Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Documento de Diseño

Sistema de detección de plagio en archivos de texto

Creado por

Roberto Iván Morán Torres

Matrícula: 208205737

Asesora

Dra. Ma. Lizbeth Gallardo López

Departamento de Sistemas

Trimestre 13-I

Enero 2013

Índice

1. Introducción

En este documento se describe la especificación del *Sistema de detección de plagio en archivos de texto* mediante los requerimientos funcionales y no funcionales. Además se describe brevemente cada uno de los cuatro módulos que componen el sistema presentando los artefactos de análisis: diagramas de caso de uso generales y específicos, escenarios de uso, diagrama de clases y análisis de robustez.

2. Especificación del sistema

El *Sistema de detección de plagio en archivos de texto* cumple con los requisitos indicados a continuación:

2.1. Requisitos funcionales

Selección de archivos

El sistema permite al usuario seleccionar archivos de texto a comparar con la condición de que deben ser archivos con extensión .docx, .odt o .txt.

Detección de similitud en estructura

El sistema compara la estructura de los dos archivos que recibió de entrada en base al número de palabras y párrafos de cada archivo. De esta forma si el grado de similitud es alto entonces se procederá a realizar el procesamiento del texto en los archivos; de lo contrario, no se procesarán los archivos retornando como resultado que la similitud entre ellos es muy baja o nula.

Procesamiento de texto

El sistema procesa cada uno de los archivos de texto que se recibió como entrada con técnicas de procesamiento de lenguaje natural listadas a continuación:

1. *Minusculización:* Con esta técnica todo el texto se pasará a minúsculas.
2. *Remplazo de números:* Con esta técnica todos los datos numéricos se remplazarán por un identificador, "num".
3. *Limpiado:* Con esta técnica se removerán caracteres no deseados, en este caso solo se manejarán caracteres alfabéticos eliminando puntos, comas, etc. Además se omitirán los acentos, es decir se remplazaran los caracteres á, é, í, ó y ú por a, e, i, o y u respectivamente.
4. *Lematización:* Con esta técnica utilizando WordNet 3.0 se transformarán las palabras en su forma base, es decir, dada una forma en plural, en femenino, conjugada, etcétera, hallar el lema correspondiente.
5. *Remplazado de sinónimo:* Con esta técnica utilizando WordNet 3.0 las palabras se remplazarán por sus respectivos sinónimos con mayor frecuencia de uso.

Detección de similitud en el texto

El sistema compara el texto de los dos archivos que recibió de entrada generando tres medidas distintas de similitud basadas en los trigramas en común, subcadenas comunes más largas y las huellas digitales de cada par de archivos de texto.

En el caso de la medida de similitud basada en huella digital el usuario deberá proporcionar los dos parámetros numéricos que definen el tamaño de ventana; en el caso de la medida de similitud basada subcadenas comunes más largas el usuario deberá proporcionar el parámetro numérico que definen el número mínimo de palabras de cada subcadena. De lo contrario el sistema asignara los valores determinados por default.

Presentación de resultados

El sistema además de mostrar al usuario las medidas de similitud por trigramas, huella digital y subcadenas comunes más largas, mostrará el texto en común de cada par de los archivos comparados, a través de un marcado en color en ambos textos. Esto se realizará en base a los trigramas en común de cada archivo o a las subcadenas comunes más largas según lo decida el usuario.

2.2. Requisitos no funcionales

Facilidad de uso

Una vez que el usuario obtenga acceso al sistema podrá observar todas las herramientas disponibles.

El sistema contará con fuentes y colores adecuados para facilitar la lectura del texto en el sistema.

Fiabilidad

Cuando el sistema no pueda abrir los archivos de entrada, mostrará el siguiente mensaje: "No se pudo abrir el archivo".

Cuando el sistema no pueda encontrar palabras para la lematización y/o remplazado de sinónimos en la base de datos WordNet 3.0 para cierta palabra en los archivos de texto, no se modificara la palabra y se continuará el procesamiento de los archivos.

Soporte

Las técnicas PLN del sistema están enfocadas solo documentos en idioma español.

Implementación

El sistema se desarrolla como un sistema de escritorio.

Hardware

HP pavilion dv4-2145dx entertainment notebook pc con procesador AMD Turion(tm) II Dual-Core Mobile M520 2.30GHz, memoria Ram de 4.00 gigabytes y disco duro de 320 gigabytes.

Software

NetBeans: Entorno de desarrollo integrado de código abierto.

Lenguaje de programación Java.

MySQL: Manejador de bases de datos de código abierto.

3. Módulos del sistema

Los cuatro módulos que conforman el sistema son: módulo 1, encargado de la detección de similitud estructural a través del número de palabras y párrafos de dos archivos de texto; módulo 2, encargado de procesar cada uno de los archivos de texto que se recibirán como entrada con técnicas de procesamiento de lenguaje natural; módulo 3, encargado de la detección de similitud del texto en base a la comparación de trigramas, huellas digitales y subcadenas comunes más largas; y módulo 4, encargado de presentar las funcionalidades del sistema, recuperar los datos de entrada y devolver los resultados al usuario.

3.1. Diagrama de casos de uso general

En la figura 1, se muestran el diagrama general de casos de uso del sistema en el cual podemos observar los cuatro módulos que lo conforman.

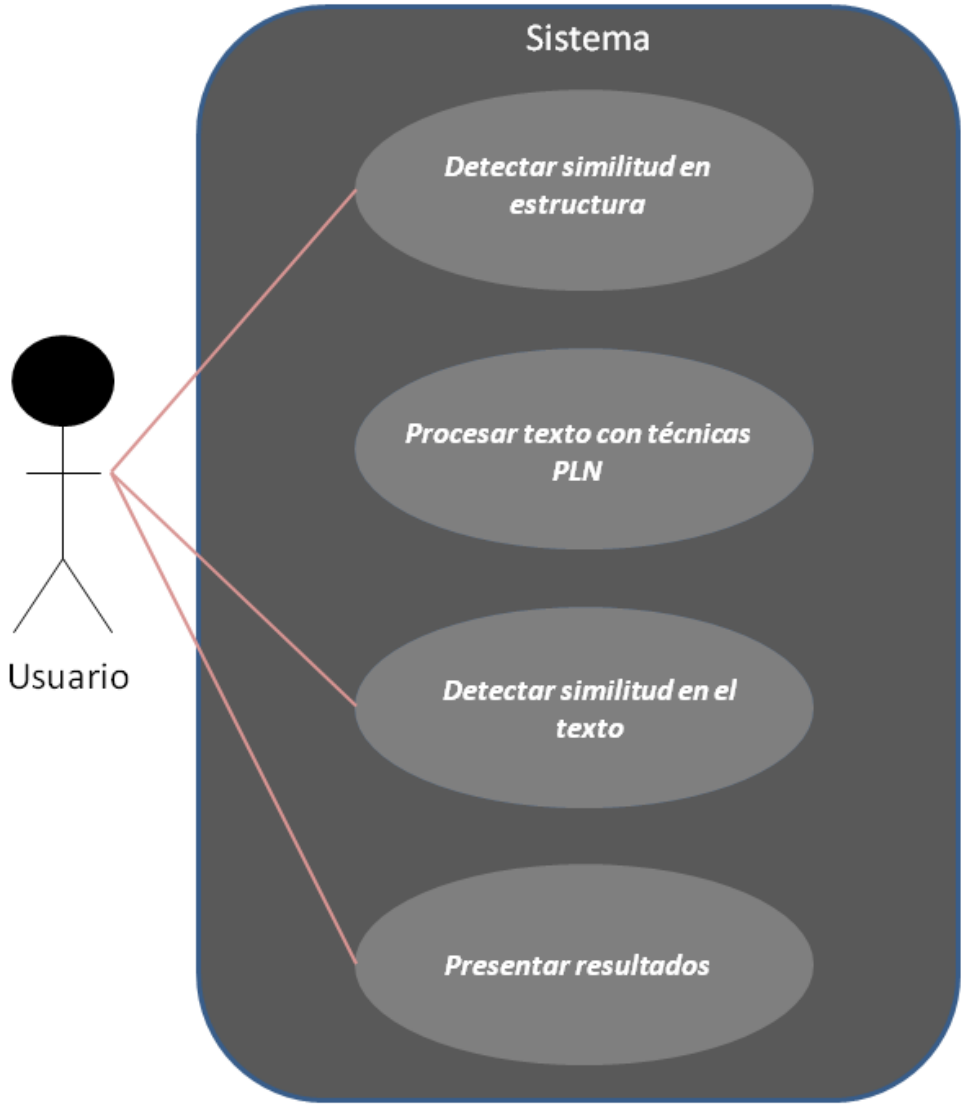

Figura 1: Casos de uso general

3.2. Escenarios de uso

3.2.1. Caso de uso: Detectar similitud en estructura

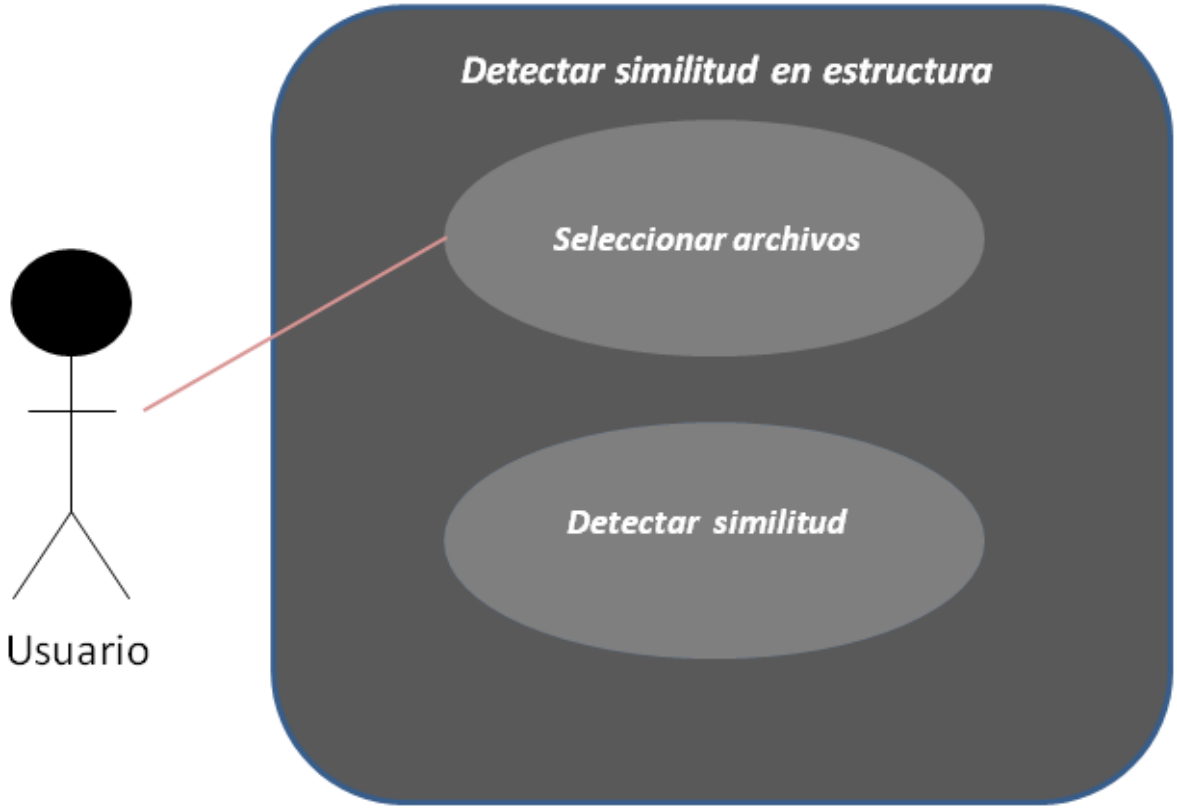

Figura 2: Caso de uso Detectar similitud en estructura

Seleccionar archivos: Se deben seleccionar los archivos de texto a comparar los cuales deben tener la extensión .doc, .docx, .odt o .txt. Además se seleccionaran los parámetros para los algoritmos de detección de similitud en huella digital y subcadenas comunes más largas.

Detectar similitud: Se compara la estructura de todos los posibles pares de archivos seleccionados en base al número de palabras y párrafos para determinar si se procesan o no los archivos.

3.2.2. DSE-1: Seleccionar Archivos

Descripción: Caso de uso que permite seleccionar archivos de texto con extensión .doc, .docx, .odt o .txt, para ser comparados y la selección de parámetros para los algoritmos de detección de similitud en huella digital y subcadenas comunes más largas.

Actores:

- Primario: Usuario
- Secundario: Ninguno

Disparador: El actor primario solicita al sistema seleccionar los archivos a comparar.

Precondiciones: Ninguna

Poscondiciones: Los archivos quedan seleccionados para proceder a la detección de similitud en estructura para cada par posible de archivos.

Flujo Principal:

1. El actor primario oprime el botón “Seleccionar archivos”.
2. El sistema muestra una ventana donde puede seleccionar varios archivos a la vez.
3. El actor primario selecciona los archivos a comparar.
4. El sistema muestra la cantidad de archivos seleccionados.
5. El actor primario selecciona los parámetros para para los algoritmos de detección de similitud en huella digital y subcadenas comunes más largas.
6. El actor primario oprime el botón “Analizar archivos”.

Flujos alternativos

a. Uno o más archivos seleccionados no son de extensión .doc, .docx, .odt o .txt.
   1. El sistema muestra el mensaje “Uno o más archivos no son de extensión .doc, .docx, .odt o .txt”.
   2. El usuario selecciona archivos con una extensión soportada por el sistema.
   3. El flujo continúa en el punto 5.

b. El usuario no selecciona los parámetros para para los algoritmos de detección de similitud en huella digital y subcadenas comunes más largas.
   1. El sistema toma los valores definidos por default.
   2. El flujo continúa en el punto 6.

Requerimientos no funcionales

El tiempo de demora del para el proceso de cargar archivo depende del usuario.

3.2.3. DSE-2: Detectar similitud

Descripción: Caso de uso que permite comparar la estructura de dos archivos de texto, ver pseudocódigo 1.

Actores:

- Primario: Usuario
- Secundario: ninguno

Disparador: El actor primario solicita al sistema analizar los archivos seleccionados.

Precondiciones: El actor primario ha seleccionado al menos dos archivos.

Poscondiciones: El sistema genera una medida estructural de similitud de cada par de archivos y prosigue su procesamiento.

Flujo Principal:

1. El actor primario oprime el botón “Analizar archivos”.
2. Para cada par de archivos:
   2.1. El sistema realiza un conteo de las palabras y párrafos de del archivo 1.
   2.2. El sistema realiza un conteo de las palabras y párrafos de del archivo 2.
   2.3. El sistema genera la medida de similitud de los archivos en base al número de palabras y párrafos de los archivos.
   2.4. El sistema verifica que la medida de similitud estructural es mayor o igual a 0.5.
   2.5. El sistema continúa a procesar el texto de los archivos con técnicas PLN.

Flujos alternativos

a. El archivo 1, 2 o ambos no pueden ser leídos por el sistema.
   1. El sistema muestra el mensaje “El archivo no se pudo abrir”.
   2. El flujo continúa en el punto 1 del flujo principal de DSE-1: Seleccionar archivos.

b. La medida de similitud estructural es menor a 0.5
   1. No se continúa con el procesamiento de los archivos.

Requerimientos no funcionales
El tiempo de demora del para obtener la medida de similitud estructural depende del tamaño de los archivos.

EL caso de uso *Detectar similitud en estructura* representa el módulo 1 del sistema. El módulo 1 interactúa con el usuario para la selección de archivos y para la definición de los parámetros de detección de similitud. Posteriormente este módulo se encarga de la comparación de la estructura de los pares de archivos seleccionados, lo cual se realiza de la siguiente manera:

Se cuenta el número de párrafos de los documentos d_1 y d_2 siendo *PA1* y *PA2* respectivamente; también se cuenta el número de palabras de los documentos d_1 y d_2 siendo *WA1* y *WA2* respectivamente de los cuales se obtiene la diferencia absoluta del número de párrafos (*DAP*) y la diferencia absoluta del número de palabras (*DAW*) de los archivos de texto de la siguiente forma:

$$DAP = |PA1 - PA2| \qquad DAW = |WA1 - WA2|$$

Una vez obtenidos el *DAP* y el *DAW* se procede a obtener una medida de similitud del número de párrafos (*SP*) y una medida de similitud del número de palabras (*SW*) de los archivos de la siguiente forma:

$$SP = 1 - \frac{DAP}{MAX\{PA1, PA2\}} \qquad SW = 1 - \frac{DAW}{MAX\{WA1, WA2\}}$$

$$Donde: \quad 0 \leq SP, SW \leq 1$$
$$Siendo\ 0\ el\ menor\ y\ 1\ el\ mayor\ grado\ de\ similitud$$

Ya con las medidas *SP* y *SW* se procede a obtener la medida de similitud estructural (*SimilitudE*) de los archivos de texto, de la siguiente forma:

$$SimilitudE = 0.2SP + 0.8SW$$
$$Donde: \quad 0 \leq SimilitudE \leq 1$$
$$Siendo\ 0\ el\ menor\ y\ 1\ el\ mayor\ grado\ de\ similitud$$

El pseudocódigo 1 muestra como se realiza la comparación de la estructura de dos archivos de texto.

```
Función Similitud(Archivos A1, Archivos A2)
    SP ← abs(A1.getNumParrafos()-A2.getNumParrafos())//Se obtiene |PA1-PA2|
    SP ← 1-(SP/Max{A1.getNumParrafos(), A2.getNumParrafos()})//Se calcula SP
    SW ← abs(A1.getNumPalabras()-A2.getNumPalabras())//Se obtiene |WA1-WA2|
    SW ← 1-(SW/Max{A1.getNumPalabras(), A2.getNumPalabras()})//Se calcula SW
    SimilitudE ← (0.2*SP) + (0.8*SW) //Se calculala medida de similitud
    Regresa SimilitudE
Fin
```

