mejor valor de los conseguidos hasta el momento, al utilizar sólo la marca de Referencia, se obtienen valores de 0,9941 (precision), 0,9478 (recall) y 0,9704 (F-Measure).

5.5.6. Observaciones

Dado el nivel alcanzado por el patrón de referencia, hay que hacer algunos alcances al respecto. Para definir un poco más de que se habla al referirse a este parámetro.

Para considerar una referencia se utilizaron en total trece expresiones regulares distintas, algunas de ellas están fuera de la definición usada como guía y parece ser que esto ha sido un factor determinante ya que permitió ampliar el rango de búsqueda.

Finalmente las estructuras que más se encuentran en el texto son:

- (Apellido et al., año)
- (Apellido et al. año)
- (Apellido et al.)
- Apellido et al. (año)
- (año) — et al.
- SIGLA, año)
- (nombre1 nom2 nom3, año)

Al hacer un cruce entre la información recopilada por la investigación que generó el corpus y los resultados obtenidos, una gran cantidad de citas queda determinada por el uso de paréntesis (95 % de los documentos las utiliza), ya sea para la cita misma o para la referencia. Es muy posible que la diferencia que queda entre el resultado obtenido y el óptimo esté dada por las citas que no utilizan paréntesis (citas directas, integrales) y que están identificadas dentro del corpus.

5.6. Experimentos adicionales

Se define un experimento adicional sobre textos reales, principalmente para evaluar el comportamiento del módulo de citas fuera del laboratorio. Este test pretende poner a prueba el módulo de

citas para cuando esté integrado dentro del sistema DOCODE y consiste en hacer la búsqueda de citas sobre los textos de las memorias que se utilizaron para construir el corpus.

5.6.1. Respecto del corpus

Para la construcción del corpus se utilizaron 20 memorias de Agronomía, las cuales en su conjunto poseen un total de 15.169 (quince mil ciento sesenta y nueve) oraciones, de las cuales sólo 536 (quinientas treinta y seis) corresponden a citas, lo que equivale al 3,53 % de las oraciones.

5.6.2. Experimento

Para realizar este experimento sólo se utilizará la estrategia de *Pipe and Filter* sobre los patrones que dieron mejores resultados. No se realizará la exploración utilizando otras fórmulas, ya que se sospecha que los resultados serán deficientes.

5.6.3. Ejecución

La evaluación de los resultados se hará igual al procedimiento anterior, evaluando y análizando los índices de *Precision*, *Recall* y *F-Measure*.

5.6.4. Resultados

Cada uno de los siguientes cuadros muestran los resultados de ejecutar los algoritmos sobre documentos reales. El Cuadro 5.11 muestra los valores obtenidos de aplicar un filtro sobre Entidad y verbo. El Cuadro 5.12 despliega los resultados de aplicar un filtro sobre Entidad, verbo y el vocablo Que. El Cuadro 5.13 presenta los indicadores resultantes de aplicar el filtrado sobre Referencia. El Cuadro 5.14 muestra los valores resultantes de ejecutar el filtro sobre Verbo y Referencia.

A partir de los datos obtenidos, se contruye la siguiente tabla de resumen, donde se calcularon los índices sobre los acumulados de todos los documentos. El Cuadro 5.15 muestra el resultado obtenido de cada patrón estudiado.

5.6.5. Análisis de resultados

De todos los resultados obtenidos, sólo se pueden destacar algunos valores aislados, ya que en todos los casos, el indicador *Precision* es deficiente, al punto que ninguna de las mediciones consigue superar el 0,5 como base para ser considerado como un resultado aceptable. Donde si se obtienen valores importantes es en el índice *Recall*, con tres evaluaciones sobre el 0,8 lo que es considerado bueno. En definitiva, el indicador F-Measure muestra sólo un valor que podría considerarse casi

Patrón	vp	vp+fp	vp+fn	Precisión	Recall	F-Measure
AGR20	14	360	11	0,0389	1,2727	0,0755
AGR05	10	242	8	0,0413	1,2500	0,0800
AGR06	9	192	9	0,0469	1,0000	0,0896
AGR04	20	320	17	0,0625	1,1765	0,1187
AGR08	5	70	9	0,0714	0,5556	0,1266
AGR14	11	140	16	0,0786	0,6875	0,1410
AGR15	14	149	30	0,0940	0,4667	0,1564
AGR01	9	103	11	0,0874	0,8182	0,1579
AGR13	9	102	11	0,0882	0,8182	0,1593
AGR02	77	724	86	0,1064	0,8953	0,1901
AGR03	23	202	32	0,1139	0,7188	0,1966
AGR19	15	132	18	0,1136	0,8333	0,2000
AGR17	50	406	47	0,1232	1,0638	0,2208
AGR09	28	215	37	0,1302	0,7568	0,2222
AGR07	27	213	28	0,1268	0,9643	0,2241
AGR11	34	266	34	0,1278	1,0000	0,2267
AGR10	34	247	35	0,1377	0,9714	0,2411
AGR16	15	83	15	0,1807	1,0000	0,3061
AGR12	32	176	32	0,1818	1,0000	0,3077
AGR18	52	268	50	0,1940	1,0400	0,3270
Promedios	24,4	230,5	26,8	0,1073	0,9145	0,1884

Cuadro 5.11: Valores por documento usando el patrón *Entidad + Verbo*.

Patrón	vp	vp+fp	vp+fn	Precisión	Recall	F-Measure
AGR02	0	0	86	0,0000	0,0000	0,0000
AGR03	0	0	32	0,0000	0,0000	0,0000
AGR07	3	37	28,01	0,0811	0,1071	0,0923
AGR20	9	162	11	0,0556	0,8182	0,1040
AGR05	5	87	8	0,0575	0,6250	0,1053
AGR15	9	69	30	0,1304	0,3000	0,1818
AGR08	3	22	9	0,1364	0,3333	0,1935
AGR06	6	51	9	0,1176	0,6667	0,2000
AGR04	15	127	17	0,1181	0,8824	0,2083
AGR16	5	32	15	0,1563	0,3333	0,2128
AGR17	23	164	47	0,1402	0,4894	0,2180
AGR01	5	33	11	0,1515	0,4545	0,2273
AGR14	9	62	16	0,1452	0,5625	0,2308
AGR13	6	33	11	0,1818	0,5455	0,2727
AGR11	23	120	34	0,1917	0,6765	0,2987
AGR09	19	82	37	0,2317	0,5135	0,3193
AGR10	29	111	35	0,2613	0,8286	0,3973
AGR19	13	46	18	0,2826	0,7222	0,4063
AGR18	38	123	50	0,3089	0,7600	0,4393
AGR12	27	86	32	0,3140	0,8438	0,4576
Promedios	12,35	72,4	26,8	0,1531	0,5231	0,2283

Cuadro 5.12: Valores por documento usando el patrón *Entidad + Verbo + Que*.

Patrón	vp	vp+fp	vp+fn	Precisión	Recall	F-Measure
AGR20	9	104	11	0,0865	0,8182	0,1565
AGR06	9	89	9	0,1011	1,0000	0,1837
AGR05	7	67	8	0,1045	0,8750	0,1867
AGR14	12	82	16	0,1463	0,7500	0,2449
AGR03	21	128	32	0,1641	0,6563	0,2625
AGR08	8	50	9	0,1600	0,8889	0,2712
AGR13	10	53	11	0,1887	0,9091	0,3125
AGR01	9	38	11	0,2368	0,8182	0,3673
AGR16	15	52	15	0,2885	1,0000	0,4478
AGR04	17	58	17	0,2931	1,0000	0,4533
AGR10	29	91	35	0,3187	0,8286	0,4603
AGR07	25	75	28	0,3333	0,8929	0,4854
AGR09	35	106	37	0,3302	0,9459	0,4895
AGR19	18	55	18	0,3273	1,0000	0,4932
AGR11	32	89	34	0,3596	0,9412	0,5203
AGR02	63	156	86	0,4038	0,7326	0,5207
AGR18	50	117	50	0,4274	1,0000	0,5988
AGR15	27	55	30	0,4909	0,9000	0,6353
AGR17	43	85	47	0,5059	0,9149	0,6515
AGR12	30	59	32	0,5085	0,9375	0,6593
Promedios	23,45	80,5	26,8	0,2888	0,8905	0,4200

Cuadro 5.13: Valores por documento usando el patrón *Referencia*.

Patrón	vp	vp+fp	vp+fn	Precisión	Recall	F-Measure
AGR20	9	57	11	0,1579	0,8182	0,2647
AGR14	11	62	16	0,1774	0,6875	0,2821
AGR06	9	53	9	0,1698	1,0000	0,2903
AGR05	7	29	8	0,2414	0,8750	0,3784
AGR08	8	29	9	0,2759	0,8889	0,4211
AGR03	21	67	32	0,3134	0,6563	0,4242
AGR13	9	29	11	0,3103	0,8182	0,4500
AGR01	8	24	11	0,3333	0,7273	0,4571
AGR19	15	47	18	0,3191	0,8333	0,4615
AGR04	17	52	17	0,3269	1,0000	0,4928
AGR09	32	73	37	0,4384	0,8649	0,5818
AGR07	25	56	28	0,4464	0,8929	0,5952
AGR11	31	69	34	0,4493	0,9118	0,6019
AGR10	27	54	35	0,5000	0,7714	0,6067
AGR15	21	39	30	0,5385	0,7000	0,6087
AGR16	14	30	15	0,4667	0,9333	0,6222
AGR02	60	101	86	0,5941	0,6977	0,6417
AGR18	50	101	50	0,4950	1,0000	0,6623
AGR17	41	72	47	0,5694	0,8723	0,6891
AGR12	30	41	32	0,7317	0,9375	0,8219
Promedios	22,25	54,3	26,8	0,3927	0,8443	0,5177

Cuadro 5.14: Valores por documento usando el patrón *Verbo + Referencia*.

Patrón	vp	vp+fp	vp+fn	Precisión	Recall	F-Measure
Entidad+Verbo	488	4610	536	0,1059	0,9105	0,1897
Entidad+Verbo+que	247	1447	536	0,1707	0,4608	0,2491
Referencia	469	1609	536	0,2915	0,8750	0,4373
Verbo+Referencia	445	1085	536	0,4101	0,8302	0,5490
Promedios	412,3	2188	536	0,2445	0,7691	0,3563

Cuadro 5.15: Resumen de los totales de cada patrón sobre documentos reales.

aceptable para el caso del patrón *Verbo+Referencia*, donde se obtuvo una puntuación de 0,549; resultado que todavía se debería mejorar.

5.6.6. Observaciones

Después de analizar en detalle los resultados, podemos enumerar algunas de las causas probables de la caída de los resultados.

- Problemas al cortar oraciones, el proceso para cortar oraciones no existe en las pruebas de laboratorio.
- Problemas con el corpus, ya que es posible que en el proceso de construcción del corpus, por realizarse mediante un procedimiento manual, existan varias oraciones que no fueron marcadas como citas y que el módulo detecta como citas válidas.
- Existen algunos complejos grupos de palabras que se relacionan mediante su sintaxis, los cuales pueden representar una entidad. Estas entidades no pueden ser identificadas mediante el uso de expresiones regulares, razón por la cual siempre quedarán fuera de este tipo de análisis.

Existe un error de cálculo relacionado a algunos valores de recall en el patrón *Entidad+Verbo*, los cuales son superiores al máximo posible del índice que es uno. Esto se debe a que algunas de las oraciones se cortaron en 2 o más partes y cada parte hizo referencia a la misma cita del corpus, de esta forma es posible tener más de una oración considerada como correcta. Por construcción de la solución, no es posible corregir este problema.

5.7. Análisis comparativos

Como se puede apreciar en los resultados obtenidos de ambos experimentos, existe una baja en todos los indicadores respecto de las pruebas de laboratorio, lo que según los análisis puede estar justificado en varios puntos.

5.7.1. Respecto de los resultados

Al comparar los resultados obtenidos por índice, se puede ver que:

Precision En los experimento de laboratorio, se alcanza el valor óptimo en varias pruebas, sin embargo, estos resultados siempre estuvieron acompañados de bajos valores para *Recall*. Mientras que en las pruebas con documentos reales, el mejor valor obtenido fue de 0,4101 lo que es considerado deficiente.

Recall Este índice en términos generales fue el que mejor se comportó. A pesar de que sólo alcanzó el óptimo para algunos casos aislados, obtuvo valores sobre 0,8 para muchas mediciones, lo que es considerado bueno. Los menores valores de este indicador se alcanzaron cuando el valor de *Precision* llegó a uno. El mejor valor para los experimentos controlados fue de 0,9478 mientras que para los reales fue de 0,9105 lo que representa una diferencia de -0,0373 entre ellos.

F-Measure Centrando la comparación en el mejor valor obtenido por cada experimento, podemos ver que el mejor valor para el laboratorio estuvo en el patrón de *Referencia* con un valor de 0,9704 mientras que el mejor valor para las pruebas en ambiente real fueron para el patrón *Verbo+Referencia* que obtuvo 0,5490 lo que representa una diferencia de -0,4214 resultado que se justifica principalmente en los malos resultados obtenidos en el índice *Precision*.

Capítulo 6

Conclusiones

El principal objetivo es *Diseñar, construir, probar e integrar un módulo con algoritmos de análisis y detección de citas bibliográficas para el sistema de análisis de originalidad DOCODE 2.0*, el cual ha sido logrado con éxito. Para ordenar la conclusiones, se separan respecto de los temas más desarrollados: Mejoras para DOCODE 2.0 y desarrollo del módulo de detección de citas bibliográficas.

6.1. Respecto de DOCODE 2.0

Se ha concluído el desarrollo de las mejoras para la plataforma DOCODE 2.0. Estas mejoras modificaron casi toda la arquitectura de la solución anterior y se potenciaron aspectos fundamentales relacionados con mejorar la calidad del servicio ofrecido.

Este rediseño de DOCODE sitúa a la plataforma en un nivel cuatro de madurez de SOA (de siete posibles) según el modelo OSIMM [1]. Esta evaluación es debido principalmente a la infraestrctura actual y que todavía no se han implementado todas las herramientas de SOA. Mejorando estos puntos, se podría pasar rápidamente a un nivel seis, porque se considera la posibilidad de utilizar una arquitectura de servicios virtualizados, utilizando BPM y BAM como ejes principales.

Las aplicaciones más importantes están relacionadas con el crecimiento. La solución actual es completamente escalable (tanto horizontal como verticalmente), además de permitir esquemas de alta disponibilidad, todo esto de manera transparente para el cliente que consume los servicios.

Otra recomendación es tratar de mantener el encapsulamiento de los algoritmos, ya que la falta de una interfaz para ellos significó muchas horas de esfuerzo y dedicación en revisar cada una de las implementaciones ya existentes y hacerlas trabajar de manera homogénea.

[1] Mayores detalles revisar [16].

Este trabajo, podría formar parte de una metodología de migración de arquitectura de sistemas para pasar hacia una arquitectura orientada a servicios.

Trabajo futuro

- Una vez terminada la implementación del modelo de servicios, el siguiente paso es analizar los procesos involucrados para implementar una capa de *BPM*. Mediante ella, se estructurarán nuevos y mejores procesos de negocio, medibles y replicables.
- Se debe evaluar la incorporación de otras herramientas de SOA como por ejemplo: BAM, Registry, Identity Service y API Administrator, entre otras.
- Es necesario crear un RoadMap que permita tener claridad respecto de los objetivos que cubrirá DOCODE, principalmente para priorizar las tareas del equipo.
- Falta realizar pruebas de carga sobre la plataforma, ya que las pruebas realizadas hasta ahora no permitieron botar la estrutura, razón por la cual no se ha establecido la máxima carga que soporta la nueva solución.
- Se deben explorar algunas alternativas de crecimiento, como por ejemplo el uso de *Amazon Web Services*, lo que permitirá tener una infraestructura escalable tanto horizontal como verticalmente, que se ajuste a los requerimientos del servicio, pero por sobre todo, independiente de la administración interna.
- Existen otros modelos de negocio, asociados a servicios en *Cloud*, que se pueden explorar: *Infrastructure as a Service (IaaS)*, *Platform as a service (PaaS)* y *Software as a service (SaaS)*. A pesar de que conocemos las limitantes técnicas y políticas de implementar estas soluciones, no se deben dejar de investigar estos alcances.

6.2. Respecto del módulo de citas bibliográficas

Se concluye el desarrollo del módulo de citas bibliográficas integrado dentro de DOCODE 2.0 como otro índice de análisis de originalidad, permitiendo cubrir una de las deficiencias más complejas que arrastraba el sistema desde sus inicios.

El desarrollo del módulo otorga una nueva herramienta para que los profesores puedan utilizar DOCODE como una guía que les permita detectar problemas en la forma de citar de sus alumnos y de esta forma mejorar la calidad de los trabajos que entregan, lo que debiese a su vez impactar en mejores evaluaciones para los pupilos.