Pseudocódigo 1: SimilitudEstructura

3.2.4. Caso de uso: Procesar texto con técnicas PLN

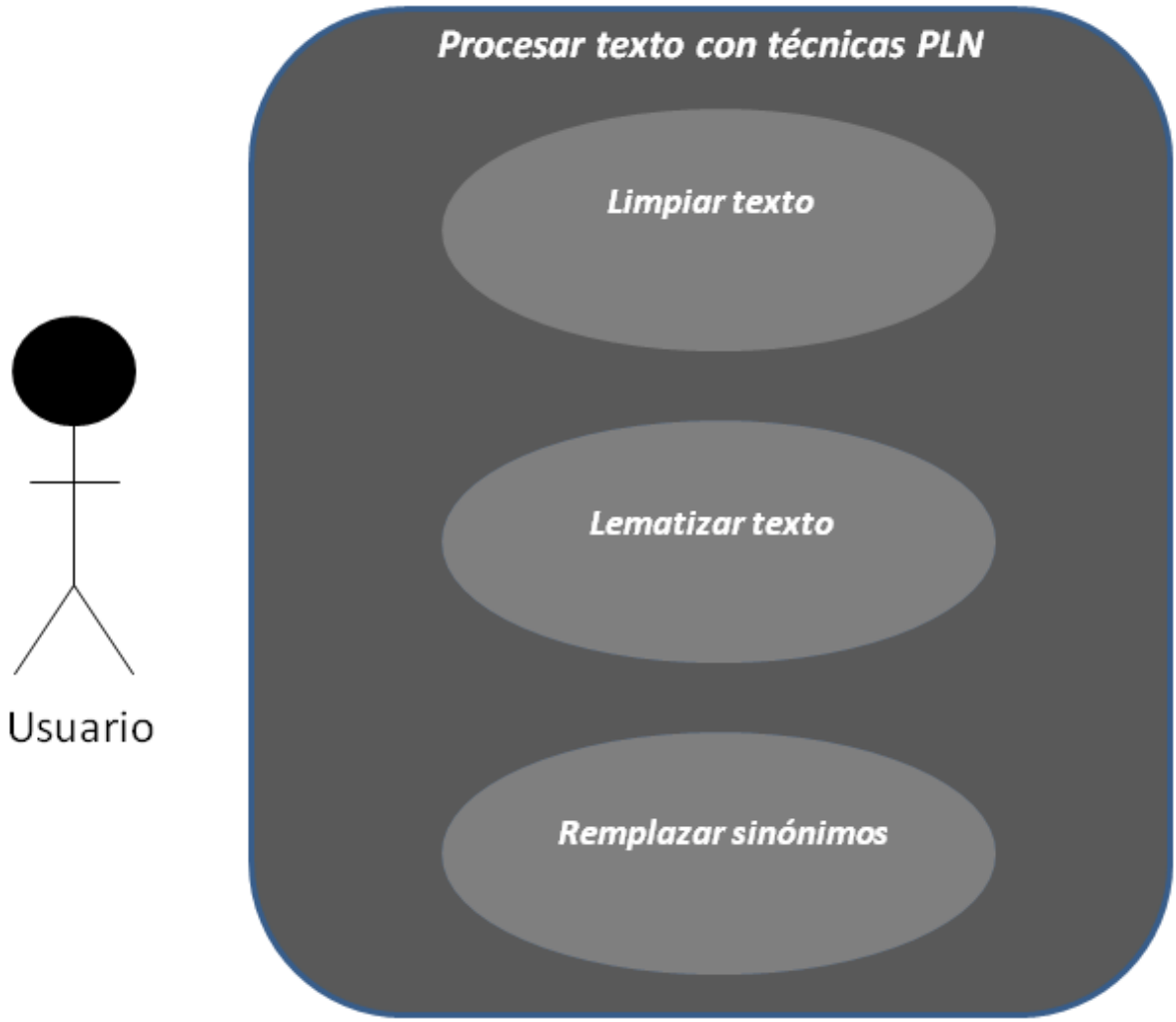

Figura 3: Caso de uso Procesar texto con técnicas PLN

Limpiar texto: El sistema aplica al texto de cada par de archivos a comparar las técnicas PLN de minusculización, remplazo de números y limpiado.

Lematizar texto: El sistema aplica al texto de cada par de archivos a comparar la técnica PLN de lematización.

Remplazar sinónimos: El sistema aplica al texto de cada par de archivos a comparar la técnica PLN[1] de remplazado de sinónimos.

[1] PLN (Procesamiento de lenguaje natural), que se define como una serie de técnicas enfocadas a que las máquinas sean capaces de manejar lenguajes no formales.

3.2.5. PTP-1: Limpiar texto

Descripción: Caso de uso que permite procesar cada par de archivos a comparar con las técnicas PLN: minusculización, remplazo de números y limpiado. Ver pseudocódigo 2.

Actores:

- Primario: Ninguno
- Secundario: Usuario

Disparador: Los archivos pasan la prueba de similitud estructural.

Precondiciones: La similitud estructural de los archivos es mayor o igual 0.5.

Poscondiciones: El texto de los archivos se ha modificado para su procesamiento de lematización.

Flujo Principal:

1. El sistema lee el archivo 1.
2. El sistema aplica la técnica de minusculización al texto del archivo 1.
3. El sistema aplica la técnica de remplazo de números al texto del archivo 1.
4. El sistema aplica la técnica de limpiado al texto del archivo 1.
5. El sistema lee el archivo 2.
6. El sistema aplica la técnica de minusculización al texto del archivo 2.
7. El sistema aplica la técnica de remplazo de números al texto del archivo 2.
8. El sistema aplica la técnica de limpiado al texto del archivo 2.
9. El sistema continúa con el procesamiento de los archivos aplicando la técnica de lematización.

Flujos alternativos

a. El archivo 1, 2 o ambos no pueden ser leídos por el sistema.
   1. El sistema muestra el mensaje “El archivo no se pudo abrir”.
   2. El flujo continúa en el punto 1 del flujo principal de DSE-1: Seleccionar archivos.

Requerimientos no funcionales

El avance del procesamiento de los archivos es visible para el usuario.

El pseudocódigo 2 muestra como se procesan los archivos con la técnica de *limpiar texto* la cual consiste en: 1) pasar todo el texto de los archivos a minúsculas, 2) remover todos los caracteres no alfabéticos y 3) remplazar todos los números con la etiqueta “num”; para generalizar la comparación, 4) remplazar las vocales acentuadas por vocales simples.

```
Función procesaArchivo (Archivos A1)
    Para i ← 0 hasta i<A1.getTokens.size()
        A1.setTokens(i, tokens.get(i).toLowerCase()) //texto en minusculas
        //Se eliminan acentos
        A1.setTokens(i, tokens.get(i).replace('á', 'a'))
        A1.setTokens(i, tokens.get(i).replace('é', 'e'))
        A1.setTokens (i, tokens.get(i).replace('í', 'i'))
        A1.setTokens (i, tokens.get(i).replace('ó', 'o'))
        A1.setTokens (i, tokens.get(i).replace('ú', 'u'))
        //Se remplazan datos numéricos con la etiqueta “num”
        Si A1.getTokens(i) es numero
            A1.setTokens(i,”num”)
        //Se eliminan cadenas no alfabéticas
        Otro Si A1.getTokens(i) no es alfabético
            A1. getTokens.remove(i)
            i ← i-1
        Fin Si
    Fin Para
Fin
```

Pseudocódigo 2: procesaArchivo

3.2.6. PTP -2: Lematizar texto

Descripción: Caso de uso que permite procesar cada par de archivos a comparar con la técnica PLN: lematización. Ver pseudocódigo 3.

Actores:

- Primario: Ninguno
- Secundario: Usuario

Disparador: Los archivos pasan la prueba de similitud estructural.

Precondiciones: Los archivos fueron procesados con las técnicas PLN: minusculización, remplazo de números y limpiado.

Poscondiciones: El texto de los archivos se ha modificado para procesamiento de remplazado de sinónimos.

Flujo Principal:

1. El sistema se conecta a la base de datos que contiene la base de palabras WordNet 3.0.
2. El sistema lee el archivo 1.
3. El sistema aplica la técnica de lematización al texto del archivo 1.
4. El sistema lee el archivo 2.
5. El sistema aplica la técnica de lematización al texto del archivo 2.
6. El sistema continúa con el procesamiento de los archivos aplicando la técnica de remplazado de sinónimos.

Flujos alternativos

a. El sistema no se pudo conectar a la base de palabras WordNet 3.0.

1. El sistema muestra el mensaje “Conexión a WordNet 3.0 Fallida”.
2. El flujo continúa en el punto 6 del flujo principal.

b. El archivo 1, 2 o ambos no pueden ser leídos por el sistema.

1. El sistema muestra el mensaje “El archivo no se pudo abrir”.
2. El flujo continúa en el punto 1 del flujo principal de DSE-1: Seleccionar archivos.

Requerimientos no funcionales

El avance del procesamiento de los archivos es visible para el usuario.

El pseudocódigo 3 muestra como se lleva a cabo la aplicación de la técnica de lematización, la cual consiste en remplazar cada palabra del documento con su respectivo lema, en caso de que éste exista. Los lemas se obtienen mediante el método *consultaLema*(), el cual accede a la base de datos WordNet 3.0[2] para extraer los lemas existentes de cada palabra.

```
Función Lematización (Archivos A1)
    Para i ← 0 hasta i<A1.getTokens.size()
        Lema ← consultaLema(A1.getTokens(i)) //Consulta WordNet 3.0
        Si Lema es distinto de NULL //Si se encontró lema
            A1.setTokens(i, Lema)
        Fin Si
    Fin Para
Fin
```

Pseudocódigo 3: Lematización

[2] WordNet es una gran base de datos léxica de sustantivos, verbos, adjetivos y adverbios agrupadas en conjuntos de sinónimos cognitivos y lemas.

3.2.7. PTP -3: Remplazar sinónimos

Descripción: Caso de uso que permite procesar cada par de archivos a comparar con la técnica PLN: remplazado de sinónimos. Ver pseudocódigo 4.

Actores:

- Primario: Ninguno
- Secundario: Usuario

Disparador: Los archivos pasan la prueba de similitud estructural.

Precondiciones: Los archivos fueron procesados con la técnica PLN: lematización.

Poscondiciones: Los archivos se han modificado para su procesamiento de detección de similitud en el texto.

Flujo Principal:

1. El sistema se conecta a la base de datos que contiene la base de palabras WordNet 3.0.
2. El sistema lee el archivo 1.
3. El sistema aplica la técnica de remplazado de sinónimos al texto del archivo 1.
4. El sistema lee el archivo 2.
5. El sistema aplica la técnica de remplazado de sinónimos al texto del archivo 2.
6. El sistema continúa con el procesamiento de detección similitud en el texto.

Flujos alternativos

a. El sistema no se pudo conectar a la base de palabras WordNet 3.0.

1. El sistema muestra el mensaje “Conexión a WordNet 3.0 Fallida”.
2. El flujo continúa en el punto 6 del flujo principal.

b. El archivo 1, 2 o ambos no pueden ser leídos por el sistema.

1. El sistema muestra el mensaje “El archivo no se pudo abrir”.
2. El flujo continúa en el punto 1 del flujo principal de DSE-1: Seleccionar archivos.

Requerimientos no funcionales

El avance del procesamiento de los archivos es visible para el usuario.

El pseudocódigo 4 muestra como se lleva a cabo la aplicación de la técnica de remplazo de sinónimos, la cual consiste en remplazar cada palabra del documento con su respectivo sinónimo, en caso de que éste exista. Los sinónimos se extraen mediante el método *consultaSinonimo*(), el cual accede a la base de datos WordNet 3.0 para extraer los sinónimos existentes de cada palabra.

```
Función Sinónimos (Archivos A1)
        Para i ← 0 hasta i<A1.getTokens.size()
                //Consulta WordNet 3.0
                Sinónimo ← consultaSinonimo(A1.getTokens (i))
                Si Sinónimo es distinto de NULL //Si se encontró sinónimo
                        A1.setTokens(i, Sinónimo)
                Fin Si
        Fin Para
Fin
```

Pseudocódigo 4: Sinónimos

3.2.8. Caso de uso: Detectar similitud en el texto

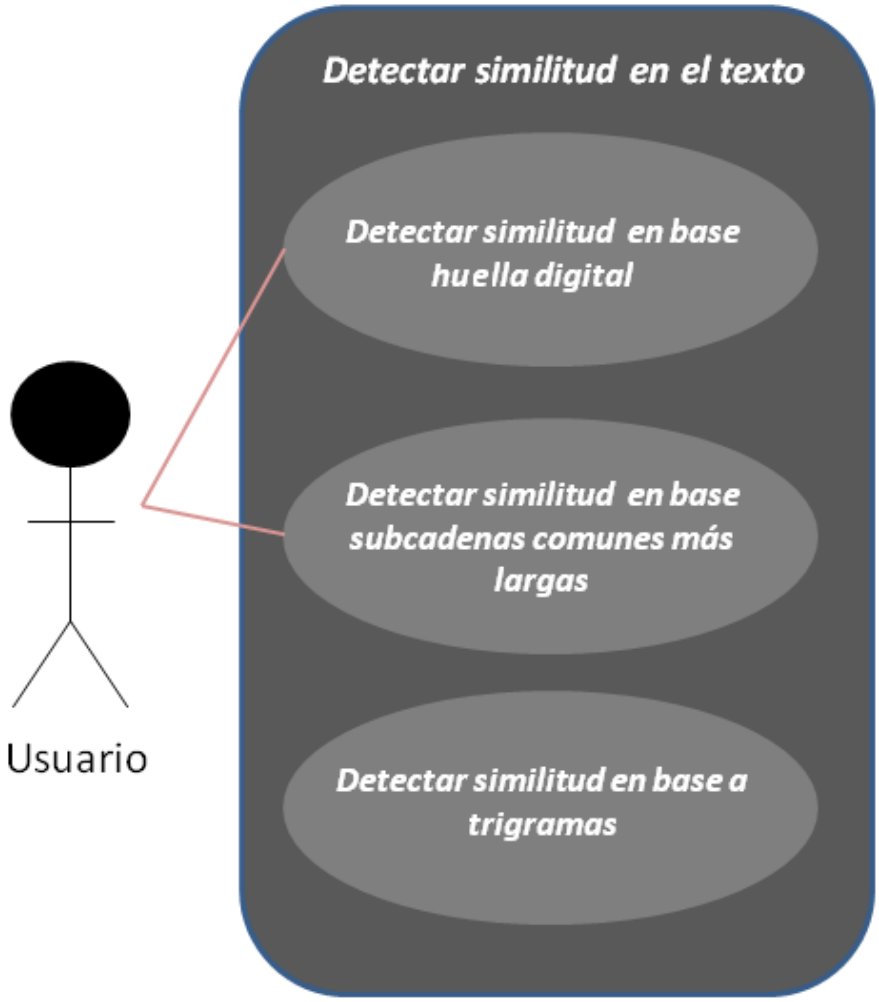

Figura 4: Caso de uso Detectar similitud en el texto

Detectar similitud en base a huella digital: El sistema obtendrá las huellas digitales de cada par de archivos y las comparara obteniendo una medida de similitud.

Detectar similitud en base a subcadenas comunes más largas: El sistema obtendrá las subcadenas comunes más largas de cada par de archivos obteniendo una medida de similitud.

Detectar similitud en base a trigramas: El sistema compara los trigramas de cada par de archivos obteniendo una medida de similitud.

3.2.9. DST-1: Detectar similitud en base a huella digital

Descripción: Caso de uso que permite detectar la similitud en el texto para cada par de archivos en base a su huella digital. Ver pseudocódigos 5 y 6.

Actores:

- Primario: Ninguno
- Secundario: Usuario

Disparador: Los archivos pasan la prueba de similitud estructural.

Precondiciones: Los archivos fueron procesados con las técnicas PLN.

Poscondiciones: El sistema retorna una medida de similitud de ambos archivos.

Flujo Principal:

1. El sistema obtiene la huella digital del archivo 1.
2. El sistema obtiene la huella digital del archivo 2.
3. El sistema obtiene una medida de similitud compara las huellas digitales de ambos archivo.
4. El sistema continúa con la detección de similitud en base a subcadenas comunes más largas.

Requerimientos no funcionales

El avance del procesamiento de los archivos es visible para el usuario.

Algoritmo: Detección de similitud en base a huella digital

El cálculo de similitud de dos archivos en base a huella digital se basa en el algoritmo *Winnowing para hashings de k-gramas*, el cual para formar las huellas digitales toma una secuencia de valores *hash* del documento, a la secuencia de valores se le llama *ventana*. Después de generar todas las ventanas posibles de un documento, se selecciona al *hash* mínimo de cada una de esas ventanas. Los valores *hash* seleccionados conformarán la huella digital del documento. El usuario necesitará proporcionar al algoritmo los valores numéricos *t* y *k*. Si hay una cadena repetida de al menos *t* caracteres, el algoritmo garantiza que, de existir esta misma cadena en el otro documento, será detectada; el algoritmo no garantiza detectar coincidencias en cadenas menores a *k caracteres*.

Después de aplicar el algoritmo HD, la huella digital de un documento queda representada por un conjunto de valores *hash*. Sea d_1 y los documentos a comparar, y sean $W(d_1)$ y $W(d_2)$ las hullas digitales correspondientes; la medida de similitud (*similitudHD*) entre ambas huellas se define como sigue:

$$similitudHD = \frac{|W(d_1) \cap W(d_2)|}{|W(d_1) \cup W(d_2)|} \qquad Donde: 0 \leq similitudHD \leq 1$$

En el pseudocódigo 6 se muestra el algoritmo para obtener la similitud de huella digital de dos documentos.

```
Función comparaHD(Archivos A1, Archivos A2,int t,int k)
        HD1←HD(A1, t, k)//Se genera huella digital para el archivo 1
        HD2←HD(A2, t, k) //Se genera huella digital para el archivo 2
        tamaño ← HD1.size()+HD2.size()
        Para i ← 0 hasta i<HD1.size()//Se verifican las coincidencias
                Para j ←0 hasta j<HD2.size()
                        Si HD1.get(i)-HD2.get(j) es cero
                                HD.add(HD1.get(i))
                                HD.add(HD2.get(j))
                                HD2.remove(j)
                                HD1.remove(i)
                                j ← HD2.size();
                                i ← i-1
                        Fin Si
                Fin Para
        Fin Para
        //Se calcula la similitud de los archivos
        similitudHD=(tamaño-(HD1.size()+HD2.size()))/tamaño;
Fin
```

Pseudocódigo 6: ComparaHD

3.2.10. DST -2: Detectar similitud en base a subcadenas comunes más largas

Descripción: Caso de uso que permite detectar la similitud en el texto para cada par de archivos en base a subcadenas comunes más largas. Ver pseudocódigo 7.

Actores:

- Primario: Ninguno
- Secundario: Usuario

Disparador: Los archivos pasan la prueba de similitud estructural.

Precondiciones: Los archivos fueron procesados con las técnicas PLN.

Poscondiciones: El sistema retorna una medida de similitud de ambos archivos.

Flujo Principal:

1. El sistema obtiene las subcadenas comunes en ambos archivos.
2. El sistema obtiene una medida de similitud en base a las subcadenas comunes encontradas.
3. El sistema continúa con la detección de similitud en base a trigramas.

Requerimientos no funcionales

El avance del procesamiento de los archivos es visible para el usuario.

Algoritmo: Detección de similitud en base a subcadenas comunes más largas

La detección de similitud en cadenas comunes más largas se basa en el algoritmo *Greedy-String-Tiling*, el cual para localizar las cadenas comunes más largas en dos documentos, utiliza el concepto de *match*, que es una asociación temporal de un par de cadenas comunes en los documentos d_1 y d_2. Esta asociación no necesariamente es única ya que una de las dos cadenas podría estar involucrada en más de un *match*. También usa el concepto de *tile*, que es un conjunto de *matches* únicos. Un *match* es único, si la longitud de las cadenas que forman el match es máxima. El desarrollo del algoritmo se descompone en dos etapas:

> Etapa 1: Se itera en los textos buscando los *matches* de mayor longitud no menor a *M*. Donde *M* es una constante proporcionada por el usuario que define el número de palabras mínimas de cada *match.* Un valor mínimo *M* debe ser mayor o igual a 3 palabras para considerarse un buen parámetro para obtener las subcadenas comunes más largas.

Etapa 2: Se marcan todos los *matches* de longitud máxima encontrados en la etapa 1 formando el conjunto *tile*, es decir se marcan las palabras del texto que conforman al *match*, para evitar su uso en *matches* posteriores.

Estas dos etapas se repiten hasta no encontrar más *matches* de longitud mayores a *M*. Los *matches* se conformarán por una tripleta *match(a, b, l)* donde *a* denota la posición de comienzo de la subcadena en el texto 1, *b* denota la posición de comienzo de la subcadena en el texto 2 y *l* denota la longitud de la cadena. De la misma manera el conjunto *tile* se conformará por tripletas *match (a, b, l).*

Una vez obtenido el conjunto *tile*, se puede obtener una medida de similitud *similitudSCL* en base a la sumatoria de la longitud de los *matches* que lo conforman, la cantidad de palabras en el texto *T1* y en el texto *T2* como sigue:

$$similitudSCL = \frac{2\sum_{matc\ h(a,b,l)\epsilon\ tiles}\ l}{T1 + T2} \qquad Donde: 0 \leq similitudSCL \leq 1$$

$$Siendo\ 0\ el\ menor\ y\ 1\ el\ máximo\ grado\ de\ similitud$$

En el pseudocódigo 7 se muestra el algoritmo para obtener la similitud en cadenas comunes de dos documentos

```
Función comparaSCL(Archivo A, Archivo B,int M)
        tiles ← {}//Arreglo contenedor de las cadenas comunes
        Haz
            maxmatch ← M//Tamaño mínimo de cadena
            matches ← {}////Arreglo contenedor de las cadenas comunes
            Para todos los tokens sin marcar Aa en A //Se itera sobre el texto en el archivo A
                Para todos los tokens sin marcar Bb in B //Se itera sobre el texto en el archivo B
                    j ← 0
                    //Se detectan los matches
                    Mientras (Aa+j == Bb+j && SinMarcar(Aa+j) && SinMarcar (Bb+j))
                          j ← j+1
                           Si j == maxmatch //Se valida tamaño mínimo de match
                                matches ← matches ⊕ match(a, b, j) //se agregan matches encontrados
                           Otro Si j >maxmatch //Se encontró match de longitud mayor a tamaño mínimo
                                matches ← {match(a, b, j)} //se agrega match eliminanndo los anteriores
                                maxmatch ← j //Se define nuevo tamaño mínimo de match
                           Fin Si
                    Fin Mientras
                Fin Para
                //Se marca el texto de los matches de máxima longitud
                Para todos los match(a, b, maxmatch) ∈ matches
                    Para j ←0 hasta maxmatch− 1
                         marca(Aa+j)
                        marca(Bb+j )
                    Fin Para
                    //Se agregan los matches encontrados al conjunto tile
                    tiles ←tiles ∪ match(a, b, maxmatch)
                Fin Para
          Mientras (maxmatch > M)//Se itera hasta no encontrar matches de longitud mayor a M
          Regresa tiles
Fin
```

Pseudocódigo 7: comparaSCL

3.2.11. DST -3: Detectar similitud en base a trigramas

Descripción: Caso de uso que permite detectar la similitud en el texto para cada par de archivos en base a la comparación de sus trigramas. Ver pseudocódigo 8.

Actores:

- Primario: Ninguno
- Secundario: Usuario

Disparador: Los archivos pasan la prueba de similitud estructural.

Precondiciones: Los archivos fueron procesados con las técnicas PLN.

Poscondiciones: El sistema retorna una medida de similitud de ambos archivos.

Flujo Principal:

1. El sistema obtiene los trigramas únicos del archivo 1.
2. El sistema obtiene los trigramas únicos del archivo 2.
3. El sistema obtiene una medida de similitud en base los trigramas en común de ambos archivos.
4. El sistema continúa con la presentación de los resultados obtenidos al usuario.

Requerimientos no funcionales

El avance del procesamiento de los archivos es visible para el usuario.

Algoritmo: Detección de similitud en base a trigramas

La detección de similitud en trigramas, consiste en formar todos los trigramas distintos de cada documento, detectando las coincidencias de los trigramas. Una vez formados los trigramas distintos para los documentos d_1 y d_2 que se representarán como el conjunto A y B respectivamente, se comparan para obtener una medida de similitud (*similitudT*) de la siguiente forma:

$$similitudT = \frac{Número\ Trigramas\ en\ comun}{Número\ total\ de\ Trigramas} = \frac{A \cap B}{A \cup B} \qquad Donde: 0 \leq similitudT \leq 1$$

Pseudocódigo 8 muestra el algoritmo para comparar dos archivos de texto en base a sus trigramas.

```
Función comparaT(Archivos A1, Archivos A2)
    aux[0] ← A1.getTokens(0)//Primer palabra del trigrama
    aux[1] ← A1.getTokens(1)//Segunda palabra del trigrama
    aux[2] ← A1.getTokens(2)//Tercera palabra del trigrama
    aux[3] ← "0"//Posición del trigrama en el archivo 1, i= inexistente
    aux[4] ← "i"//Posición del trigrama en el archivo 2, i= inexistente
    trigrama.add(aux)
    //Se generan todos los trigramas distintos del archivo 1
    Para i ← 1 hasta i < A1.getToken().size() – 2
        aux[0] ← A1.getTokens(i)
        aux[1] ← A1.getTokens(i + 1)
        aux[2] ← A1.getTokens(i + 2)
        aux[3] ← "" + i
        aux[4] ← "i"
        index ← binarySearch(trigrama, aux)//Se verifica existencia de trigrama
        Si index >= 0//Si existe se agrega la posición de nueva aparición
            aux[0] ← trigrama.get(index)[0]
            aux[1] ← trigrama.get(index)[1]
            aux[2] ← trigrama.get(index)[2]
            aux[3] ← trigrama.get(index)[3] + ":" + i
            aux[4] ← trigrama.get(index)[4]
            trigrama.set(index, aux)
        Otro //De lo contario se agrega el nuevo trigrama
            trigrama.add(aux)
            Sort(trigrama)//Se ordenan alfabéticamente los trigramas
        Fin Si
    Fin Para
    //Se generan todos los trigramas distintos del archivo 2
    Para  i ← 0 hasta i < A2.getTokens().size() – 2
        aux[0] ← A2.getTokens(i)
        aux[1] ← A2.getTokens(i + 1)
        aux[2] ← A2.getTokens(i + 2)
        aux[3] ← "i"
        aux[4] ← "" + i
        index ← binarySearch(trigrama, aux); //Se verifica existencia de trigrama
        Si  index >= 0 //Si existe se agrega la posición de nueva aparición
            aux[0] ← trigrama.get(index)[0]
            aux[1] ← trigrama.get(index)[1]
            aux[2] ← trigrama.get(index)[2]
            aux[3] ← trigrama.get(index)[3]
```

```
                    Si  trigrama.get(index)[4].equalsIgnoreCase("i")
                            aux[4] ← "" + i
                    Otro
                            aux[4] ← trigrama.get(index)[4] + ":" + i
                    Fin Si
                    trigrama.set(index, aux)
            Otro
                    trigrama.add(aux)
                    Sort(trigrama)
            Fin Si
        Fin Para
        similitudT ← 0
        //Se cuenta el número de trigramas en común
        Para i ←0 hasta i < trigrama.size()
            Si  ( !trigrama.get(i)[3].equalsIgnoreCase("i")  &&
                  !trigrama.get(i)[4].equalsIgnoreCase("i") )
                      similitudT++
            Fin Si
        Fin Para
        similitudT ← similitudT / trigrama.size();//Se calcula similitud de archivos
        //Solo se mantienen los trigramas en común
         Para i ←0 hasta i < trigrama.size()
            Si ( trigrama.get(i)[3].equalsIgnoreCase("i")  ||
                 trigrama.get(i)[4].equalsIgnoreCase("i") )
                      trigrama.remove(i)
                      i ← i-1
            Fin Si
        Fin Para
Fin
```

Pseudocódigo 8: comparaT

3.2.12. Caso de uso: Presentar Resultados

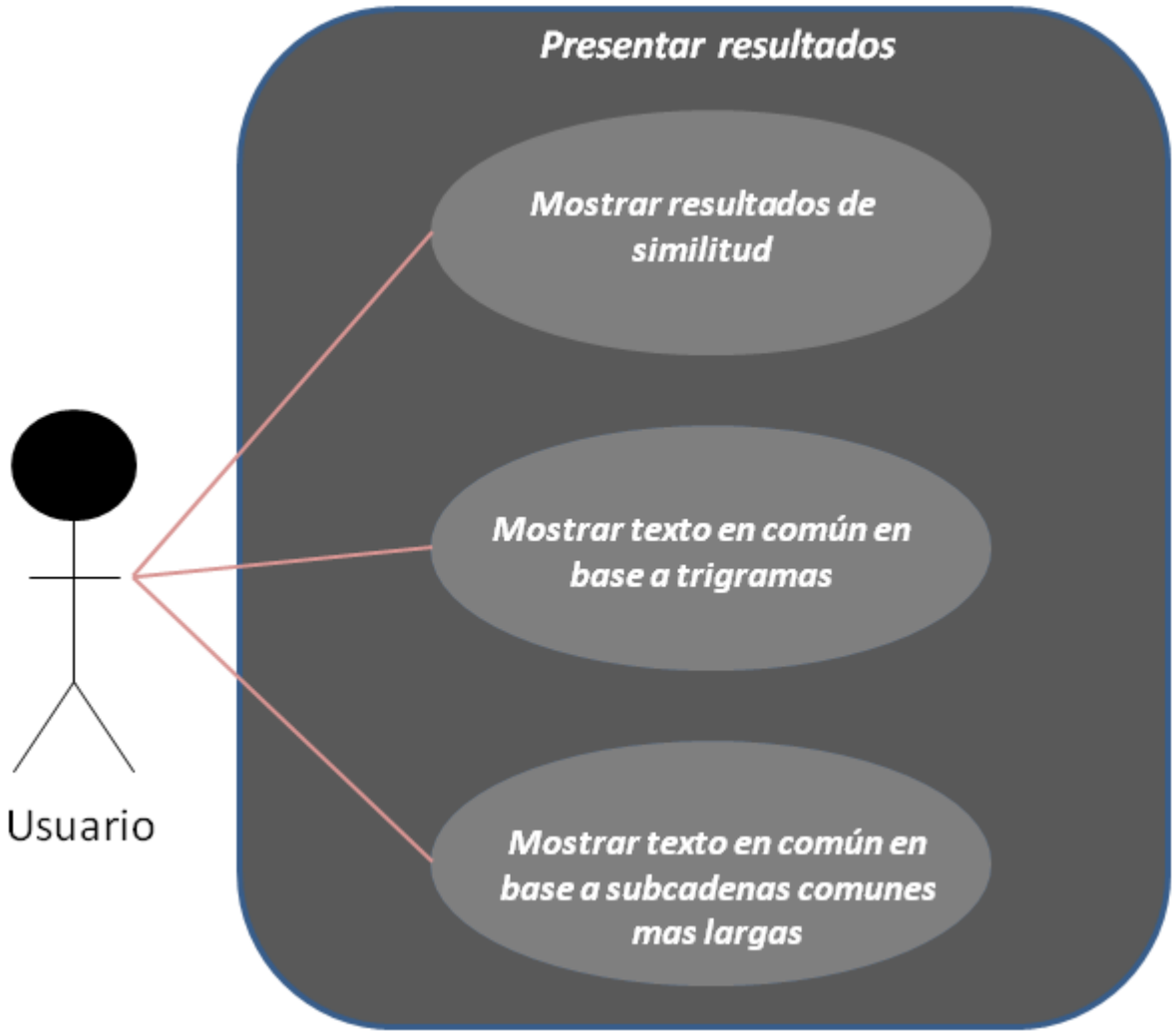

Figura 5: Caso de uso Presentar resultados

Mostrar resultados de similitud: El sistema muestra una lista de los pares de archivos comparados mostrando las medidas de similitud obtenidas.

Mostrar texto en común en base a trigramas: El sistema mostrara el texto en común en base a trigramas del par de archivos seleccionado por el usuario realizando un marcado del texto.

Mostrar texto en común en base a subcadenas comunes más largas: El sistema mostrara el texto en común en base a subcadenas comunes del par de archivos seleccionado por el usuario realizando un marcado del texto.

3.2.13. PR-1: Mostrar resultados de similitud

Descripción: Caso de uso que muestra al usuario las medidas de similitud obtenidas al analizar cada par de archivos.

Actores:

- Primario: Ninguno
- Secundario: Usuario

Disparador: El sistema finalizo la detección de similitud en el texto en todos los pares de archivos.
Precondiciones: Los archivos fueron analizados.

Poscondiciones: Se muestran los resultados al usuario.

Flujo Principal:

1. El sistema Indica que se finalizó el análisis de los archivos.
2. El sistema muestra una lista de los resultados de similitud.
3. Se despliega menú de resultados.

Requerimientos no funcionales
Para cada par de archivos las cuatro medidas de similitud obtenidas son visibles al usuario.

3.2.14. PR -2: En base a subcadenas comunes más largas

Descripción: Caso de uso que muestra al usuario el texto en común del par de archivos seleccionados en base a las subcadenas comunes más largas.

Actores:

- Primario: Usuario
- Secundario: Ninguno

Disparador: El actor primario solicita ver el texto en común de los archivos en base a subcadenas comunes más largas.

Precondiciones: Los archivos fueron analizados.

Poscondiciones: Se muestra el texto en común con un marcado en el texto.

Flujo Principal:

1. El actor primario oprime el botón “Texto en común SCML”.
2. El sistema muestra los resultados realizando un marcado del texto en común.
3. Se despliega menú de resultados.

Requerimientos no funcionales

El color del marcado en el texto debe ser un color que no dificulte su lectura.

3.2.15. PR -3: En base a trigramas

Descripción: Caso de uso que muestra al usuario el texto en común del par de archivos seleccionados en base a los trigramas en común.

Actores:

- Primario: Usuario
- Secundario: Ninguno

Disparador: El actor primario solicita ver el texto en común de los archivos en base a Trigramas.

Precondiciones: Los archivos fueron analizados.

Poscondiciones: Se muestra el texto en común con un marcado en el texto.

Flujo Principal:

1. El actor primario oprime el botón “Texto en común TRIG”.
2. El sistema muestra los resultados realizando un marcado del texto en común.
3. Se despliega menú de resultados.

Requerimientos no funcionales

El color del marcado en el texto debe ser un color que no dificulte su lectura.

4. Clases del sistema

A continuación se presentan las clases del sistema las cuales siguen el patrón de diseño Modelo-Controlador-Vista.

Modelo

El modelo es la representación específica de la información con la cual el sistema opera, es decir, define las reglas de negocio. Las clases del modelo son:

La clase *Archivo*, ver figura 6, define los atributos y los métodos para obtener y modificar la información necesaria de los archivos de texto a procesar.

Figura 6: Clase Archivo

La clase *HuellaDigital*, ver figura 7, define los atributos y los métodos para obtener y modificar la información necesaria de la huella digital de un archivo de texto.

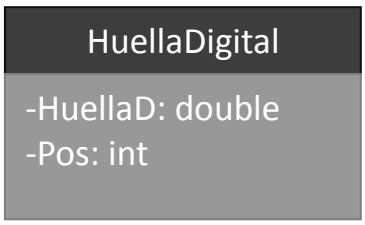

Figura 7: Clase HuellaDigital

La clase *Match*, ver figura 8, define los atributos y los métodos para obtener y modificar la información necesaria para formar las cadenas comunes de cada archivo de texto.

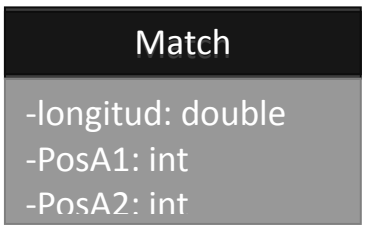

Figura 8: Clase Match

La clase *SCM*, ver figura 9, define los atributos y los métodos para obtener y modificar la información necesaria de cada token para formar las cadenas comunes más largas de cada archivo de texto.

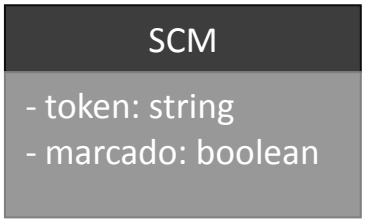

Figura 9: Clase SCM

Controladores

Un controlador es responsable de responder a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y la vista, es decir, contiene las reglas de gestión de eventos. Las clases controladoras son:

La clase *SimilitudEstructura*, ver figura 10, contiene el método para realizar la detección de similitud en estructura de los archivos de texto a procesar. Además, contiene el atributo *similitudE* para representar numéricamente la similitud entre los archivos analizados.

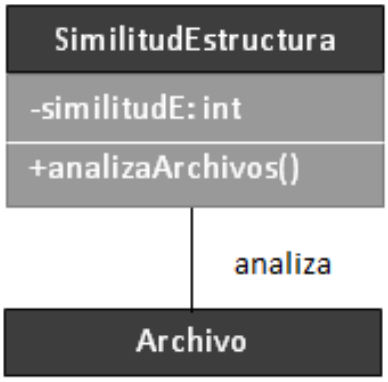

Figura 10: Clase Archivo

La clase *TecnicasPLN*, ver figura 11, contiene el método que se encarga del procesamiento de los archivos de texto de entrada con técnicas PLN de tal forma que los archivos queden listos para la detección de similitud en el texto.

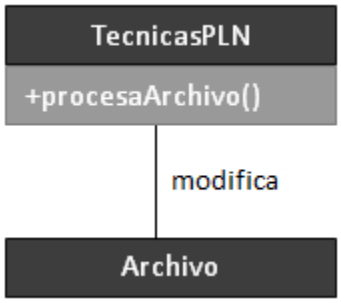

Figura 11: Clase TecnicasPLN

La clase *SimilitudHD*, ver figura 12, contiene el método que se encarga de la detección de similitud en el texto en base a la comparación de huellas digitales, a saber: *comparaHD()*. Además, contiene los atributos *similitudHD* para representar numéricamente el resultado obtenido, y *HD* que representa la huella digital en común de cada par de archivos comparados.