Se recomienda hacer un seguimiento de los resultados con documentos reales durante las primeras semanas de utilización del sistema, para poder refinar y ajustar los resultados. Además es posible utilizar este módulo para extraer las citas antes de ejecutar el resto de los análisis que tiene DOCODE lo que permitirá mejorar la calidad todos los índices.

Trabajo futuro

- Es posible refinar la investigación de las expresiones regulares utilizadas para detectar referencias, a fin determinar cual de ellas es la mejor y buscar nuevas expresiones que apunten a detectar las citas que hasta el momento no han sido detectadas.

- El proceso de detección de citas queda cubierto de manera suficiente. Sin embargo, la clasificación de las citas es un problema complejo que no pudo ser resuelto con buena calidad durante el desarrollo de esta investigación, a pesar que la definición es clara, las estructuras que se buscan son de difícil detección.

- La aproximación implementada en este trabajo es mediante el uso de expresiones regulares, razón por la cual es necesario el correcto uso de mayúsculas y minúsculas, además del formato de las distintas expresiones analizadas, esto se puede mejorar utilizando, por ejemplo, NLP para la detección de entidades, lematización para la detección de verbos, una herramienta de *POS Tagging* para análisis gramatical, entre otras herramientas más sofisticadas.

- Algunas de las fallas más frecuentes están asociadas a las complejidades del tokenizador de oraciones. En este sentido, todavía hay investigaciones que pretenden mejorar la calidad de las herramientas dedicadas a este punto. Sin embargo, es un problema que no puede ser completamente resuelto según explica Grefenstette et al. en *What is a word? What is a sentence? Problems of Tokenization* [15].

6.3. Comentarios finales

El mayor aporte de esta investigación es que se ha demostrado que es posible parametrizar la detección de citas bibliográficas y que por lo tanto mediante el uso de una herramienta computacional es posible identificarlas dentro de un texto.

Esta conclusión obedece a uno de los principios fundamentales de la computación y es que dado lo anterior es posible crear una máquina de estados que responda al problema de detectar una cita bibliográfica.

Referencias

[1] Milena Araya, Patricio Moya, Yerko Covacevich, and Juan D. Velásquez. Mecanismos de citación en tesis de pregrado de la Universidad de Chile. *VII Encuentro Nacional de la Asociación Latinoamericana de Estudios del Discurso ALED*, 7, 2012.

[2] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1st edition, 5 1999.

[3] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice (2nd Edition)*. Addison-Wesley Professional, 2 edition, 4 2003.

[4] K. Bontcheva, M. Dimitrov, D. Maynard, V. Tablan, and H. Cunningham. Shallow methods for named entity coreference resolution. In *Chaınes de références et résolveurs d'anaphores, workshop TALN*, 2002.

[5] F. Bravo-Marquez, G. L'Huillier, S. Ríos, J. Velásquez, and L. Guerrero. DOCODE-lite: a meta-search engine for document similarity retrieval. *Knowledge-Based and Intelligent Information and Engineering Systems*, pages 93–102, 2010.

[6] Binildas A. Christudas, Malhar Barai, and Vincenzo Caselli. *Service Oriented Architecture with Java: Using SOA and web services to build powerful Java applications*. Packt Publishing, 6 2008.

[7] L. Danlos, B. Sagot, R. Stern, et al. Analyse discursive des incises de citation. In *Actes du Second Colloque Mondial de Linguistique Française", New-Orleans, USA*, 2010.

[8] Ministerio de Educación de Brasil. Domínio Publico, Pesquisa básica. http://www.dominiopublico.gov.br, 2004. [Online; Visitada Dic-2012].

[9] É. de La Clergerie, B. Sagot, R. Stern, P. Denis, G. Recourcé, and V. Mignot. Extracting and visualizing quotations from news wires. *Human Language Technology. Challenges for Computer Science and Linguistics*, pages 522–532, 2011.

[10] Á. Díaz. *América Latina y el Caribe: La propiedad intelectual después de los tratados de libre comercio*. Naciones Unidas, CEPAL, 2008.

[11] Len DiMaggio, Kevin Conner, Magesh B. Kumar, and Tom Cunningham. *JBoss ESB Beginner's Guide*. Packt Publishing, 1 2012.

[12] M. Dimitrov, K. Bontcheva, H. Cunningham, D. Maynard, et al. A light-weight approach to coreference resolution for named entities in text. In *Proceedings of the Fourth Discourse Anaphora and Anaphor Resolution Colloquium (DAARC), Lisbon*. Citeseer, 2002.

[13] Martin Porter et al. Snowball. http://snowball.tartarus.org/, 2006. [Online; Visitada Sep-2012].

[14] EXCOM. EXCOM Site. http://www.excom.fr, 2008. [Site Online].

[15] Gregory Grefenstette and Pasi Tapanainen. What is a word, What is a sentence? Problems of Tokenization. pages 79–87, 1994.

[16] The Open Group. The model. http://www.opengroup.org/soa/source-book/osimmv2/model.htm, 1995-2013. [Online; Visitada Dic-2012].

[17] Nicolai M. Josuttis. *SOA in Practice: The Art of Distributed System Design (Theory in Practice)*. O'Reilly Media, 1 edition, 8 2007.

[18] R. Krestel, S. Bergler, R. Witte, et al. Minding the source: Automatic tagging of reported speech in newspaper articles. *Reporter*, 1(5):4, 2008.

[19] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[20] R. Leaman, G. Gonzalez, et al. Banner: an executable survey of advances in biomedical named entity recognition. In *Pacific Symposium on Biocomputing*, volume 13, pages 652–663, 2008.

[21] Francesco Marchioni. *JBoss AS 7 Configuration, Deployment and Administration*. Packt Publishing, 12 2011.

[22] Moodle. Moodle.org: open-source community-based tools for learning. https://moodle.org/, 2009. [Site Online].

[23] P. Moya Muñoz. Plagio e integración de fuentes múltiples en textos de estudiantes de primer año de universidad: aproximación desde la teoría de la comprensión y el análisis semántico latente. 2011.

[24] L. O'Brien, P. Merson, and L. Bass. Quality attributes for service-oriented architectures. In *Proceedings of the International Workshop on Systems Development in SOA Environments*, page 3. IEEE Computer Society, 2007.

[25] University of Chicago Press Staff, editor. *The Chicago Manual of Style, 16th Edition*. University Of Chicago Press, 16 edition, 8 2010.

[26] OASIS Open (Advancing open standards for the information society). OASIS Web Services Secure Exchange (WS-SX) TC. https://www.oasis-open.org/committees/ws-sx, 2007. [Online; Accessed 22-Jun-2012].

[27] Oracle Systems (Oracle). Java EE 6 Technologies. http://www.oracle.com/technetwork/java/javaee/tech/index.html, 2005. [Online; Visitada Sep-2012].

[28] Oracle Systems (Oracle). JAX-WS Reference Implementation. http://jax-ws.java.net/, 2008. [Online; Visitada Sep-2012].

[29] Oracle Systems (Oracle). Glassfish - Open Source Application Server. http://glassfish.java.net/, 2010. [Online; Visitada Jun-2012].

[30] Oracle Systems (Oracle). Java EE Technical Documentation. http://docs.oracle.com/javaee/, 2010. [Online; Visitada Sep-2012].

[31] Debu Panda, Reza Rahman, and Derek Lane. *EJB 3 in Action*. Manning Publications, 1 edition, 4 2007.

[32] MINISTERIO DE EDUCACIÓN PÚBLICA. Ley 17.336 02-OCT-1970 Ministerio de Educación Pública. *Publicada en el Diario Oficial N° 27.761 de 2 de octubre de 1970*, 1970.

[33] B. Pouliquen, R. Steinberger, and C. Best. Automatic detection of quotations in multilingual news. In *Proceedings of Recent Advances in Natural Language Processing*, pages 487–492, 2007.

[34] B. Pouliquen, R. Steinberger, C. Ignat, I. Temnikova, A. Widiger, W. Zaghouani, and J. Zizka. Multilingual person name recognition and transliteration. *arXiv preprint cs/0609051*, 2006.

[35] V. Qazvinian and D.R. Radev. Identifying non-explicit citing sentences for citation-based summarization. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 555–564. Association for Computational Linguistics, 2010.

[36] Área de Infotecnologías (ADI). U-Cursos, Una Herramienta de Apoyo a la Docencia Presencial. https://www.u-cursos.cl/, 1998. [Plataforma Online desde 1998].