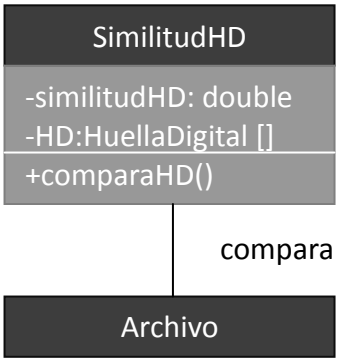

Figura 12: Clase SimilitudHD

La clase *SimilitudSCL*, ver figura 13, contiene el método que se encarga de la detección de similitud en el texto en base a la comparación de subcadenas comunes, a saber: *comparaSCL()*. Además, contiene los atributos *similitudSCL* para representar numéricamente el resultado obtenido, y *tile* que representa las cadenas en común de cada par de archivos comparados.

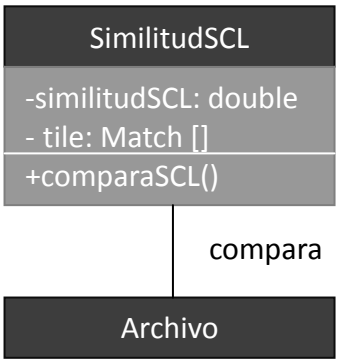

Figura 13: Clase SimilitudSCL

La clase *SimilitudT*, ver figura 14, contiene el método que se encarga de la detección de similitud en el texto en base a la comparación trigramas, a saber: *comparaT()*. Además, contiene los atributos *similitudT* para representar numéricamente el resultado obtenido, y *trigrama* para representar los trigramas en común de los archivos comparados.

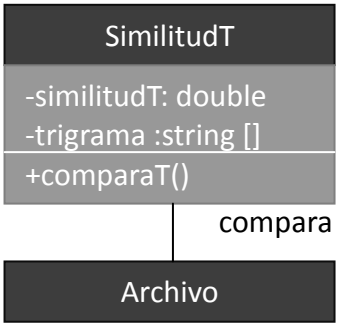

Figura 14: Clase SimilitudT

La clase *Resultados*, ver figura 15, contiene los métodos para presentar los resultados obtenidos de la comparación de los archivos. Esta clase muestra el texto en común de los archivos ya sea en base a las subcadenas comunes más largas o en base a los trigramas en común.

Figura 15: Clase Resultados

Vista

La vista presenta el modelo en un formato adecuado para interactuar con el usuario mediante la interfaz grafica. La interfaz grafica está definida por las siguientes clases:

La clase *ResultadosG*, ver figura 16, contiene el método que se encarga de controlar la interfaz grafica del usuario, donde se muestran los resultados del análisis.

Figura 16: ResultadosG

La clase *Principal*, ver figura 17, lleva el control general del sistema. Además aquí se realiza la selección de los archivos a analizar y la detección de similitud en estructura.

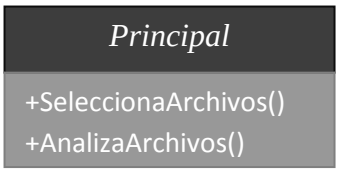

Figura 17: Principal

5. Diagrama de Clases

A continuación se puede observar en la figura 18 el diagrama general de paquetes, en el cual se ve la interacción de los paquetes Modelo-Controlador-Vista.

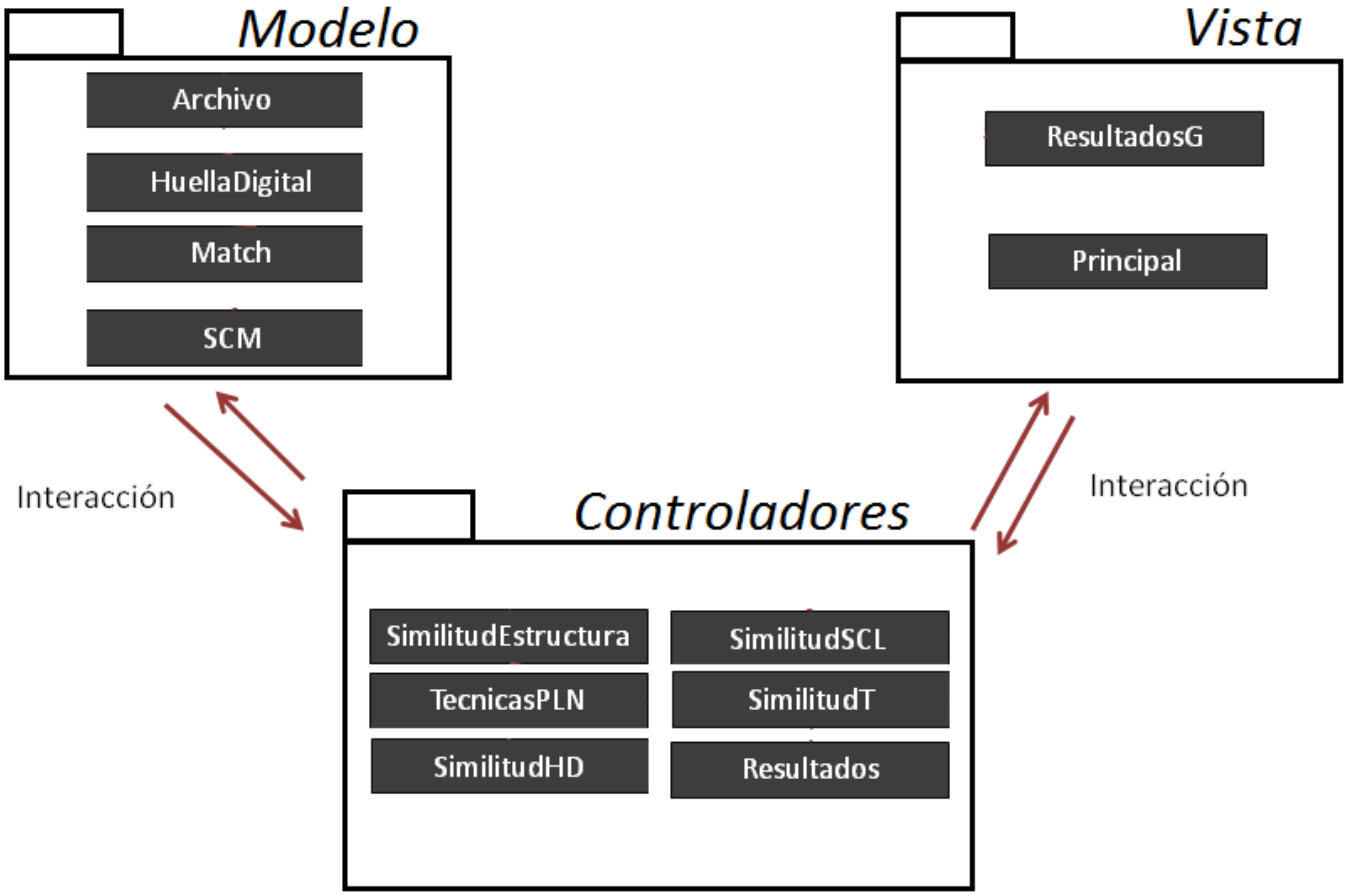

Figura 18: Diagrama de paquetes

En la figura 19 podemos observar el diagrama general de clases en el cual podemos notar las relaciones y la cardinalidad entre las clases que conforman el sistema.

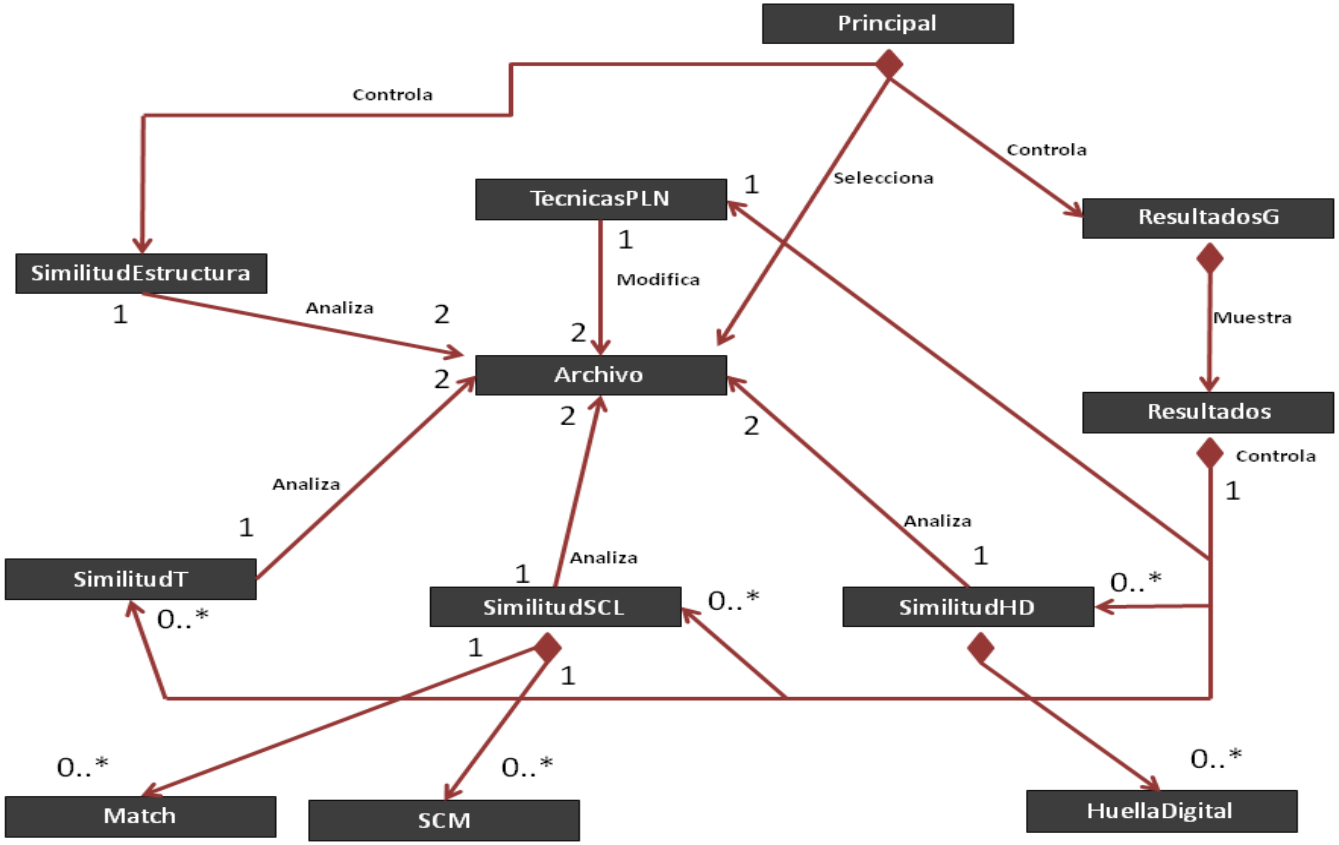

Figura 19: Diagrama de clases

6. Análisis de robustez

En la figura 20, se muestra un diagrama general de secuencia en el cual se puede observar el proceso del sistema desde que el usuario realiza la selección de los archivos hasta que la presentación de resultados de la comparación.

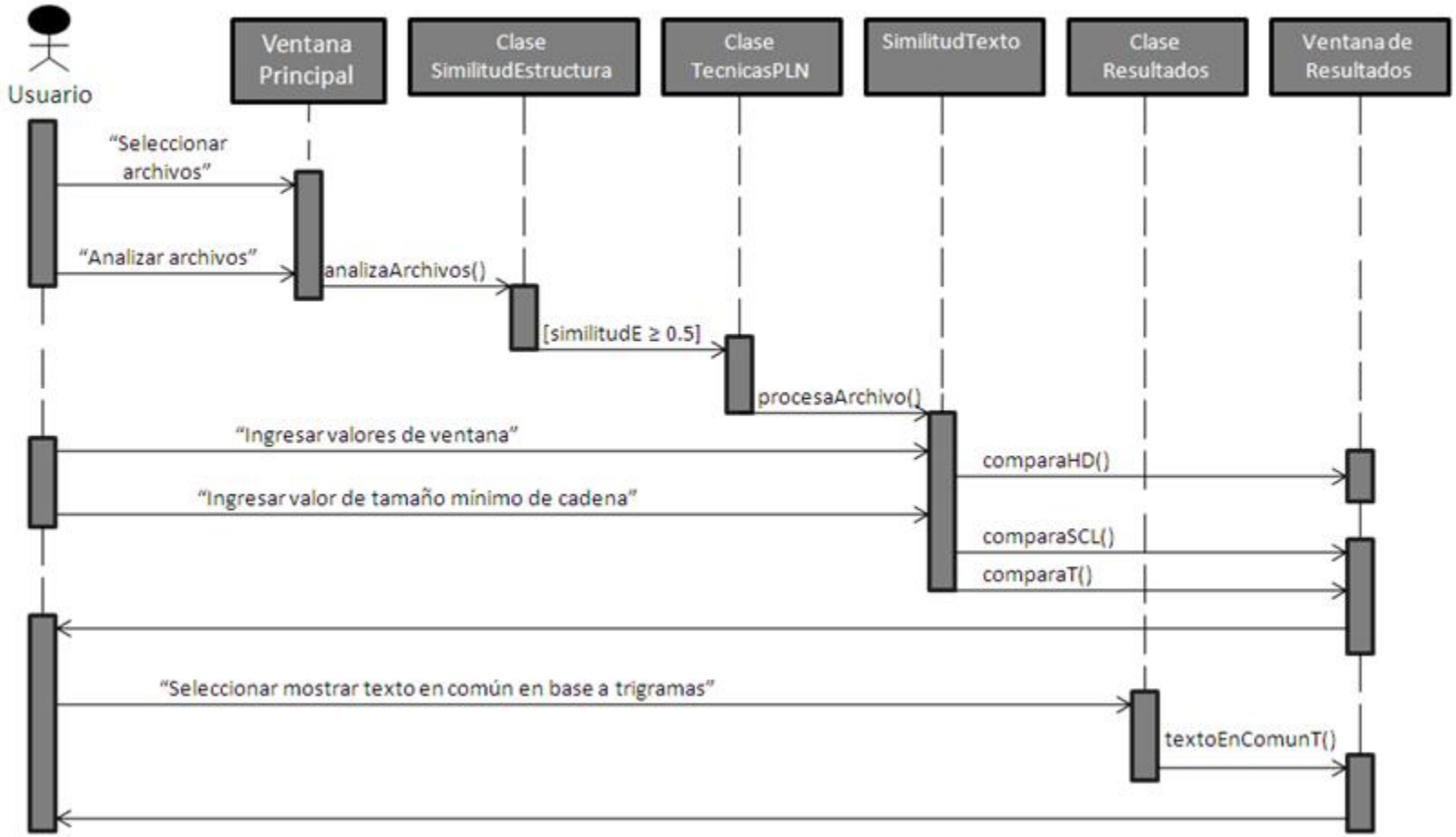

Figura 20: Diagrama general de secuencia

Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Manual de usuario

Sistema de detección de plagio en archivos de texto

Creado por

Roberto Iván Morán Torres

Matrícula: 208205737

Asesora

Dra. Ma. Lizbeth Gallardo López

Departamento de Sistemas

Trimestre 13-I

Marzo 2013

Índice

1. Introducción

En este documento se presentan los módulos del *sistema de detección de plagio en archivos de texto*; mostrando las impresiones de pantalla de cada uno de los módulos sobre los cuales un usuario puede interactuar. El primer módulo se encarga de seleccionar los documentos a analizar y detectar su similitud en estructura. El segundo módulo está encargado de procesar el texto de los documentos con técnicas de procesamiento de lenguaje natural. El tercer módulo tiene las tareas de detectar la similitud en el texto, en base a tres comparaciones: por trigramas, por huella digital y por subcadenas comunes más largas. El cuarto módulo tiene como tarea presentar al usuario los resultados del análisis así como el texto en común de los archivos analizados.

2. Inicio del sistema

Para ingresar al sistema, basta con iniciar la aplicación, no es necesario autenticarse con el sistema de detección de plagio. En la pantalla principal (ver figura 1) se puede visualizar el botón de *Selecciona archivos* y una etiqueta que muestra la cantidad de archivos seleccionados.

Al pasar el ratón por encima del botón *Seleccionar archivos* se muestra una leyenda que informa al usuario que se deben seleccionar al menos dos archivos para proceder al análisis de detección de plagio y un máximo de 50. Este valor de 50 archivos se escogió porque la cantidad de veces que se requiere analizar a los pares de archivos tiene un orden cuadrático O(n^2); las pruebas revelaron que al seleccionar como máximo 50 archivos, que no exceden 10 cuartillas, el sistema mantiene un tiempo de respuesta aproximado de 19 minutos.

Figura 1: Inicio del sistema

3. Selección de archivos

Al oprimir el botón de *Seleccionar archivos*, como se observa en la figura 2, se abre una ventana que permite navegar entre los subdirectorios hasta encontrar los archivos a analizar. Es importante mencionar que los archivos pueden existir en distintas carpetas. El sistema filtra los archivos con extensión .doc, .docx, .odt ó .txt.

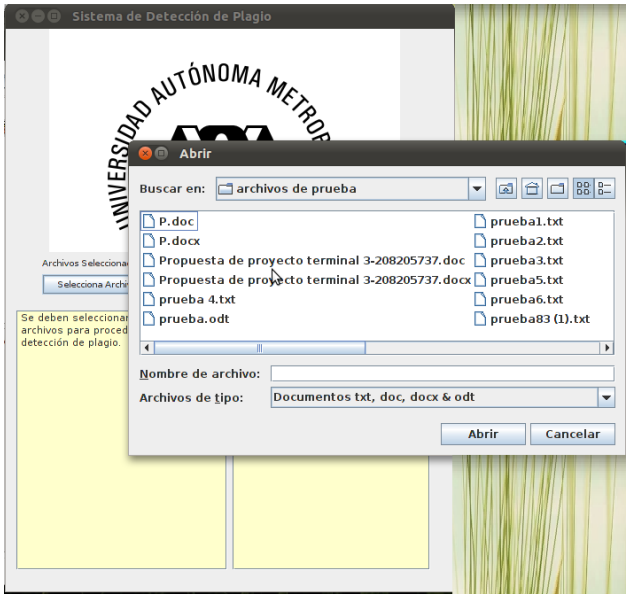

Figura 2: Selección de archivos

En dado caso que se seleccione mostrar todos los archivos y se seleccionen archivos con distintas extensiones, el sistema las valida rechazando otro tipo de archivos y mostrando al usuario un mensaje de error, ver figura 3.