[37] M. Romanello, F. Boschetti, and G. Crane. Citations in the digital library of classics: extracting canonical references by using conditional random fields. In *Proceedings of the 2009 Workshop on Text and Citation Analysis for Scholarly Digital Libraries*, pages 80–87. Association for Computational Linguistics, 2009.

[38] B. Sagot, L. Danlos, R. Stern, et al. A lexicon of french quotation verbs for automatic quotation extraction. In *7th International Conference on Language Resources and Evaluation-LREC 2010*, 2010.

[39] SUN. Designing Enterprise Applications with the J2EE Platform. http://java.sun.com/blueprints/guidelines/designing_enterprise_applications, 2008. [Online; visitada 20-May-2012].

[40] J. Swales. *Genre Analysis: English in Academic and Research Settings: Cambridge Applied Linguistics*. Cambridge University Press, 1990.

[41] J.D. Velásquez and V. Palade. Adaptive web Sites. A knowledge extraction from web data approach. In *Proceeding of the 2008 conference on Adaptive Web Sites: A Knowledge Extraction from Web Data Approach*, pages 1–272. IOS Press, 2008.

[42] W3C. SOAP Specifications. http://www.w3.org/TR/soap/, 2007. [Online; Accessed 15-Jun-2012].

Apéndices

A . Definición funcional de DOCODE mediante casos de uso

Nombre del caso de uso	Agregar Profesor	
Descripción Corta	Proceso mediante el cual se crea un nuevo profesor al sistema, se asignándolo a una institución en particular	
Perfiles	Super Administrador, Administrador	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
Email del profesor RUT del profesor Primer Nombre del profesor Segundo nombre del profesor Apellido Paterno del Profesor Apellido Materno del Profesor Sexo del Profesor Fecha nacimiento del Profesor	Administrador	Nuevo profesor con las características entregadas, asignándolo a una institución específica
Descripción del proceso		
1. Acceder al sistema 2. Seleccionar la institución donde se quiere agregar un nuevo profesor 3. Seleccionar la opción Profesores 4. Seleccionar agregar profesor 5. Ingresar los datos requeridos 6. Presionar el botón *Guardar*		

Cuadro 6.1: Caso de uso: Agregar Profesor.

Nombre del caso de uso	Editar datos Profesor	
Descripción Corta	Proceso mediante el cual se modifican datos de un profesor ya creado y asignado a una Institución	
Perfiles	Super Administrador, Administrador	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
Email del profesor RUT del profesor Primer Nombre del profesor Segundo nombre del profesor Apellido Paterno del Profesor Apellido Materno del Profesor Sexo del Profesor Fecha nacimiento del Profesor	Administrador	Profesor con las características modificadas
Descripción del proceso		
1. Acceder al sistema 2. Seleccionar la institución a la que pertenece el profesor a modificar 3. Seleccionar la opción Profesores 4. Seleccionar la opción *Editar*, correspondiente al profesor que se desea modificar 5. Ingresar los datos requeridos 6. Presionar el botón *Guardar*		

Cuadro 6.2: Caso de uso: Editar datos Profesor.

Nombre del caso de uso	Borrar Profesor	
Descripción Corta	Proceso mediante el cual se elimina un profesor ya creado y asignado a una institución	
Perfiles	Super Administrador, Administrador	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
No requiere datos	Administrador	Eliminación del profesor
Descripción del proceso		
1. Acceder al sistema como administrador 2. Seleccionar la institución a la que pertenece el profesor a eliminar 3. Seleccionar la opción Profesores 4. Seleccionar la opción *Borrar*, correspondiente al profesor que se desea modificar 5. Presionar *Aceptar*		

Cuadro 6.3: Caso de uso: Borrar Profesor.

Nombre del caso de uso	Agregar Lista de Curso	
Descripción Corta	Proceso mediante el cual se crea una nueva lista de curso, asignándola a una institución	
Perfiles	Super Administrador, Profesor	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
Nombre del Curso Datos de él/los alumnos pertenecientes a la Lista	Administrador	Nueva Lista con las características entregadas asignando a una institución específica
Descripción del proceso		
1. Acceder al sistema como administrador 2. Seleccionar la institución a la que se desea agregar una nueva Lista 3. Seleccionar la opción Listas 4. Seleccionar la opción *Agregar* 5. Ingresar los datos requerido 6. Presionar *Guardar*		

Cuadro 6.4: Caso de uso: Agregar Lista de Curso.

Nombre del caso de uso	Editar Lista de Curso	
Descripción Corta	Proceso mediante el cual se modifican datos de una lista de curso, alumnos o nombre de la Lista.	
Perfiles	Super Administrador, Profesor	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
Nombre del Curso Datos de él/los alumnos pertenecientes a la Lista	Administrador	Nueva Lista con las características modificadas
Descripción del proceso		
1. Acceder al sistema como administrador 2. Seleccionar la institución a la que se desea agregar una nueva Lista 3. Seleccionar la opción Listas 4. Seleccionar la opción *Editar*, correspondiente a la Lista que se desea modificar 5. Ingresar los datos requerido 6. Presionar *Guardar*		

Cuadro 6.5: Caso de uso: Editar Lista de Curso.

Nombre del caso de uso	Borrar Lista de Curso	
Descripción Corta	Proceso mediante el cual se elimina una Lista de Curso ya creada y asignada a una institución.	
Perfiles	Super Administrador, Profesor	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
No requiere datos	Administrador	Eliminación de la Lista de Curso
Descripción del proceso		
1. Acceder al sistema como administrador 2. Seleccionar la institución a la que pertenece el profesor a eliminar 3. Seleccionar la opción Listas 4. Seleccionar la opción *Borrar*, correspondiente a la Lista de Curso que se desea modificar 5. Presionar *Aceptar*		

Cuadro 6.6: Caso de uso: Borrar Lista de Curso.

Nombre del caso de uso	Agregar Alumno a Lista de Curso	
Descripción Corta	Proceso mediante el cual se agrega un nuevo Alumno a una Lista de Curso ya creada y asignada a una institución	
Perfiles	Super Administrador, Profesor	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
Datos del alumno E-Mail Nombre 1 Nombre 2 Ap. paterno Ap. materno	Administrador	Nueva Lista con las características entregadas asignando a una institución específica
Descripción del proceso		
1. Acceder al sistema como administrador 2. Seleccionar la institución a la que se desea agregar un nuevo alumno 3. Seleccionar la opción Listas 4. Seleccionar la opción *Editar*, correspondiente a la Lista a la que se le desea agregar un nuevo alumno 5. Ingresar los datos requeridos (entradas especificadas) 6. Presionar *Guardar*		

Cuadro 6.7: Caso de uso: Agregar alumno en Lista de Curso.

Nombre del caso de uso	Agregar Tarea	
Descripción Corta	Proceso mediante el cual se crea una nueva tarea, asignándola a una única Lista de Curso	
Perfiles	Super Administrador, Profesor	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
Título de tarea Tipo de tarea (descripción) Palabras Claves Áreas de trabajo Curso asignado a la tarea Descripción de la tarea Archivo Adjunto Período de entrega	Administrador	Nueva Tarea asignada a una única Lista de Curso
Descripción del proceso		
1. Acceder al sistema 2. Seleccionar la institución a la que se desea agregar una nueva Tarea 3. Seleccionar la opción Tareas 4. Seleccionar la opción *Agregar* 5. Ingresar los datos requerido 6. Presionar *Guardar*		

Cuadro 6.8: Caso de uso: Agregar Tarea.

Nombre del caso de uso	Editar Tarea	
Descripción Corta	Proceso mediante el cual se modifican datos de una tarea ya creada	
Perfiles	Super Administrador, Profesor	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
Título de tarea Tipo de tarea (descripción) Palabras Claves Áreas de trabajo Curso asignado a la tarea Descripción de la tarea Archivo Adjunto Período de entrega	Administrador	Tarea con datos modificados
Descripción del proceso		
1. Acceder al sistema 2. Seleccionar la institución en la que se desea modificar una Tarea 3. Seleccionar la opción Tareas 4. Seleccionar la opción *Editar*, correspondiente a la Tarea que se desea modificar 5. Ingresar los datos requeridos 6. Presionar *Guardar*		

Cuadro 6.9: Caso de uso: Editar Tarea.

Nombre del caso de uso	Borrar Tarea	
Descripción Corta	Proceso mediante el cual se elimina una Tarea ya creada y asignada a una Lista de Curso	
Perfiles	Super Administrador, Profesor	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
No requiere datos	Administrador	Eliminación de la Lista de Curso
Descripción del proceso		

1. Acceder al sistema
2. Seleccionar la institución en la que se quiere eliminar la Tarea
3. Seleccionar la opción Tareas
4. Seleccionar la opción *Borrar*, correspondiente a la Tarea que se desea modificar
5. Presionar *Aceptar*

Cuadro 6.10: Caso de uso: Borrar Tareas.

Nombre del caso de uso	Revisión de Entregas o Descargas de Tareas	
Descripción Corta	Revisión de la entrega de tareas	
Perfiles	Super Administrador, Profesor, Alumno	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
No requiere datos	Administrador	Descarga de las tareas entregadas
Descripción del proceso		

1. Acceder al sistema
2. Seleccionar la institución en la que se desea revisar una nueva Tarea
3. Seleccionar la opción Tareas
4. Seleccionar la opción *Entregas*, correspondiente a la tarea que se desea revisar
5. Seleccionar las tareas que se deseen descargar
6. Presionar *Descargar*

1. Acceder al sistema utilizando el correo electrónico
2. Seleccionar la opción Tareas
3. Seleccionar la opción *Entregas*, correspondiente a la tarea que se desea revisar
4. Seleccionar las tareas que se deseen revisar
5. Presionar *Descargar*

Cuadro 6.11: Caso de uso: Revisión de entregas y descarga de tareas.

Nombre del caso de uso	Revisión de reportes de análisis de originalidad de documentos	
Descripción Corta	Opción de revisión de detección de copia de documentos	
Perfiles	Super Administrador, Profesor	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
Tareas entregadas	Alumnos	Reportes agregados y específicos por documento
Descripción del proceso		
1. Acceder al sistema. 2. Una vez entregadas las tareas que el profesor desea revisar, se puede solicitar reporte DOCODE. 3. Esperar que la solicitud sea procesada por el sistema DOCODE 4. Recibir los reportes agregados para todos los documentos y específicos por cada documento. 5. Revisar reportes agregados. 6. En caso de ser necesario, el profesor puede revisar los reportes específicos por cada documento.		

Cuadro 6.12: Caso de uso: Revisión de reportes de análisis de originalidad.

Nombre del caso de uso	Agregar Instituciones	
Descripción Corta	Proceso mediante el cual se crea una nueva Institución	
Perfiles	Super Administrador, Administrador	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
Identificador de la Institución Estado (activado/desactivado) Nombre de la Institución Dirección de la Institución Código Postal Administrador Nombre, fono, email contacto 1 Nombre, fono, email contacto 2	Administrador	Nueva Institución
Descripción del proceso		
1. Acceder al sistema 2. Seleccionar la opción Administrar 3. Seleccionar la opción Instituciones 4. Seleccionar la opción *Agregar* 5. Ingresar los datos requeridos 6. Presionar *Guardar*		

Cuadro 6.13: Caso de uso: Agregar Instituciones.

Nombre del caso de uso	Editar Instituciones	
Descripción Corta	Proceso mediante el cual se modifican datos de una Institución	
Perfiles	Super Administrador, Administrador	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
Identificador de la Institución Estado (activado/desactivado) Nombre de la Institución Dirección de la Institución Código Postal Administrador Nombre, fono, email contacto 1 Nombre, fono, email contacto 2	Administrador	Nueva Institución
Descripción del proceso		
1. Acceder al sistema como administrador 2. Seleccionar la opción Administrar 3. Seleccionar la opción Instituciones 4. Seleccionar la opción *Editar*, correspondiente a la Institución que se desea modificar 5. Ingresar los datos requeridos 6. Presionar *Guardar*		

Cuadro 6.14: Caso de uso: Editar Instituciones.

Nombre del caso de uso	Habilitar/Desahabilitar Instituciones	
Descripción Corta	Proceso mediante el cual se Habilita/Deshabilita una Institución	
Perfiles	Super Administrador	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
No requiere datos	Administrador	Institución Habilitada/Deshabilitada
Descripción del proceso		
1. Acceder al sistema como administrador 2. Seleccionar la opción Administrar 3. Seleccionar la opción Instituciones 4. Seleccionar la opción *Deshabilitar*, correspondiente a la Institución que se desea Deshabilitar 5. Ingresar los datos requerido 6. Presionar *Aceptar*		

Cuadro 6.15: Caso de uso: Habilitar/Desahabilitar Instituciones.

Nombre del caso de uso	Agregar Persona	
Descripción Corta	Proceso mediante el cual se crea una nueva Persona	
Perfiles	Super Administrador	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
Email RUT Primer Nombre Segundo Nombre Apellido Paterno Apellido Materno Sexo Fecha de Nacimiento Estado	Administrador	Nueva Persona
Descripción del proceso		
1. Acceder al sistema 2. Seleccionar la opción Administrar 3. Seleccionar la opción Personas 4. Seleccionar la opción *Agregar* 5. Ingresar los datos requerido 6. Presionar *Guardar*		

Cuadro 6.16: Caso de uso: Agregar Persona.

Nombre del caso de uso	Editar Persona	
Descripción Corta	Proceso mediante el cual se modifican datos de una Persona	
Perfiles	Super Administrador	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
Email RUT Primer Nombre Segundo Nombre Apellido Paterno Apellido Materno Sexo Fecha de Nacimiento Estado	Administrador	Persona con datos modificados
Descripción del proceso		
1. Acceder al sistema 2. Seleccionar la opción Administrar 3. Seleccionar la opción Personas 4. Seleccionar la opción *Editar*, correspondiente a la persona que se desea modificar 5. Ingresar los datos requerido 6. Presionar *Guardar*		

Cuadro 6.17: Caso de uso: Editar Persona.

Nombre del caso de uso	Habilitar o Desahabilitar Persona	
Descripción Corta	Proceso mediante el cual se Habilita o Deshabilita una Persona existente en el sistema	
Perfiles	Super Administrador	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
No requiere datos	Administrador	Persona Habilitada o Deshabilitada
Descripción del proceso		
1. Acceder al sistema 2. Seleccionar la opción Administrar 3. Seleccionar la opción Personas 4. Seleccionar la opción *Deshabilitar*, correspondiente a la Persona que se desea Deshabilitar 5. Ingresar los datos requerido 6. Presionar *Aceptar*		

Cuadro 6.18: Caso de uso: Habilitar o Deshabilitar Persona.

Nombre del caso de uso	Borrar Persona	
Descripción Corta	Proceso mediante el cual se elimina una Persona ya creada	
Perfiles	Super Administrador	
Entradas proceso	**Quien lo entrega**	**Salida del proceso**
No requiere datos	Administrador	Eliminación de la Persona del sistema
Descripción del proceso		
1. Acceder al sistema 2. Seleccionar la opción Administrar 3. Seleccionar la opción Personas 4. Seleccionar la opción *Borrar*, correspondiente a la persona que se desea eliminar 5. Presionar *Aceptar*		

Cuadro 6.19: Caso de uso: Borrar Persona.

B . XSD Definición estructuras del servicio de envío y consultas de tareas DOCODE 2.0

```
<?xml version="1.0"?>
<xs:schema version="1.0"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:tn="http://cl.docode.xml/DocodeEE/service"
           targetNamespace="http://cl.docode.xml/DocodeEE/service"
           elementFormDefault="unqualified"
           attributeFormDefault="unqualified">
<xs:element name="QueryRequest" type="tn:QueryRequestType"/>

<xs:complexType name="QueryRequestType">
  <xs:sequence>
    <xs:element name="query" minOccurs="1" maxOccurs="1">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="All"/>
          <xs:enumeration value="WR"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="User" type="tn:UserType" minOccurs="0" maxOccurs="1"/>
    <xs:element name="Task" type="tn:TaskType" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="UserType">
  <xs:sequence>
    <xs:element name="Nombre" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="Correo" type="xs:string" minOccurs="1" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="TaskType">
  <xs:sequence>
    <xs:element name="id" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="titulo" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="descripcion" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="tipo" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="area" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="tag" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="FileQuantity" type="xs:int" minOccurs="1" maxOccurs="1"/>
    <xs:element name="File" type="tn:FileType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="FileType">
  <xs:sequence>
    <xs:element name="fileId" type="xs:int" minOccurs="1" maxOccurs="1"/>
```

```
    <xs:element name="nombre" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="largo" type="xs:long" minOccurs="1" maxOccurs="1"/>
    <xs:element name="data" type="xs:hexBinary" minOccurs="1" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="QueryResponseType">
  <xs:sequence>
    <xs:element name="id" type="xs:int" minOccurs="1" maxOccurs="1"/>
    <xs:element name="description" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="token" type="xs:string" minOccurs="1" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```

C . WSDL del servicio de envío y consultas de tareas DOCODE 2.0