Figura 3: Selección de archivos erroneos

4. Parámetros de algoritmos

Después de haber seleccionado dos ó más archivos, se visualizan las casillas para ingresar los parámetros numéricos para los algoritmos de detección de similitud *huella digitales* y *cadenas comunes*. Existe una leyenda de ayuda que explica los parámetros que se ingresan para los algoritmos de comparación, la cual describe el significado de cada parámetro y los valores recomendados. Los valores que se recomiendan para huella digital son k=7 y t=27 y para cadenas comunes se recomienda $5 \leq M \leq 9$ ya que en las pruebas realizadas se obtuvo un mejor resultado con estos valores, ver figura 4.

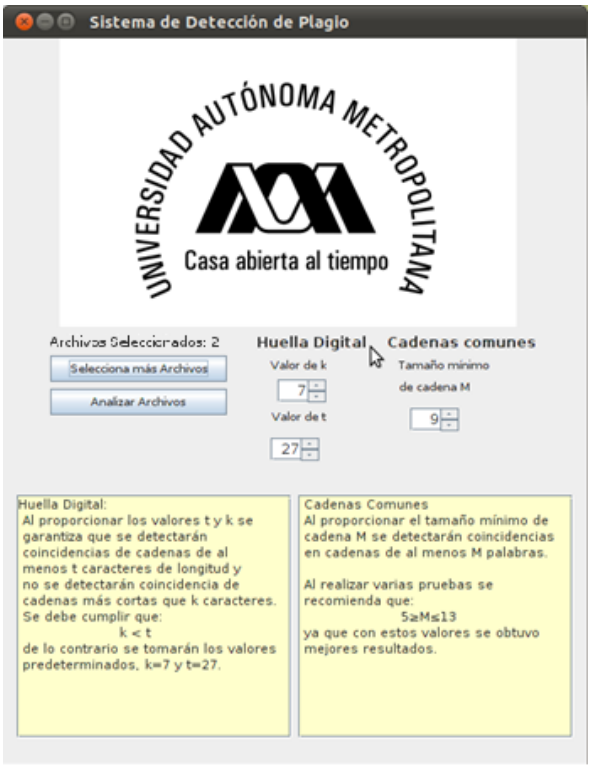

Figura 4: Parámetros de algoritmos

También existe una leyenda de ayuda que da una descripción de los pasos consistentes del análisis de los archivos la cual contiene la información siguiente:

Se analizarán todos los archivos seleccionados todos contra todos.

El análisis completo de un par de archivos consiste de cuatro posibles comparaciones

1. Se comparan los archivos en base a su estructura, es decir en base a su número de palabras y párrafos . Si la similitud en base
   a su estructura es mayor al 50% se continúa con las siguientes etapas.
2. Se comparan los archivos en base a los trigramas en común.
3. Se comparan los archivos en base a su huella digital.
4. Se comparan los archivos en base a sus cadenas comunes.

Esta ayuda se muestra al pasar el ratón por encima del botón *Analizar Archivos*, ver figura 6.

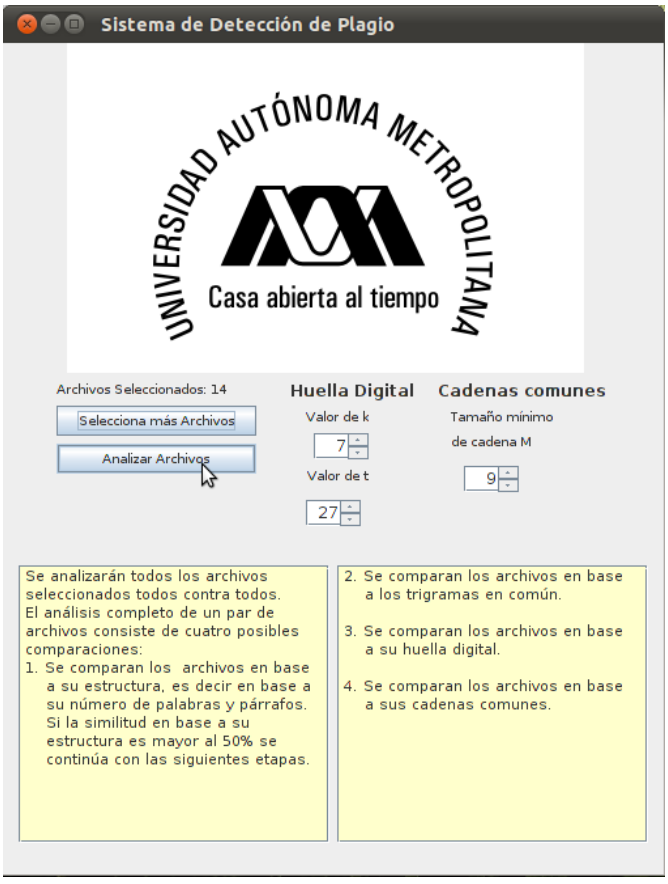

Figura 6: Descripción de análisis

5. Analisis de archivos

Al oprimir el botón de *Analizar Archivos*, el sistema procede a analizar por pares todos los archivos seleccionados. Al finalizar el análisis, el sistema muestra la ventana de resultados, ver figura 7, donde se despliega una lista con los nombres de cada par de archivos analizados, los cuales pueden estar marcados en tres colores distintos: blanco, rosa y rojo.

- El color blanco indica que la similitud entre los archivos es nula;
- El color rosa indica que la similitud entre los archivos es baja; y
- El color rojo indica que la similitud entre los archivos es alta.

Al seleccionar un par de archivos específico se muestran los resultados sobre el grado de similitud calculada a través de: la estructura, cadenas comunes, huella digital y trigramas comunes; además, se puede observar el texto en común de los archivos ya sea en base a trigramas o en base a cadenas comunes si es que la similitud estructural fue mayor o igual al 0.5 (ver figura 8). En este caso el texto en común de los archivos se marca de color amarillo; además, se muestra una lista de los trigramas y cadenas en común; al seleccionar una de ellas, se muestra su ubicación en el texto marcándolas de color verde.

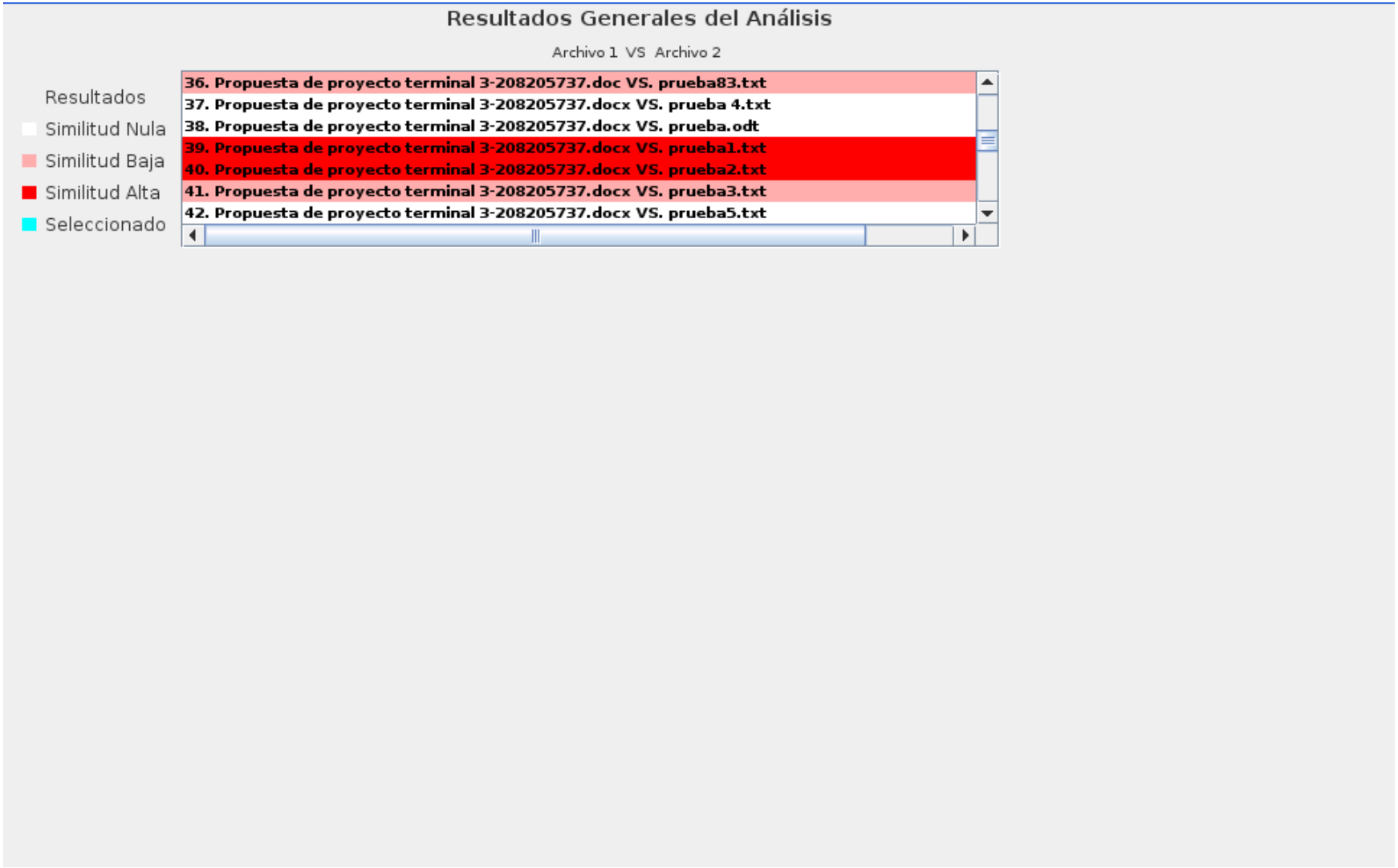

Figura 7: Resultados generales

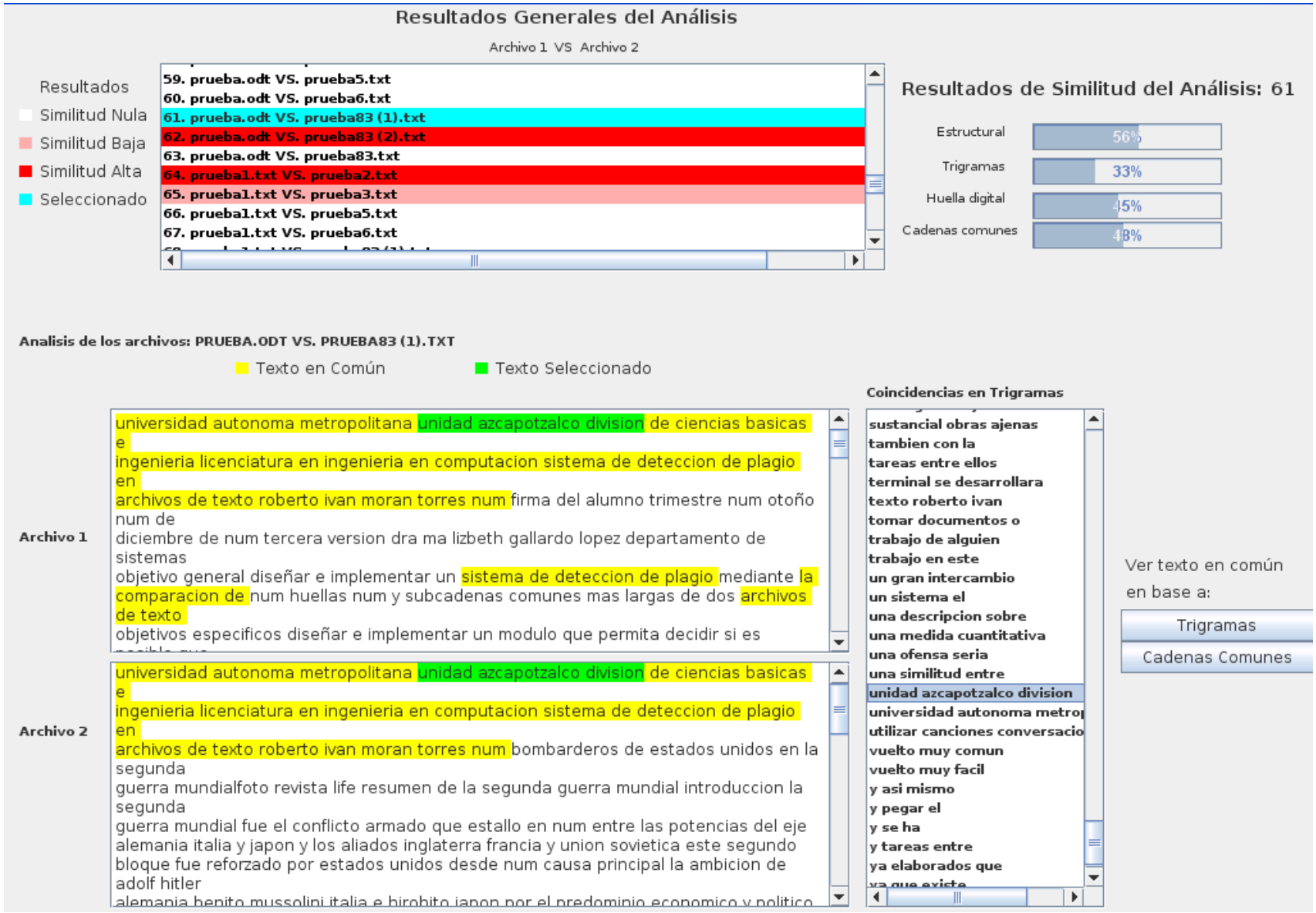

Figura 8: Resultados específicos

Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Manual Técnico

Sistema de detección de plagio en archivos de texto

Creado por

Roberto Iván Morán Torres

Matrícula: 208205737

Asesora

Dra. Ma. Lizbeth Gallardo López

Departamento de Sistemas

Trimestre 13-I

Marzo 2013

Índice

1. Introducción

Este proyecto terminal que lleva por nombre: S*istema de detección de plagio en archivos de texto*, se desarrolló como un sistema de escritorio (*standalone*) utilizando el manejador de base de datos MySQL y el lenguaje de programación Java.

En el presente documento se muestra la estructura del directorio del sistema desarrollado en el proyecto terminal. También se presenta la tecnología empleada en el desarrollo del proyecto; así como el proceso de instalación que se debe seguir para que otro usuario pueda utilizar el sistema.

2. Tecnología Empleada

Para la realización del proyecto se utilizó el siguiente hardware:

- HP pavilion dv4-2145dx entertainment notebook PC con procesador AMD Turion(tm) II Dual-Core Mobile M520 2.30GHz, memoria Ram de 4.00 gigabytes y disco duro de 320 gigabytes.

El software utilizado para el desarrollo del proyecto es de código abierto, y de libre distribución:

- Sistema operativo Debian GNU/Linux.
- NetBeans IDE 7.2: Entorno de desarrollo integrado de código abierto.
- MySQL: Manejador de bases de datos de código abierto.

Debian GNU/Linux es un sistema operativo estable, y es de libre distribución.

NetBeans es un entorno de desarrollo integrado, fácil de usar, cómodo y de excelente calidad y es de libre distribución. Además cuanta con herramientas de depuración que facilita el desarrollo de software.

MySQL es un sistema gestor de bases de datos estable, y cuenta con suficiente documentación en su página Web oficial; además, la versión de WordNet utilizada está orientada a operar en MySQL.

Se utilizó Apache POI[1] en su versión 3.8, ya que es la versión más estable hasta el momento; además se cuenta con documentación disponible en la página oficial de Apache. POI es una API de java para la manipulación de documentos de Microsoft; lo cual se aprovechó para extraer el texto de documentos de Microsoft Word.

También se utilizó JDOM[2] en su versión 2.0.4, ya que es una versión estable hasta el momento; además cuenta con documentación disponible en su página oficial. JDOM es una API de java para acceder, manipular y crear documentos XML. Lo cual fue utilizado para extraer el texto de documentos generados con OpenOffice.

Finalmente se utilizó MCR WordNet 3.0[3] como base de palabras del idioma español, cuya versión está disponible gratuitamente, además cuenta con una versión para MySQL.

[1] Apache POI es una API para manipular distintos formatos de archivo basados en el formato de Microsoft OLE2.
[2] JDOM es una API basada en Java para acceder, manipular y generar datos en XML desde el código Java.
[3] WordNet es una gran base de datos léxica que contiene: sustantivos, verbos, adjetivos y adverbios; las palabras están agrupadas por sinónimos cognitivos y por lemas.

3. Estructura del directorio del sistema

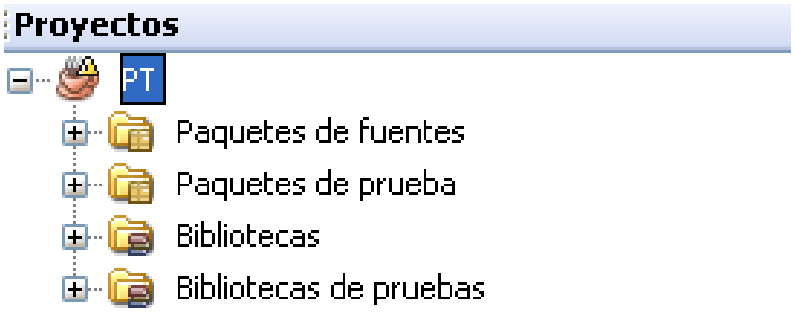

Figura 1: Esquema general del directorio

En la figura 1 se muestra el esquema general del directorio del s*istema de detección de plagio en archivos de texto*, en el que se pueden observar 4 carpetas. A continuación se presenta la descripción de cada una de ellas y su contenido.

3.1. Paquetes de fuentes

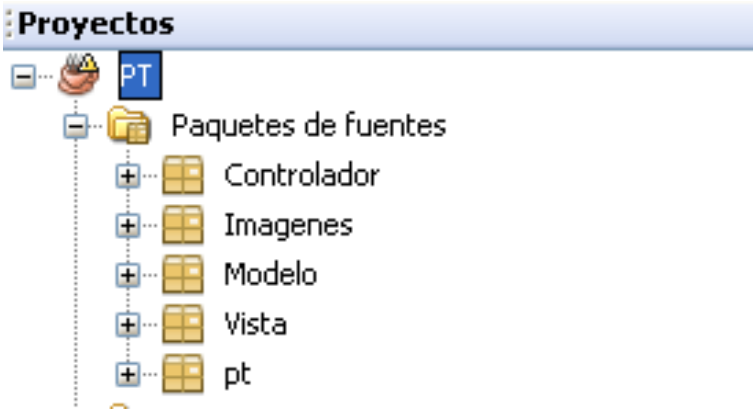

Figura 2: Paquetes de fuentes

En la carpeta Paquetes de Fuentes, podemos encontrar cinco paquetes como se observa en la figura 2, siguiendo el patrón de diseño MVC (Modelo Vista Controlador). Los paquetes son:

3.1.1. Modelo

En este paquete se encuentran las clases que definen todo la lógica de negocio incluyendo la clase *Conexion*, la cual nos permite la conexión a la base de datos contenedora de MCR WordNet 3.0 en español, ver figura 3.