```
<definitions
     name='QueryServiceWS'
     targetNamespace='http://ws.docode.cl/'
     xmlns='http://schemas.xmlsoap.org/wsdl/'
     xmlns:ns1='http://cl.docode.xml/DocodeEE/service'
     xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
     xmlns:tns='http://ws.docode.cl/'
     xmlns:xsd='http://www.w3.org/2001/XMLSchema'>

 <types>

  <xs:schema
     targetNamespace='http://ws.docode.cl/'
     version='1.0'
     xmlns:ns1='http://cl.docode.xml/DocodeEE/service'
     xmlns:tns='http://ws.docode.cl/'
     xmlns:xs='http://www.w3.org/2001/XMLSchema'>

   <xs:import namespace='http://cl.docode.xml/DocodeEE/service'/>
   <xs:element name='doQuery' type='tns:doQuery'/>
   <xs:element name='doQueryResponse' type='tns:doQueryResponse'/>

   <xs:complexType name='doQuery'>
    <xs:sequence>
     <xs:element minOccurs='0' name='queryRequest' type='ns1:QueryRequestType'/>
    </xs:sequence>
   </xs:complexType>

   <xs:complexType name='doQueryResponse'>
    <xs:sequence>
     <xs:element minOccurs='0' name='queryResponse' type='ns1:QueryResponseType'/>
    </xs:sequence>
   </xs:complexType>

  </xs:schema>

  <xs:schema
      targetNamespace='http://cl.docode.xml/DocodeEE/service'
      version='1.0'
      xmlns:tns='http://cl.docode.xml/DocodeEE/service'
      xmlns:xs='http://www.w3.org/2001/XMLSchema'>

   <xs:complexType name='QueryRequestType'>
    <xs:sequence>
     <xs:element name='query' type='xs:string'/>
     <xs:element minOccurs='0' name='User' type='tns:UserType'/>
```

```
    <xs:element minOccurs='0' name='Task' type='tns:TaskType'/>
   </xs:sequence>
  </xs:complexType>

  <xs:complexType name='UserType'>
   <xs:sequence>
    <xs:element name='Nombre' type='xs:string'/>
    <xs:element name='Correo' type='xs:string'/>
   </xs:sequence>
  </xs:complexType>

  <xs:complexType name='TaskType'>
   <xs:sequence>
    <xs:element name='id' type='xs:string'/>
    <xs:element name='titulo' type='xs:string'/>
    <xs:element name='descripcion' type='xs:string'/>
    <xs:element name='tipo' type='xs:string'/>
    <xs:element name='area' type='xs:string'/>
    <xs:element name='tag' type='xs:string'/>
    <xs:element name='FileQuantity' type='xs:int'/>
    <xs:element maxOccurs='unbounded' minOccurs='0' name='File' type='tns:FileType'/>
   </xs:sequence>
  </xs:complexType>

  <xs:complexType name='FileType'>
   <xs:sequence>
    <xs:element name='fileId' type='xs:int'/>
    <xs:element name='nombre' type='xs:string'/>
    <xs:element name='largo' type='xs:long'/>
    <xs:element name='data' type='xs:hexBinary'/>
   </xs:sequence>
  </xs:complexType>

  <xs:complexType name='QueryResponseType'>
   <xs:sequence>
    <xs:element name='id' type='xs:int'/>
    <xs:element name='description' type='xs:string'/>
    <xs:element name='token' type='xs:string'/>
   </xs:sequence>
  </xs:complexType>

 </xs:schema>

</types>

<message name='QueryServiceWS_doQueryResponse'>
 <part element='tns:doQueryResponse' name='doQueryResponse'></part>
</message>

<message name='QueryServiceWS_doQuery'>
```

```
  <part element='tns:doQuery' name='doQuery'></part>
 </message>

 <portType name='QueryServiceWS'>
  <operation name='doQuery' parameterOrder='doQuery'>
   <input message='tns:QueryServiceWS_doQuery'></input>
   <output message='tns:QueryServiceWS_doQueryResponse'></output>
  </operation>
 </portType>

 <binding name='QueryServiceWSBinding' type='tns:QueryServiceWS'>
  <soap:binding style='document' transport='http://schemas.xmlsoap.org/soap/http'/>
  <operation name='doQuery'>
   <soap:operation soapAction=''/>
   <input>
    <soap:body use='literal'/>
   </input>
   <output>
    <soap:body use='literal'/>
   </output>
  </operation>
 </binding>

 <service name='QueryServiceWS'>
  <port binding='tns:QueryServiceWSBinding' name='QueryServiceWSPort'>
   <soap:address location='http://localhost:9090/docode/QueryServiceWS'/>
  </port>
 </service>

</definitions>
```

D . Extracto código fuente del MDB

```
package cl.docode.fastdocode.ejb;

import cl.docode.library.utils.Propiedades;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.EJB;
import javax.ejb.MessageDriven;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;

@MessageDriven(mappedName = "FastDocodeQueue", activationConfig = {
    @ActivationConfigProperty(propertyName = "acknowledgeMode",
                              propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName = "destinationType",
                              propertyValue = "javax.jms.Queue")
})
public class FastDocodeMDB implements MessageListener {
    @EJB
    private FastDocodeSBLocal fastDocodeSB;

    public FastDocodeMDB() {   }

    @Override
    public void onMessage(Message message) {
        if(message instanceof TextMessage){
            TextMessage msg= (TextMessage)message;
            try {
                fastDocodeSB.execute(msg.getText());
            } catch (JMSException ex) {
                Logger.getLogger(
                  FastDocodeMDB.class.getName()
                  ).log(Level.SEVERE, null, ex);
            }
        }
    }
}
```

E . Extracto código fuente del EJB

```
package cl.docode.fastdocode.ejb;

import cl.docode.library.database.ManagerDB;
import cl.docode.library.utils.Propiedades;
import java.io.File;
import javax.ejb.Stateful;

@Stateful
public class FastDocodeSB implements FastDocodeSBLocal {

    @Override
    public void execute(String token) {
...
        executeFastDocode(token, path_fd_tareas, path_fd_fuentes, 3);
    }

    public boolean executeFastDocode(
                      String token,
                      String path_fd_tareas,
                      String path_fd_fuentes,
                      int idAlgoritmo) {
        try{
          ManagerDB.iniciaAlgoritmo(
               token,
               idAlgoritmo);
        } catch(Exception e){
            e.printStackTrace();
        }
        try {
          cl.docode.fastdocode.algoritmo2.fastdocode.main(
               new String[]{
                    path_fd_tareas,
                    path_fd_fuentes});
        } catch (Exception ex) {
            System.err.println(
                "Error en Fast Docode..."+ex);
        } finally {
            ManagerDB.actualizaAlgoritmo(
                token,
                idAlgoritmo,
                ManagerDB.ALGORITMO_TERMINA);
        }
        return true;
    }
}
```

F . Extracto código fuente del Entity Bean DocArchivos

```
package cl.docode.persistanceModel.ejb.entities;

import java.io.Serializable;
import java.math.BigDecimal;
import java.util.Collection;
import javax.persistence.*;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;

@Entity
@Table(name = "doc_archivos")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "DocArchivos.findAll",
                query = "SELECT d FROM DocArchivos d"),
    @NamedQuery(name = "DocArchivos.findByArcId",
                query = "SELECT d FROM DocArchivos d WHERE d.arcId = :arcId"),
    @NamedQuery(name = "DocArchivos.findByIdRemoto",
                query = "SELECT d FROM DocArchivos d WHERE d.idRemoto = :idRemoto"),
    @NamedQuery(name = "DocArchivos.findByPath",
                query = "SELECT d FROM DocArchivos d WHERE d.path = :path"),
    @NamedQuery(name = "DocArchivos.findByToken",
                query = "SELECT d FROM DocArchivos d WHERE d.token = :token")})

public class DocArchivos implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @Basic(optional = false)
    @Column(name = "arc_id")
    private BigDecimal arcId;
    @Basic(optional = false)
    @Column(name = "id_remoto")
    private String idRemoto;
    @Basic(optional = false)
    @Column(name = "path")
    private String path;
    @Basic(optional = false)
    @Column(name = "token")
    private String token;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "arcId2")
    private Collection<DocScores> docScoresCollection;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "arcId1")
    private Collection<DocScores> docScoresCollection1;
    @JoinColumn(name = "tar_id", referencedColumnName = "tar_id")
    @ManyToOne(optional = false)
    private DocTareas tarId;
```