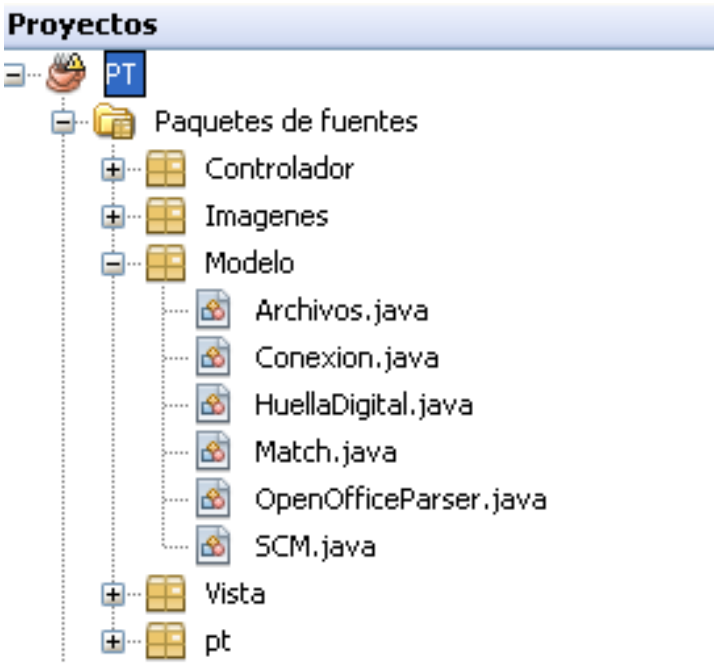

Figura 3: Paquete *Modelo*

3.1.2. Controlador

En este paquete se encuentran las clases que manipulan los datos y muestran los resultados y controlan el flujo del sistema según las acciones que tome el usuario, ver figura 4.

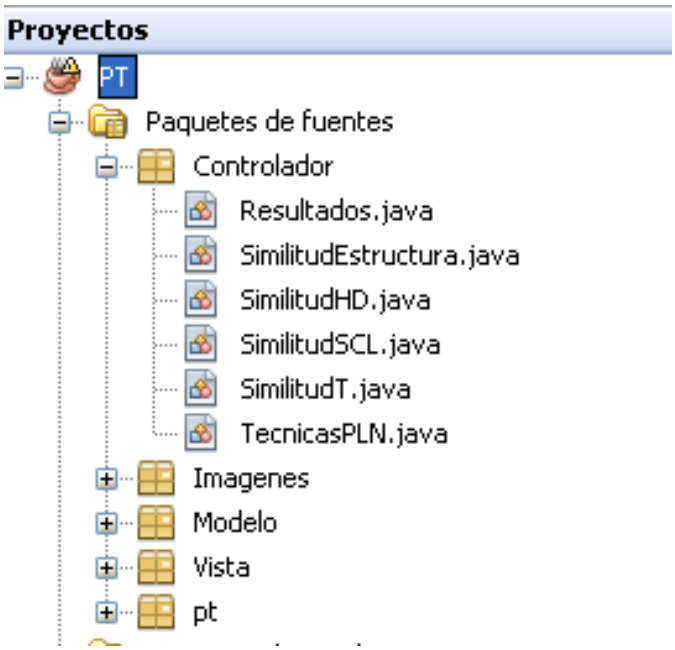

Figura 4: Paquete *Controlador*

3.1.3. Vista

En este paquete se encuentran las clases que contienen el código de generación de las ventanas diseñadas utilizadas en el sistema, ver figura 5.

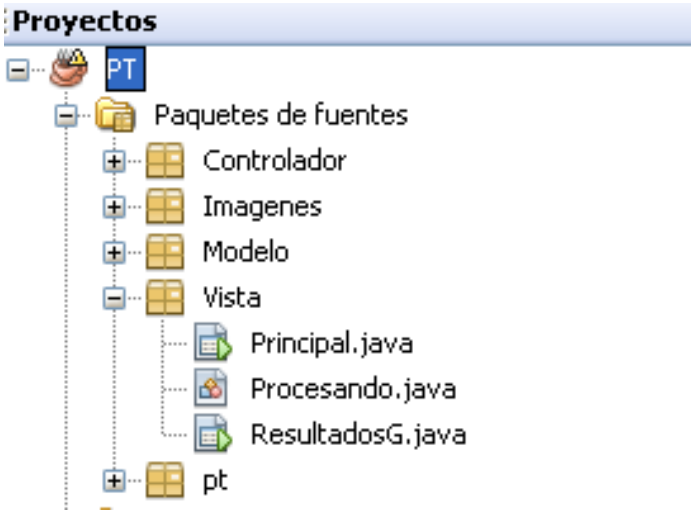

Figura 5: Paquete *Vista*

3.1.4. Imágenes y PT

En el paquete *Imagenes* se encuentra almacenada la imagen del logotipo de la UAM utilizada como fondo de la ventana principal. En el paquete *pt* se encuentra la clase *Main* que se encarga de inicializar el sistema, ver figura 6.

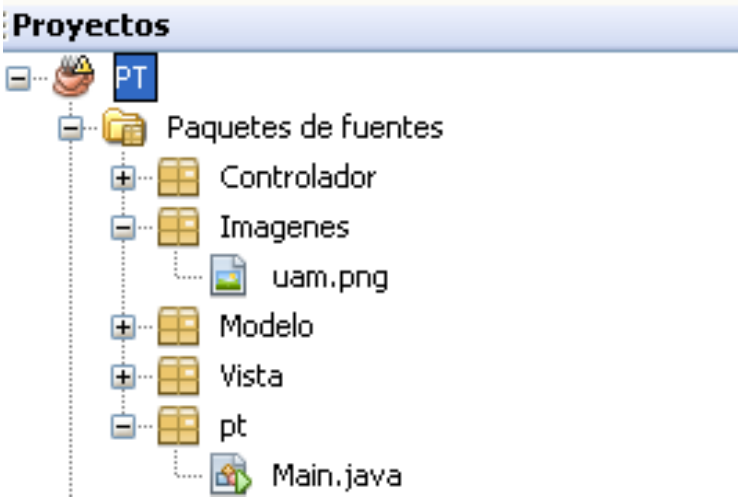

Figura 6: Paquetes *Imagenes y pt*

3.2. Paquetes de prueba y Bibliotecas de prueba

En estas carpetas se almacenan las clases de prueba y las bibliotecas de prueba; sin embargo, para el proyecto terminal no fueron utilizadas.

3.3. Bibliotecas

En esta carpeta se encuentran todas las bibliotecas necesarias para el correcto funcionamiento del sistema, ver figura 7.

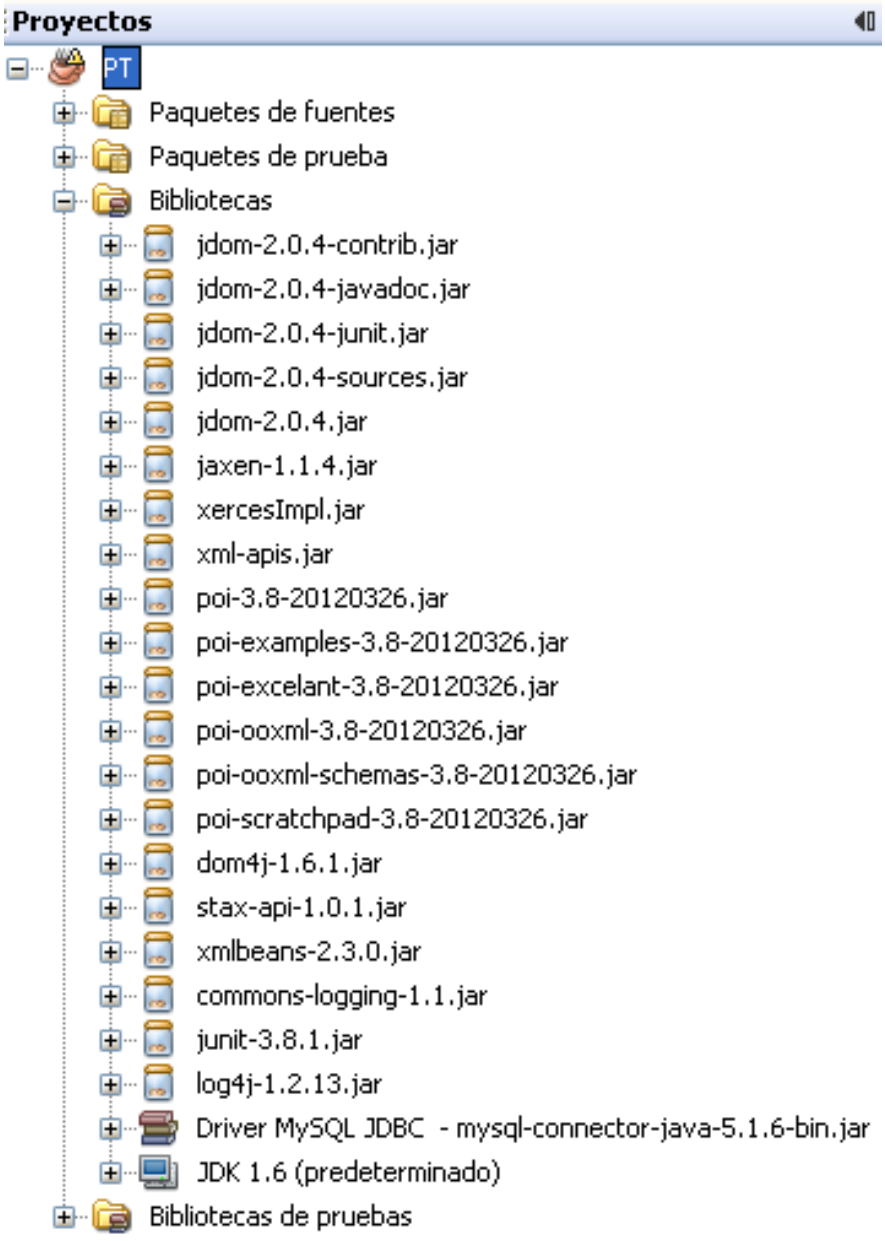

Figura 7: Bibliotecas

4. Proceso de instalación del sistema

Para poder instalar y utilizar el sistema, se requiere la instalación previa del software siguiente:

- MySQL Community Server en su versión 5.6.10. Disponible en: *http://dev.mysql.com/downloads/mysql/*
- El JDK de Java. Disponible en: *http://www.oracle.com/technetwork/java/javase/downloads/index.html*

4.1. Instalación de la base de datos

Para poder instalar la base de datos contenedora de MCR WordNet 3.0 en el servidor MySQL se deben seguir los siguientes pasos:

1. Descargar la carpeta MCR que contiene WordNet 3.0 para MySQL disponible en: *http://adimen.si.ehu.es/web/MCR/*
2. Abrir una terminal.
3. Acceder al servidor MySQL.
4. Una vez dentro de la consola MySQL, si la base de datos no existe, podemos crearla con: *create database mcr30*
5. Indicamos la base de datos a usar: *use mcr30*;
6. El proceso de importación, inicia al ejecutar el script de la base de datos *$PATH_TO_MCR/sql/mysql/mcr3.0_create_tables.sql*
7. Y por último ejecutar (colocar el correcto script) *$PATH_TO_MCR/sql/mysql/mcr3.0_load_tables.sql*

4.2. Desplegar la aplicación

Para desplegar la aplicación en una PC se debe copiar la carpeta *dist* del proyecto al cual contiene el archivo *pt.jar* y las bibliotecas necesarias para su funcionamiento. Después se debe ejecutar en una terminal: *java -jar "path/dist/pt.jar"*.

5. Pruebas

El sistema solo acepta archivos de entrada con extensión .doc, .docx, .odt y .txt; por lo tanto, solo muestra los archivos con esas extensiones, ver figura 8.

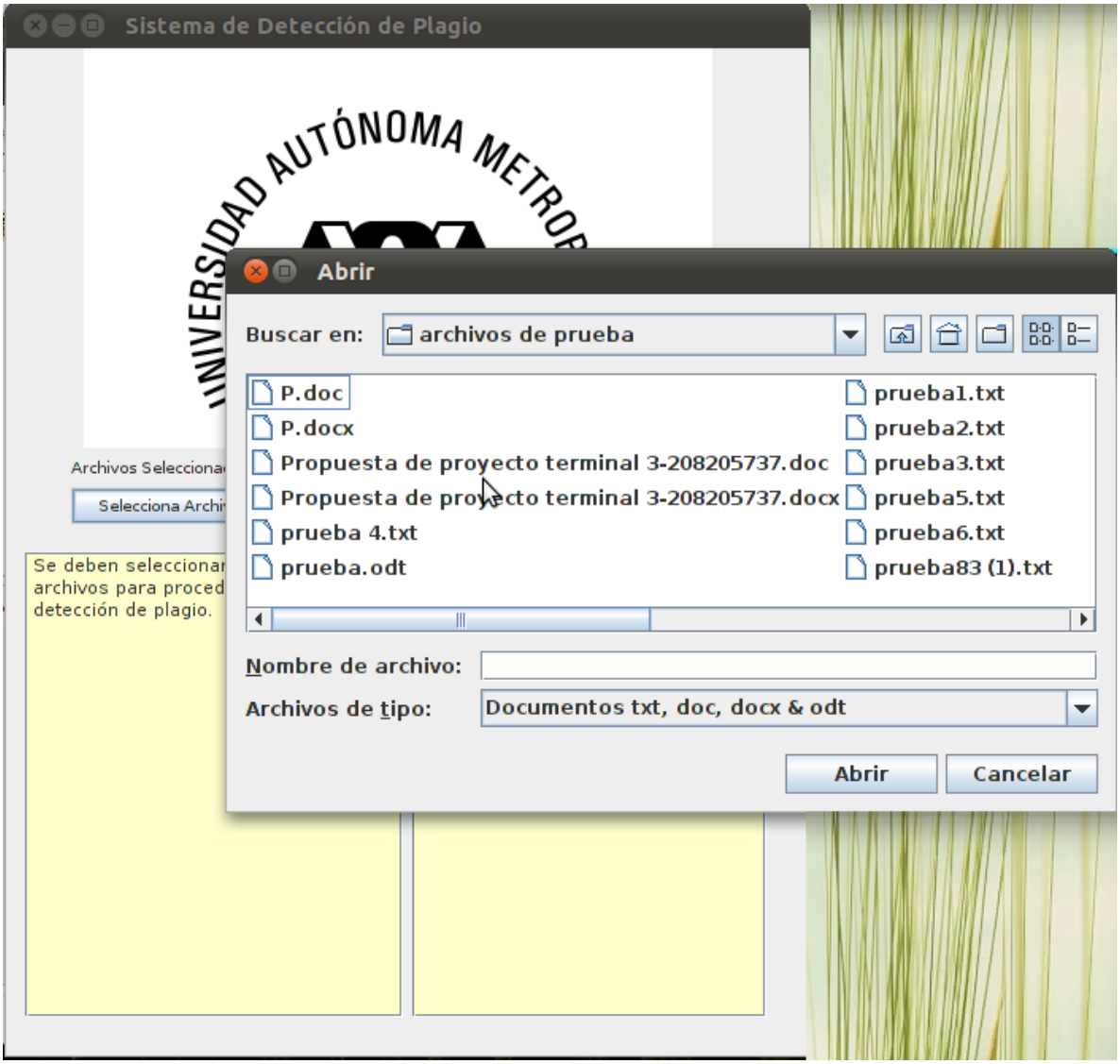

Figura 8: Filtrado de archivos

En dado caso que se seleccione mostrar todos los archivos y se seleccionen archivos con distintas extensiones, el sistema las valida rechazando otro tipo de archivos y mostrando al usuario un mensaje de error, ver figura 9.

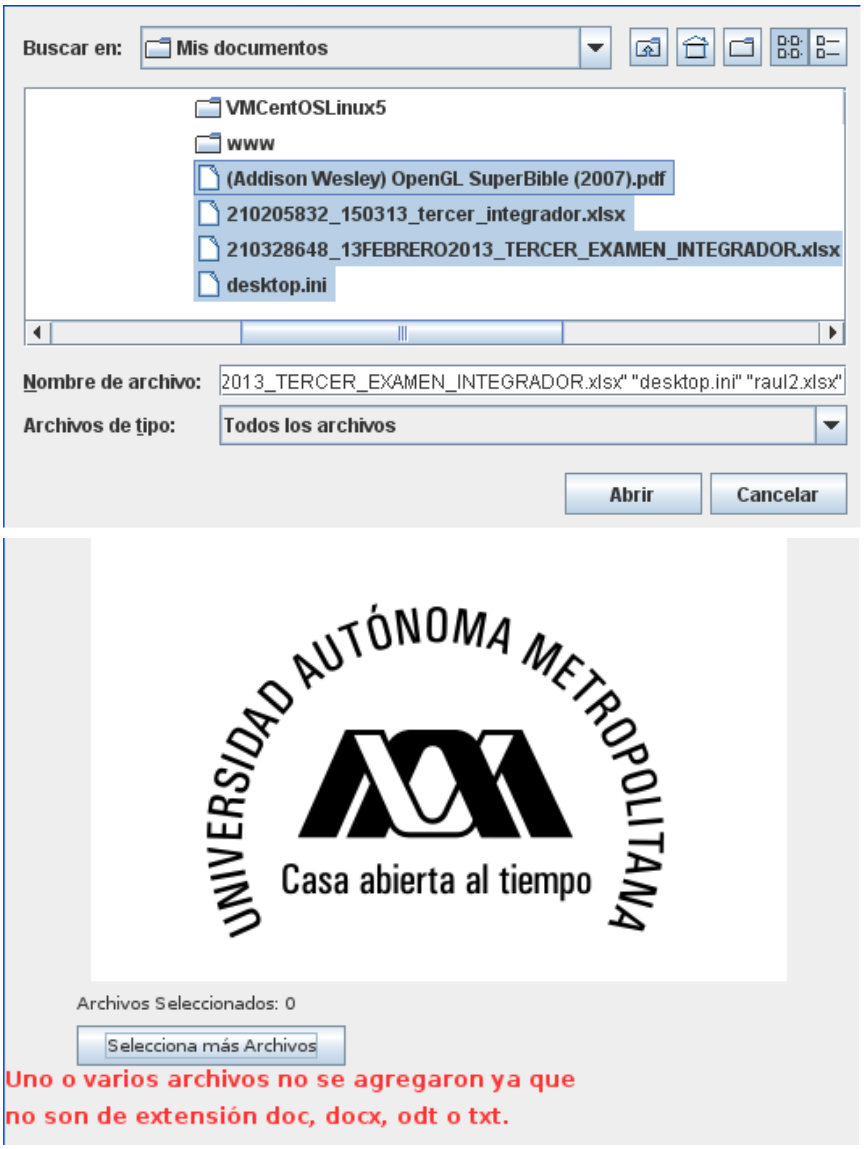

Figura 9: Archivos inválidos

Otra prueba que se realizó fue medir el desempeño del sistema al ingresar una gran cantidad de archivos; con lo cual se decidió establecer la selección de un máximo de 50 archivos a la vez. Este valor de 50 archivos se escogió porque la cantidad de veces que se requiere analizar a los pares de archivos tiene un orden cuadrático $O(n^2)$; las pruebas revelaron que al seleccionar como máximo 50 archivos, que no exceden 10 cuartillas, el sistema mantiene un tiempo de respuesta aproximado de 19 minutos.