```
public DocArchivos() {
}

public DocArchivos(BigDecimal arcId) {
    this.arcId = arcId;
}

public DocArchivos(
            BigDecimal arcId,
            String idRemoto,
            String path,
            String token) {
    this.arcId = arcId;
    this.idRemoto = idRemoto;
    this.path = path;
    this.token = token;
}

public BigDecimal getArcId() {
    return arcId;
}
public void setArcId(BigDecimal arcId) {
    this.arcId = arcId;
}
public String getIdRemoto() {
    return idRemoto;
}
public void setIdRemoto(String idRemoto) {
    this.idRemoto = idRemoto;
}
public String getPath() {
    return path;
}
public void setPath(String path) {
    this.path = path;
}
public String getToken() {
    return token;
}
public void setToken(String token) {
    this.token = token;
}
@XmlTransient
public Collection<DocScores> getDocScoresCollection() {
    return docScoresCollection;
}
public void setDocScoresCollection(Collection<DocScores> docScoresCollection) {
    this.docScoresCollection = docScoresCollection;
}
@XmlTransient
```

```
    public Collection<DocScores> getDocScoresCollection1() {
        return docScoresCollection1;
    }
    public void setDocScoresCollection1(Collection<DocScores> docScoresCollection1) {
        this.docScoresCollection1 = docScoresCollection1;
    }
    public DocTareas getTarId() {
        return tarId;
    }
    public void setTarId(DocTareas tarId) {
        this.tarId = tarId;
    }
    @Override
    public int hashCode() {
        int hash = 0;
        hash += (arcId != null ? arcId.hashCode() : 0);
        return hash;
    }
    @Override
    public boolean equals(Object object) {
        if (!(object instanceof DocArchivos)) {
            return false;
        }
        DocArchivos other = (DocArchivos) object;
        if (
            (this.arcId == null && other.arcId != null) ||
            (this.arcId != null && !this.arcId.equals(other.arcId))
            ) {
            return false;
        }
        return true;
    }
    @Override
    public String toString() {
        return "cl.docode.persistanceModel.ejb.entities.DocArchivos[ arcId=" +
                arcId + " ]";
    }
}
```

G . Ejemplo de persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<persistence version="1.0"
xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
          "http://java.sun.com/xml/ns/persistence
           http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">

  <persistence-unit name="Docode_PersistenceModelPU" transaction-type="JTA">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>java:/docode_ds</jta-data-source>
    <class>cl.docode.persistanceModel.ejb.entities.DocArchivos</class>
    <class>cl.docode.persistanceModel.ejb.entities.DocEjecucion</class>
    <class>cl.docode.persistanceModel.ejb.entities.DocScores</class>
    <class>cl.docode.persistanceModel.ejb.entities.DocTareas</class>
    <exclude-unlisted-classes>true</exclude-unlisted-classes>
  </persistence-unit>

</persistence>
```

H . Ejemplo de archivo de configuracion DataSources

```
<?xml version="1.0" encoding="UTF-8"?>

<datasources>

    <local-tx-datasource>
        <jndi-name>docode_tx_ds</jndi-name>
        <connection-url>jdbc:postgresql://localhost/engine</connection-url>
        <driver-class>org.postgresql.Driver</driver-class>
        <user-name>usuario</user-name>
        <password>password</password>
        <min-pool-size>5</min-pool-size>
        <max-pool-size>50</max-pool-size>
        <idle-timeout-minutes>10</idle-timeout-minutes>
    </local-tx-datasource>

    <no-tx-datasource>
        <jndi-name>docode_ds</jndi-name>
        <connection-url>jdbc:postgresql://localhost/engine</connection-url>
        <driver-class>org.postgresql.Driver</driver-class>
        <user-name>usuario</user-name>
        <password>password</password>
        <min-pool-size>5</min-pool-size>
        <max-pool-size>50</max-pool-size>
        <idle-timeout-minutes>10</idle-timeout-minutes>
    </no-tx-datasource>

   <xa-datasource>
     <jndi-name>docode_xa_ds</jndi-name>
     <xa-datasource-class>org.postgresql.xa.PGXADataSource</xa-datasource-class>
     <xa-datasource-property name="ServerName">localhost</xa-datasource-property>
     <xa-datasource-property name="PortNumber">5432</xa-datasource-property>
     <xa-datasource-property name="DatabaseName">engine</xa-datasource-property>
     <xa-datasource-property name="User">usuario</xa-datasource-property>
     <xa-datasource-property name="Password">password</xa-datasource-property>
     <track-connection-by-tx></track-connection-by-tx>
   </xa-datasource>

</datasources>
```

I . Script de creación de base de datos

```
--
-- Name: lite_tarea; Type: TABLE; Schema: public; Owner: engine; Tablespace:
--

CREATE TABLE lite_tarea (
    tarea_id integer NOT NULL,
    area character varying(255),
    descripcion character varying(255),
    original_id character varying(255),
    pathfile character varying(255),
    procesado smallint,
    tag character varying(255),
    tipo character varying(255),
    titulo character varying(255),
    token character varying(255),
    token_remoto character varying(255),
    usuario_id integer,
    fecha_creacion timestamp without time zone,
    fecha_actualizacion timestamp without time zone,
    remote_address character varying(255)
);


--
-- Name: lite_usuario; Type: TABLE; Schema: public; Owner: engine; Tablespace:
--

CREATE TABLE lite_usuario (
    usuario_id integer NOT NULL,
    correo character varying(128),
    nombre character varying(128),
    pkey character varying(512),
    tipousuario character varying(32),
    username character varying(16),
    ipclient character varying(255)
);


--
-- Name: litetarea_sequence; Type: SEQUENCE; Schema: public; Owner: engine
--

CREATE SEQUENCE litetarea_sequence
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;
```

```
--
-- Name: litetarea_sequence; Type: SEQUENCE SET; Schema: public; Owner: engine
--

SELECT pg_catalog.setval('litetarea_sequence', 222, true);


--
-- Name: t_algoritmos; Type: TABLE; Schema: public; Owner: engine; Tablespace:
--

CREATE TABLE t_algoritmos (
    token character varying(50),
    id integer,
    estado integer
);

--
-- Name: t_archivos; Type: TABLE; Schema: public; Owner: engine; Tablespace:
--

CREATE TABLE t_archivos (
    id character varying(50),
    path character varying(200),
    token character varying(50)
);


--
-- Name: t_descargas; Type: TABLE; Schema: public; Owner: engine; Tablespace:
--

CREATE TABLE t_descargas (
    token character varying(50),
    url character varying(255),
    a_id character varying(50),
    index integer,
    estado integer
);


--
-- Name: t_score; Type: TABLE; Schema: public; Owner: engine; Tablespace:
--

CREATE TABLE t_score (
    id1 character varying(50),
    id2 character varying(50),
    valor real,
```

```
    token character varying(50)
);


--
-- Name: t_tareas; Type: TABLE; Schema: public; Owner: engine; Tablespace:
--

CREATE TABLE t_tareas (
    id character varying(50),
    titulo character varying(100),
    descripcion character varying(200),
    tipo character varying(100),
    area character varying(100),
    tag character varying(200),
    ticket character varying(50) DEFAULT 0,
    procesado integer DEFAULT 0
);
```

J . Listado de verbos de reporte

acentuar
aceptar
aconsejar
acusar
admitir
advertir
afirmar
alabar
alertar
amenazar
analizar
analizar
anunciar
añadir
apoyar
articular
asegurar
atacar
atribuir
basarse en
celebrar
citar
clarificar
coincidir
comentar
comparar
conceder
concluir
confirmar
considerar
contradecir
contrastar
convencer
creer
criticar
cuestionar
culpar
darse cuenta
debatir
declarar
defender
definir
desafiar
descartar
desconocer
describir
descubrir
desestimar
destacar
discurrir
dudar
elogiar
encontrar
enfatizar
entender
enumerar
especular
establecer
estar de acuerdo
estimar
evaluar
examinar
excusar
exhortar
explicar
expresar
garantizar
hacer hincapié
identificar
ignorar
imaginar
indicar
inferir
informar
insinuar
insistir
instar
interpretar
invalidar
investigar
jactarse
justificar
manifestar
mencionar
mostrar
negar
notar
observar
omitir
oponerse a
pensar
persuadir
poner en relieve
postular
presentar
probar
profesar
prohibir
prometer
proponer
razonar
rechazar
recomendar
reconocer
refutar
reiterar
reprochar
resumir
saber
sentir
señalar
sostener
sostener
subrayar
sugerir
suponer
suscribir
valorar

K . Código implementación de interfaz Interprete

```
package cl.docode.quotation.documentParser;

import java.io.File;
import java.io.FileInputStream;
import java.io.StringWriter;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.apache.poi.hwpf.HWPFDocument;
import org.apache.poi.hwpf.converter.WordToHtmlConverter;
import org.apache.poi.hwpf.converter.WordToTextConverter;
import org.apache.poi.hwpf.extractor.WordExtractor;
import org.apache.poi.hwpf.usermodel.CharacterRun;
import org.apache.poi.hwpf.usermodel.Paragraph;
import org.apache.poi.hwpf.usermodel.Range;

public class InterpreteDOC extends Interprete {
    @Override
    public String getText() {
        if (this.document_text == null || this.document_text.length() == 0) {
            this.document_text = DOC_2_TXT_format(this.file);
        }
        return this.document_text;
    }

    private String DOC_2_TXT_noFormat(File filedoc) {
        FileInputStream file = null;
        try {
            file = new FileInputStream(filedoc);
            WordExtractor wordDoc = new WordExtractor(file);
            return wordDoc.getText();
        } catch (Exception ex) {
            System.out.println("Ocurrio una excepcion al parsear el documento DOC.");
            ex.printStackTrace();
        } finally {
            try {
                if (file != null) {
                    file.close();
                }
            } catch (Exception e) {
            }
        }
        return "NO READABLE DATA";
    }
```

```
private String DOCtoHTML(File filedoc) {
    FileInputStream file = null;
    try {
        file = new FileInputStream(filedoc);

        HWPFDocument doc=new HWPFDocument(file);

        org.w3c.dom.Document newDoc =
            DocumentBuilderFactory.newInstance().newDocumentBuilder().newDocument();

        WordToHtmlConverter converter =
            new WordToHtmlConverter(newDoc);
        converter.processDocument(doc);

        StringWriter stringWriter = new StringWriter();
        javax.xml.transform.Transformer transformer =
            TransformerFactory.newInstance().newTransformer();
        transformer.setOutputProperty(OutputKeys.INDENT,"yes");
        transformer.setOutputProperty(OutputKeys.ENCODING,"UTF-8");
        transformer.setOutputProperty(OutputKeys.METHOD,"html");
        transformer.transform(
            new DOMSource(converter.getDocument()), new StreamResult(stringWriter));

        return stringWriter.toString();

    } catch (Exception ex) {
        System.out.println("Ocurrio una excepcion al parsear el documento DOC.");
        ex.printStackTrace();
    } finally {
        try {
            if (file != null) {
                file.close();
            }
        } catch (Exception e) {
        }
    }
    return "NO READABLE DATA";
}

private String DOCtoTEXT(File filedoc) {
    FileInputStream file = null;
    try {
        file = new FileInputStream(filedoc);

        HWPFDocument doc=new HWPFDocument(file);

        org.w3c.dom.Document newDoc =
            DocumentBuilderFactory.newInstance().newDocumentBuilder().newDocument();

        WordToTextConverter converter = new WordToTextConverter(newDoc);
```

```
            converter.processDocument(doc);

            StringWriter stringWriter = new StringWriter();
            javax.xml.transform.Transformer transformer =
                TransformerFactory.newInstance().newTransformer();
            transformer.setOutputProperty(OutputKeys.INDENT,"yes");
            transformer.setOutputProperty(OutputKeys.ENCODING,"UTF-8");
            transformer.setOutputProperty(OutputKeys.STANDALONE,"yes");
            transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION,"yes");
            transformer.transform(
                new DOMSource(converter.getDocument()), new StreamResult(stringWriter));

            return stringWriter.toString();

        } catch (Exception ex) {
            System.out.println("Ocurrio una excepcion al parsear el documento DOC.");
            ex.printStackTrace();
        } finally {
            try {
                if (file != null) {
                    file.close();
                }
            } catch (Exception e) {
            }
        }
        return "NO READABLE DATA";
    }
}
```

L . Código implementación de interfaz DocodeNLP

```
package cl.docode.library.nlp;

import cl.docode.library.utils.Propiedades;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URI;
import java.util.ArrayList;
import java.util.Arrays;
import opennlp.tools.postag.POSModel;
import opennlp.tools.postag.POSTaggerME;
import opennlp.tools.sentdetect.SentenceDetectorME;
import opennlp.tools.sentdetect.SentenceModel;
import opennlp.tools.tokenize.TokenizerME;
import opennlp.tools.tokenize.TokenizerModel;

public class OpenNLPDocodeNLPImpl extends DocodeNLP {

    private static POSModel posModel;
    private static TokenizerModel tokModel;
    private static SentenceModel senModel;

    static {
        cargarModelos();
    }

    private static void cargarModelos() {
        InputStream modelIn = null;
        try {
            try {
                modelIn = new FileInputStream(
                    new URI(Propiedades.getConfigurationPath()
                        + "/models/en-sent.bin").getPath());
            } catch (Exception ex) {
                ex.printStackTrace();
            }
            senModel = new SentenceModel(modelIn);
            if (modelIn != null) {
                try {
                    modelIn.close();
                } catch (IOException e) {
                }
            }
            try {
                modelIn = new FileInputStream(
                    new URI(Propiedades.getConfigurationPath()
                        + "/models/opennlp-es-pos-maxent-pos-es.model").getPath());
            } catch (Exception ex) {
```

```
            ex.printStackTrace();
        }
        posModel = new POSModel(modelIn);
        if (modelIn != null) {
            try {
                modelIn.close();
            } catch (IOException e) {
            }
        }
        try {
            modelIn = new FileInputStream(
                new URI(Propiedades.getConfigurationPath()
                    + "/models/en-token.bin").getPath());
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        tokModel = new TokenizerModel(modelIn);
        if (modelIn != null) {
            try {
                modelIn.close();
            } catch (IOException e) {
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (modelIn != null) {
            try {
                modelIn.close();
            } catch (IOException e) {
            }
        }
    }
}

/**
 * The OpenNLP Tokenizers segment an input character sequence into tokens.
 * Tokens are usually words, punctuation, numbers, etc.
 *
 * @param texto
 * @return
 */
@Override
public ArrayList<String> wordTokenizer(String texto) {
    TokenizerME tokenizer = getTokenizer();
    return new ArrayList<String>(Arrays.asList(tokenizer.tokenize(texto)));
}

public TokenizerME getTokenizer() {
    return new TokenizerME(tokModel);
```

```
}

/**
 * The OpenNLP Sentence Detector can detect that a punctuation character
 * marks the end of a sentence or not. In this sense a sentence is defined
 * as the longest white space trimmed character sequence between two
 * punctuation marks. The first and last sentence make an exception to this
 * rule. The first non whitespace character is assumed to be the begin of a
 * sentence, and the last non whitespace character is assumed to be a
 * sentence end.
 *
 * @see opennlp.tools.sentdetect.SentenceDetectorME
 * @param texto
 * @return
 */
@Override
public ArrayList<String> sentenceTokenizer(String texto) {
    SentenceDetectorME sentenceDetector = getSentenceDetector();
    return new ArrayList<String>(Arrays.asList(sentenceDetector.sentDetect(texto)));
}

public SentenceDetectorME getSentenceDetector() {
    return new SentenceDetectorME(senModel);
}

/**
 * The Part of Speech Tagger marks tokens with their corresponding word type
 * based on the token itself and the context of the token. A token might
 * have multiple pos tags depending on the token and the context. The
 * OpenNLP POS Tagger uses a probability model to predict the correct pos
 * tag out of the tag set. To limit the possible tags for a token a tag
 * dictionary can be used which increases the tagging and runtime
 * performance of the tagger.
 *
 * @see http://nlp.lsi.upc.edu/freeling/doc/tagsets/tagset-es.html
 * @param sentences
 * @return
 */
@Override
public ArrayList<String> posTag(String sentences[]) {
    POSTaggerME tagger = getPOSTagger();
    return new ArrayList<String>(Arrays.asList(tagger.tag(sentences)));
}

public POSTaggerME getPOSTagger() {
    return new POSTaggerME(posModel);
}

@Override
public String stem(String palabra) {
```

```
        System.out.println("Method public String stem(String palabra)."
            + "Not supported on OpenNLP use SnowballStemDocodeNLPImpl.");
        return new SnowballStemDocodeNLPImpl().stem(palabra);
    }
}
```