Por último se realizaron pruebas con distintos parámetros para los algoritmos de detección de similitud; los valores que se recomiendan para huella digital son k=7 y t=27 y para cadenas comunes se recomienda $5 \leq M \leq 9$ ya que en las pruebas realizadas se obtuvo un mejor resultado con estos valores

6. Dificultades durante el desarrollo

Durante el desarrollo de este proyecto, surgieron varias dificultades, tales como: al leer un archivo *.txt,* el manejo de caracteres del idioma español (ñ, á, é, í, ó ú) eran interpretados de otra manera. Esto se resolvió configurando el tipo de *encoding* a UTF8 de los archivos.

También, surgieron problemas al tratar de extraer el texto de archivos con extensión .doc, *.docx* y *.odt,* ya que estos son archivos XML comprimidos y no se puede extraer el texto directamente. Esto se resolvió utilizando las bibliotecas Apache POI 3.8 y JDOM 2.0.4. La biblioteca Apache POI 3.8 contiene métodos que permite extraer el texto de documentos de *Microsoft* (*.doc* y *.docx*). La biblioteca JDOM 2.0.4 contiene métodos de descompresión y manipulación de archivos XML con la cual se pudo extraer el texto de documentos *OpenOffice* (*.odt).*

Otra dificultad surgió al momento de generar los trigramas de los archivos de texto, ya que al irlos generando se debía verificar que estos fueran únicos; es decir, que no existieran; para esto, los trigramas se comparaban contra los trigramas ya generados. Esto ocasiono que el sistema se alentara durante su ejecución, porque al tener una gran cantidad de trigramas, el nuevo trigrama se comparaba con todos. Esto se resolvió ordenando los trigramas para poder utilizar búsqueda binaria. Los trigramas fueron almacenados en forma ordenada y cada vez que se tenía un nuevo trigrama se utilizó el método de búsqueda binaria para verificar que no existiera. De no existir, el nuevo trigrama se inserta en la posición correspondiente. Con esto, la generación de trigramas se volvió más rápida.

Cabe mencionar que en la propuesta de proyecto terminal, se planteó utilizar la base de palabras MCR WordNet 3.0 que es una versión de WordNet en español, para la lematización y remplazo de sinónimos. Sin embargo, esto se logró parcialmente, porque MCR WordNet 3.0 en su versión SQL para MySQL contiene solo una parte de la base total. Como resultado, las palabras remplazadas por las técnicas de lematización y remplazo de sinónimos fueron limitadas.

Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Reporte Final de Proyecto Terminal II

Sistema de detección de plagio en archivos de texto

Creado por

Roberto Iván Morán Torres

Matrícula: 208205737

Asesora

Dra. Ma. Lizbeth Gallardo López

Departamento de Sistemas

Trimestre 13-I

Marzo 2013

Índice

Resumen

En este proyecto terminal se desarrolló un *sistema de detección de plagio sobre documentos guardados en archivos de texto*. Se trata de un sistema diseñado siguiendo el patrón de diseño Modelo-Vista-Controlador. La metodología seguida en el desarrollo del proyecto fue el Proceso Unificado, la cual se caracteriza por estar dirigida por casos de uso. El sistema está dividido en cuatro módulos. El primer módulo se encarga de seleccionar los documentos a analizar y detectar su similitud en estructura, es decir su similitud en base al número de párrafos y palabras de los documentos. El segundo módulo está encargado de procesar el texto de los documentos con las técnicas de procesamiento de lenguaje natural: *Minusculización*, *Remplazo de números*, *Limpiado*, *Lematización* y *Remplazado de sinónimos*. El tercer módulo tiene las tareas de detectar la similitud en el texto en base a tres técnicas de comparación: por trigramas [12], por huella digital [13] y por subcadenas comunes más largas [14]. El cuarto módulo tiene como tarea presentar al usuario los resultados del análisis así como el texto en común de los archivos analizados.

.

1. Introducción

De acuerdo con la Real Academia Española, *plagio* significa *copiar en lo sustancial obras ajenas, dándolas como propias*. El concepto de obra es muy amplio ya que se refiere a la creación intelectual en artes, ciencias, letras, etc. El plagio en material escrito es muy común en estos días debido a la gran cantidad de información disponible; en particular, entre los estudiantes existe un frecuente intercambio de documentos y tareas; acceden a miles de fuentes a través del Internet, en cuestión de segundos; de tal manera que los estudiantes suelen copiar y pegar parcial o totalmente el trabajo de alguien más, para reportarlo como propio.

En la UAM Azcapotzalco se imparte una gran cantidad de Unidades de Enseñanza-Aprendizaje (UEA), en las cuales su programa de estudios generalmente no cambia, es decir, el contenido sintético es el mismo trimestre tras trimestre; debido a esto las tareas y proyectos realizados en las UEA difícilmente son distintas; además, los profesores no siempre cuentan con el material necesario para modificar las tareas trimestre a trimestre. Por lo tanto, los estudiantes tienen la facilidad de conseguir los trabajos realizados en las UEA que otros cursaron en trimestres anteriores, y cuyo formato electrónico facilita su intercambio. Para los profesores resulta complicado detectar copias o similitud entre las tareas; sería muy agotador y consumiría mucho tiempo estar comparándolas de un trimestre a otro.

En base a lo anterior, se puede observar el problema que existe respecto al plagio en la Universidad. Esto es grave, ya que el plagio se considera como un fraude o robo intelectual, lo cual es una violación a los derechos de autor, y puede conducir a problemas legales.

En este proyecto terminal se desarrolló una herramienta para que facilite a los profesores la detección de plagio en archivos de texto. La herramienta está focalizada a las UEA cuyas tareas involucran ensayos, resúmenes, análisis de lecturas solicitadas en formato electrónico. Ejemplos de UEA son: Metodologías de Análisis y Diseño de Sistemas de Información, Ingeniería de Software, Economía Mexicana, entre otras. El profesor solo debe proporcionar al sistema los archivos a analizar, y en base a los resultados de similitud calculados entre pares de archivos, el profesor podrá examinar aquellos para los cuales se sospecha de plagio. De esta forma se podrá tomar la acción correspondiente sobre los alumnos que tomen el trabajo de otros; de tal forma que se cree conciencia sobre la propiedad intelectual.

La sección 2 presenta los objetivos generales y particulares del proyecto; la sección 3 describe la metodología de base empleada; la sección 4 presenta el desarrollo, en particular análisis, diseño y e implementación del sistema de detección de plagio; por último, la sección 5 presenta las conclusiones y perspectivas del proyecto terminal.

2. Objetivo General

- Diseñar e implementar un sistema de detección de plagio mediante la comparación de trigramas, huellas digitales y subcadenas comunes más largas de dos archivos de texto.

Objetivos Particulares

- Diseñar e implementar un módulo que permita decidir si es posible que exista similitud entre dos archivos de texto por medio de su estructura.
- Diseñar e implementar un módulo que aplique técnicas de procesamiento de lenguaje natural sobre los archivos de texto a comparar.
- Diseñar e implementar un módulo el cual realice la comparación de trigramas, huellas digitales y subcadenas comunes más largas de dos archivos de texto.
- Diseñar una interfaz de resultados que desplegué las medidas de similitud y permita al usuario visualizar el texto en común entre los archivos comparados.

3. Metodología de base empleada en el proyecto

En la elaboración de este proyecto terminal se optó por utilizar el Proceso Unificado de desarrollo de software el cual se basa en un diseño incremental impulsado mediante la construcción de vistas de la arquitectura del sistema con el apoyo de casos de uso. Esta metodología tiene las siguientes características clave:

- Se basa en componentes, usualmente para coordinar con proyectos de programación orientado a objetos.
- Utiliza UML, una notación de diagramas para el diseño orientado a objetos.
- Se basa y es impulsado por casos de uso que ayudan a mantener la vista de los comportamientos esperados del sistema.
- Es centrado en la arquitectura.
- El diseño es iterativo e incremental.

El diseño en el proceso unificado pasa a través de una serie de ciclos donde cada ciclo tiene cuatro fases fundamentales como se muestra en la figura 1.

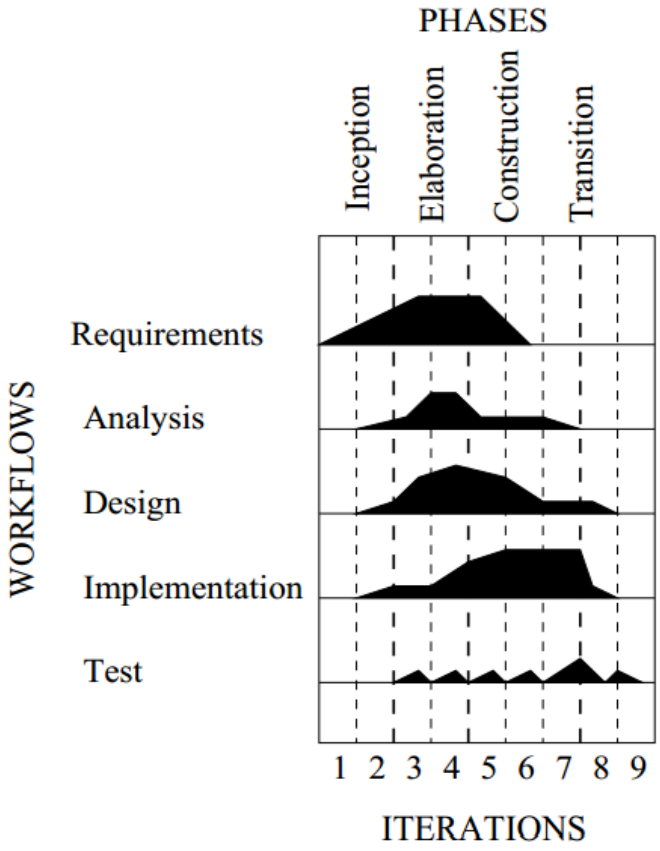

Figura 1: Fases del proceso unificado

Fases del proceso unificado:

1. Inicio: Visión aproximada, análisis del negocio, alcance, estimaciones imprecisas. Al final de esta fase se debe haber realizado un caso de negocio tomando en cuenta la viabilidad del proyecto evaluado y el alcance del diseño debe ser conocido.
2. Elaboración: Esta fase nos lleva a la especificación del sistema. Al final de esta fase se tiene una arquitectura del sistema básica, un plan de construcción y se han identificado todos los riesgos significativos.
3. Construcción: En esta fase se lleva a cabo la implementación del sistema. Al final de esta fase debe existir un prototipo del sistema trabajando, lo suficiente para realizar pruebas preliminares en condiciones realistas.
4. Transición: En esta fase se realizan pruebas de integración y de despliegue.

Dentro de estas fases se pueden pasar por un número de iteraciones, cada una involucrando las actividades de trabajo: requisitos, especificaciones, análisis, diseño, implementación y pruebas.

4. Desarrollo del proyecto

A continuación se explica cómo se llevó a cabo el desarrollo del *sistema de detección de plagio en archivos de texto* presentando el diseño del sistema detallando los cuatro módulos que lo componen. Además, se presentan los detalles de la implementación del sistema y se expone la tecnología empleada y su justificación.

4.1. Diseño

A continuación se presentan los artefactos de diseño siguientes: casos de uso, la arquitectura del sistema, el diseño de la base de datos y el diagrama de clases; explicando con detalle la especificación de cada módulo del sistema.

4.1.1. Módulos del sistema

Los cuatro módulos que conforman el sistema son: módulo 1, encargado de la detección de similitud estructural a través del número de palabras y párrafos de dos archivos de texto; módulo 2, encargado de procesar cada uno de los archivos de texto que se recibirán como entrada con técnicas de procesamiento de lenguaje natural; módulo 3, encargado de la detección de similitud del texto en base a la comparación de trigramas, huellas digitales y subcadenas comunes más largas; y módulo 4, encargado de presentar las funcionalidades del sistema, recuperar los datos de entrada y devolver los resultados al usuario.

4.1.2. Diagrama de casos de uso general

En la figura 2, se muestran el diagrama general de casos de uso del sistema en el cual podemos observar los cuatro módulos que lo conforman.

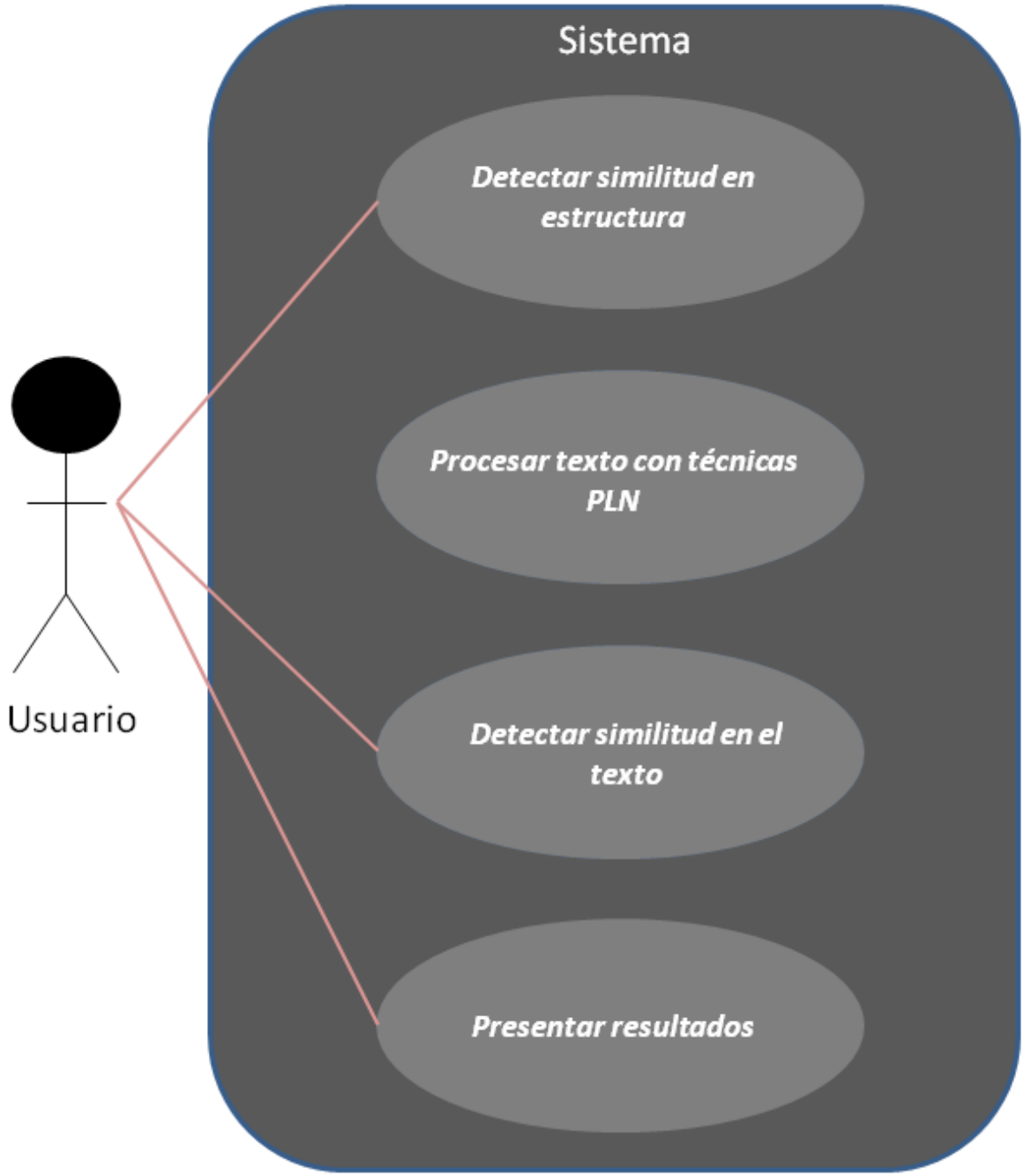

Figura 2: Casos de uso general

Cada caso de uso general tiene, a su vez, casos de uso específicos que a continuación se presentan:

4.1.3. Caso de uso: Detectar similitud en estructura

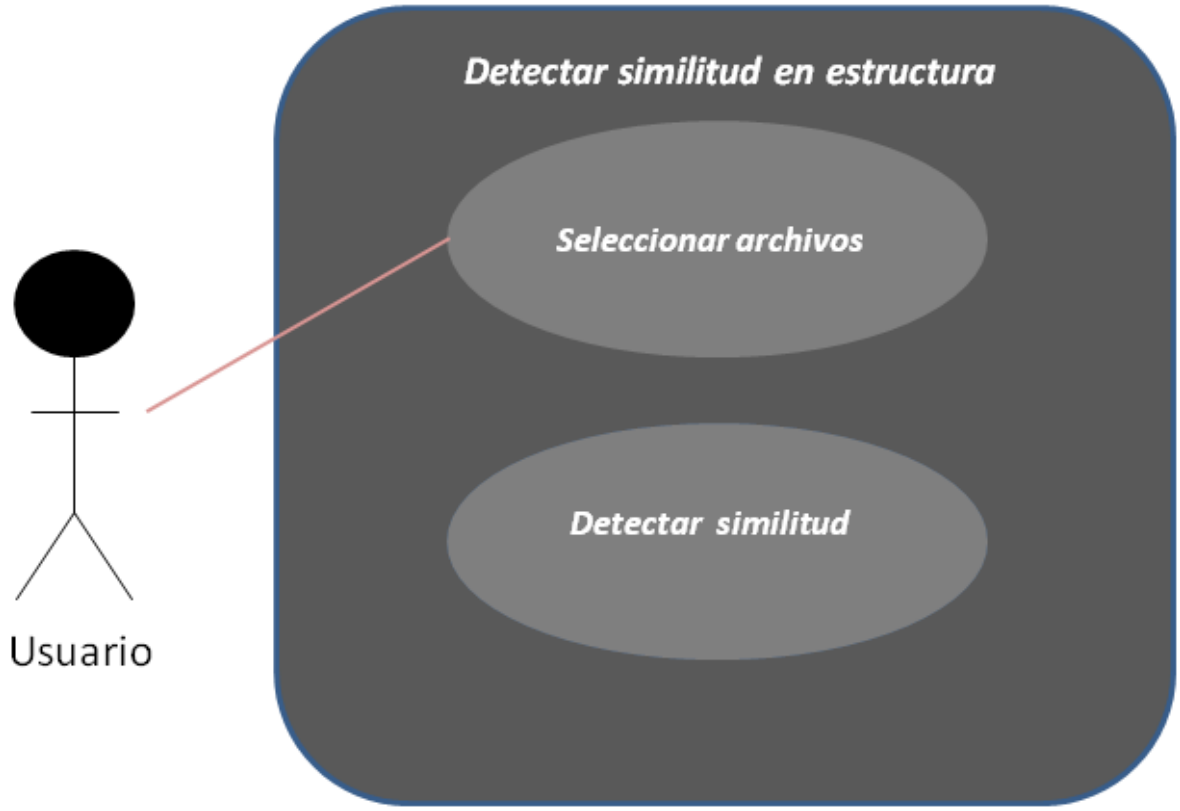

Figura 3: Caso de uso Detectar similitud en estructura

En la figura 3 se muestra el caso de uso: *Detectar similitud en estructura*, así como los casos de uso específicos que lo conforman:

- *Seleccionar archivos*: Se deben seleccionar los archivos de texto a comparar, los cuales deben tener la extensión .doc, .docx, .odt ó .txt. Además se seleccionarán los parámetros para los algoritmos de detección de similitud en huella digital y subcadenas comunes más largas.
- *Detectar similitud*: Se encarga de la comparación de la estructura de todos los posibles pares de archivos seleccionados en base al número de palabras y párrafos para determinar si se procesan ó no los archivos.

EL caso de uso *Detectar similitud en estructura* representa el módulo 1 del sistema. El módulo 1 interactúa con el usuario para la selección de archivos y para la definición de los parámetros de detección de similitud. Posteriormente este módulo se encarga de la

comparación de la estructura de los pares de archivos seleccionados, la cual se realiza de la siguiente manera:

Se cuenta el número de párrafos que tiene cada uno de los documentos d_1 y d_2; el número de párrafos lo representamos por *PA1* y *PA2* respectivamente; también se cuenta el número de palabras de los documentos d_1 y d_2 siendo *WA1* y *WA2* respectivamente; de los cuales se obtiene la diferencia absoluta del número de párrafos (*DAP*); y la diferencia absoluta del número de palabras (*DAW*).

$$DAP = |PA1 - PA2| \qquad DAW = |WA1 - WA2|$$

Una vez obtenidos el *DAP* y el *DAW* se procede a obtener una medida de similitud entre el número de párrafos (*SP*); y una medida de similitud entre el número de palabras (*SW*), de la siguiente forma:

$$SP = 1 - \frac{DAP}{MAX\{PA1, PA2\}} \qquad SW = 1 - \frac{DAW}{MAX\{WA1, WA2\}}$$

$$Donde: \quad 0 \leq SP, SW \leq 1$$
$$Siendo\ 0\ el\ menor\ y\ 1\ el\ mayor\ grado\ de\ similitud$$

Ya con las medidas *SP* y *SW* se procede a obtener la medida de similitud estructural (*SimilitudE*) de los archivos de texto, de la siguiente forma:

$$SimilitudE = 0.2SP + 0.8SW$$
$$Donde: \quad 0 \leq SimilitudE \leq 1$$
$$Siendo\ 0\ el\ menor\ y\ 1\ el\ mayor\ grado\ de\ similitud$$

El pseudocódigo 1 muestra como se realiza la comparación de la estructura de dos archivos de texto.

```
Función Similitud(Archivos A1, Archivos A2)
    SP ← abs(A1.getNumParrafos()-A2.getNumParrafos())//Se obtiene |PA1-PA2|
    SP ← 1-(SP/Max{A1.getNumParrafos(), A2.getNumParrafos()})//Se calcula SP
    SW ← abs(A1.getNumPalabras()-A2.getNumPalabras())//Se obtiene |WA1-WA2|
    SW ← 1-(SW/Max{A1.getNumPalabras(), A2.getNumPalabras()})//Se calcula SW
    SimilitudE ← (0.2*SP) + (0.8*SW) //Se calculala medida de similitud
    Regresa SimilitudE
Fin
```

Pseudocódigo 1: Similitud

En el anexo A se proporciona el código fuente del método *Similitud* de la clase *SimilitudEstructura*.

4.1.4. Caso de uso: Procesar texto con técnicas PLN[1]

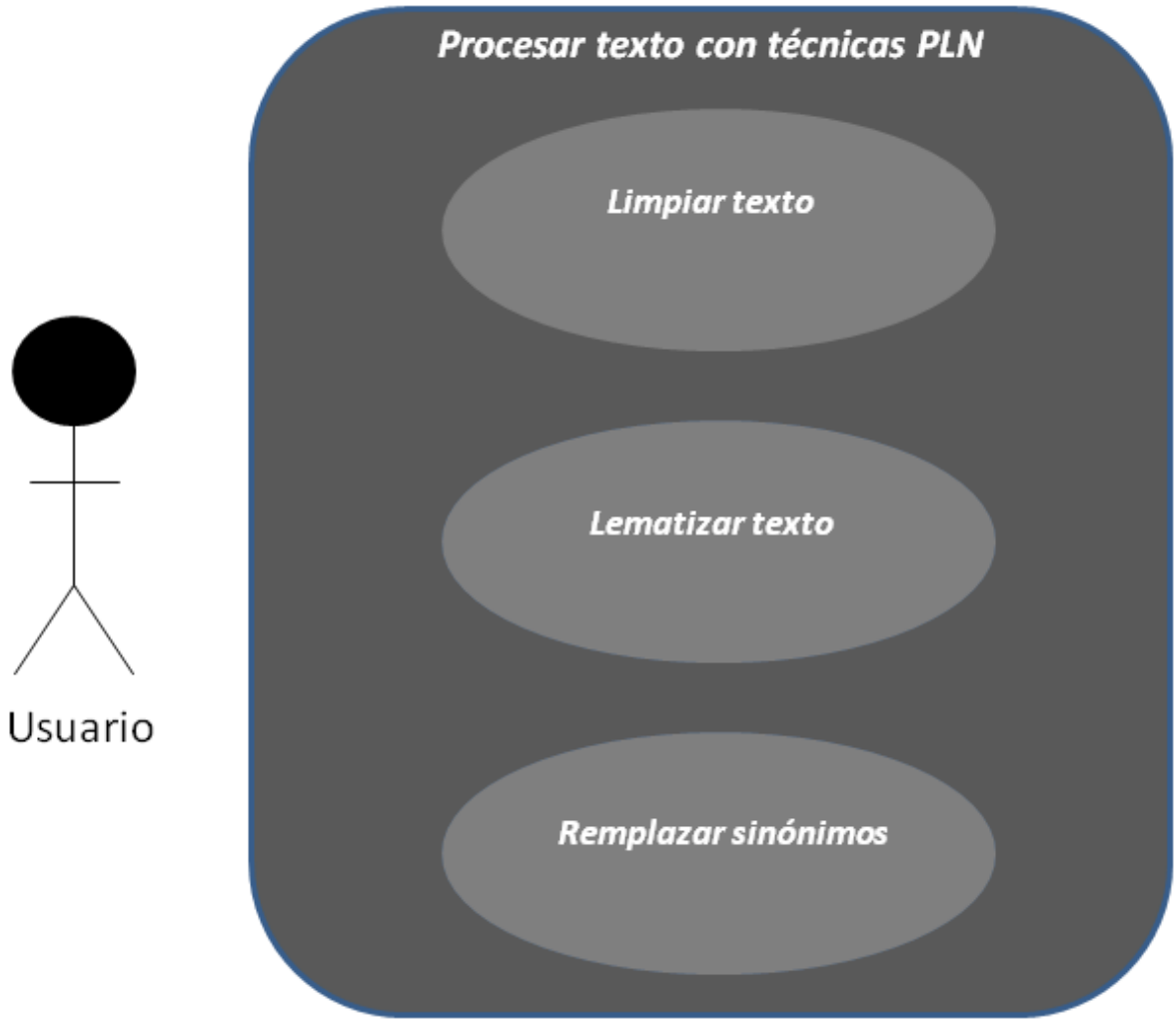

Figura 4: Caso de uso Procesar texto con técnicas PLN

En la figura 4 se muestra el caso de uso: *Procesar texto con técnicas PLN*, así como los casos de uso específicos que lo conforman:

Limpiar texto: Se encarga de aplicar al texto de cada par de archivos a comparar las técnicas PLN de minusculización, remplazo de números y limpiado.

Lematizar texto: Se encarga de aplicar al texto de cada par de archivos a comparar la técnica PLN de lematización.

Remplazar sinónimos: Se encarga de aplicar al texto de cada par de archivos a comparar la técnica PLN de remplazado de sinónimos.

[1] PLN (Procesamiento de lenguaje natural), que se define como una serie de técnicas enfocadas a que las máquinas sean capaces de manejar lenguajes no formales.

El caso de uso *Procesar texto con técnicas PLN* representa el módulo 2 del sistema. El módulo 2 aplica las técnicas PLN a cada par de archivos que obtuvieron una medida de similitud estructural mayor o igual al 0.5. Primero se aplica la técnica de *limpiar texto* a todo el documento seguido de la técnica de *lematizar texto* y por último la técnica de *remplazar sinónimos*.

El pseudocódigo 2 muestra como se procesan los archivos con la técnica de *limpiar texto* la cual consiste en: 1) pasar todo el texto de los archivos a minúsculas, 2) remover todos los caracteres no alfabéticos y 3) remplazar todos los números con la etiqueta "num"; para generalizar la comparación, 4) remplazar las vocales acentuadas por vocales simples.

```
Función procesaArchivo (Archivos A1)
    Para i ← 0 hasta i<A1.getTokens.size()
        A1.setTokens(i, tokens.get(i).toLowerCase()) //texto en minusculas
        //Se eliminan acentos
        A1.setTokens(i, tokens.get(i).replace('á', 'a'))
        A1.setTokens(i, tokens.get(i).replace('é', 'e'))
        A1.setTokens (i, tokens.get(i).replace('í', 'i'))
        A1.setTokens (i, tokens.get(i).replace('ó', 'o'))
        A1.setTokens (i, tokens.get(i).replace('ú', 'u'))
        //Se remplazan datos numéricos con la etiqueta "num"
        Si A1.getTokens(i) es numero
            A1.setTokens(i,"num")
        //Se eliminan cadenas no alfabéticas
        Otro Si A1.getTokens(i) no es alfabético
            A1. getTokens.remove(i)
            i ← i-1
        Fin Si
    Fin Para
Fin
```

Pseudocódigo 2: procesaArchivo

El pseudocódigo 3 muestra como se lleva a cabo la aplicación de la técnica de lematización, la cual consiste en remplazar cada palabra del documento con su respectivo lema, en caso de que éste exista. Los lemas se obtienen mediante el método *consultaLema*(), el cual accede a la base de datos WordNet 3.0[2] para extraer los lemas existentes de cada palabra.

```
Función lematizacion (Archivos A1)
        Para i ← 0 hasta i<A1.getTokens.size()
                Lema ← consultaLema(A1.getTokens(i)) //Consulta WordNet 3.0
                Si Lema es distinto de NULL //Si se encontró lema
                        A1.setTokens(i, Lema)
                Fin Si
        Fin Para
Fin
```

Pseudocódigo 3: lematizacion

El pseudocódigo 4 muestra como se lleva a cabo la aplicación de la técnica de remplazo de sinónimos, la cual consiste en remplazar cada palabra del documento con su respectivo sinónimo, en caso de que éste exista. Los sinónimos se extraen mediante el método *consultaSinonimo*(), el cual accede a la base de datos WordNet 3.0 para extraer los sinónimos existentes de cada palabra.

```
Función sinonimos (Archivos A1)
        Para i ← 0 hasta i<A1.getTokens.size()
                //Consulta WordNet 3.0
                Sinónimo ← consultaSinonimo(A1.getTokens (i))
                Si Sinónimo es distinto de NULL //Si se encontró sinónimo
                        A1.setTokens(i, Sinónimo)
                Fin Si
        Fin Para
Fin
```

Pseudocódigo 4: sinonimos

[2] WordNet es una gran base de datos léxica de sustantivos, verbos, adjetivos y adverbios agrupadas en conjuntos de sinónimos cognitivos y lemas.

4.1.5. Caso de uso: Detectar similitud en el texto

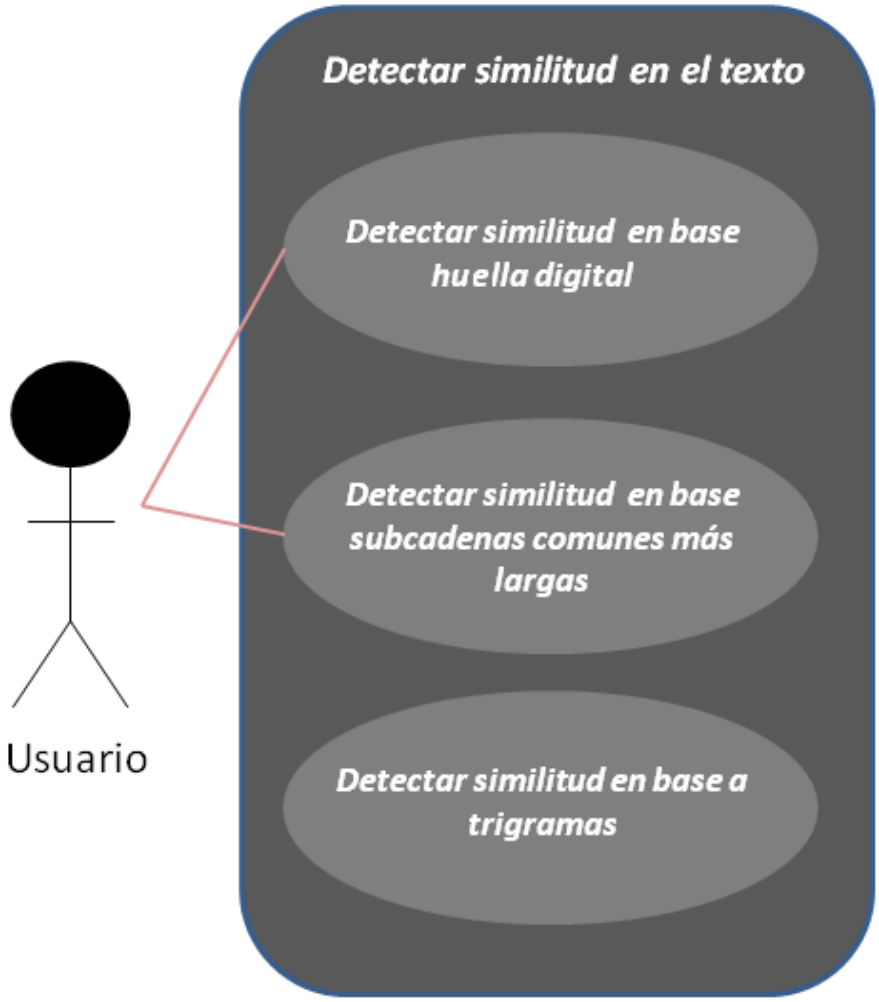

Figura 5: Caso de uso Detectar similitud en el texto

Detectar similitud en base a huella digital: Se encarga de obtener las huellas digitales de cada par de archivos, para compararlas y obtener una medida de similitud.

Detectar similitud en base a subcadenas comunes más largas: Se encarga de obtener las subcadenas comunes más largas de cada par de archivos, obteniendo una medida de similitud.

Detectar similitud en base a trigramas: Se encarga de comparar los trigramas de cada par de archivos, obteniendo una medida de similitud.

El caso de uso *Detectar similitud en el texto* representa el módulo 3 del sistema. El módulo 3 se encarga de detectar la similitud en el texto en base a la comparación de huella digital, cadenas comunes y trigramas en los pares de archivos que obtuvieron una medida de similitud estructural mayor o igual al 0.5.

Algoritmo: Detección de similitud en base a huella digital

El cálculo de similitud de dos archivos en base a huella digital se basa en el algoritmo *Winnowing para hashings de k-gramas* [13], el cual para formar las huellas digitales toma una secuencia de valores *hash* del documento, a la secuencia de valores se le llama *ventana*. Después de generar todas las ventanas posibles de un documento, se selecciona al *hash* mínimo de cada una de esas ventanas. Los valores *hash* seleccionados conformarán la huella digital del documento. El usuario necesitará proporcionar al algoritmo los valores numéricos *t* y *k*. Si hay una cadena repetida de al menos *t* caracteres, el algoritmo garantiza que, de existir esta misma cadena en el otro documento, será detectada; el algoritmo no garantiza detectar coincidencias en cadenas menores a *k caracteres*.

El pseudocódigo 5 muestra el desarrollo del algoritmo para obtener la huella digital de un documento. Cabe mencionar que se utiliza la función *hash Rabin-Karp Rolling* [15].

```
Función HD(Archivos A,int t,int k)
        Para i ← 0 hasta i<A.getTokens().size()
                texto ← texto+A.getTokens().get(i)//Se crea una sola cadena de texto
        Fin Para
        Para j ← 0 hasta j<texto.lenght()/k  //Se generan los hashings de k-gramas
                hashValues[j] ← 0
                Para i ← 0 i<k
                        hashValues[j] ← hashValues[j]+pow((7*texto.charAt(j+i)), k-(i+1))
                Fin Para
        hashValues[j] ← hashValues[j]%999999937
        Fin Para
        //Se toma el hash mínimo de cada ventana de tamaño t-k+1
        Para i ← 0 hasta i<hashValues.length
        aux ← hashValues[i]
                Para j ← 1 hasta j<t-k+1
                        aux ← Min{aux, hashValues[i+j])}
                Fin Para
                Para j ←i hasta j<hashValues.length
                        Si  hashValues[j] es igual a aux
                                i ← j
                                j ← hashValues.length
                Fin Para
        HDvector.add(aux); //Se agrega hash mínimo a la huella digital
        Fin Para
        Regresa HDvector //Retorna la huella digital
Fin
```

Pseudocódigo 5: HD

Después de aplicar el algoritmo HD, la huella digital de un documento queda representada por un conjunto de valores *hash*. Sea d_1 y los documentos a comparar, y sean $W(d_1)$ y $W(d_2)$ las hullas digitales correspondientes; la medida de similitud (*similitudHD*) entre ambas huellas se define como sigue:

$$similitudHD = \frac{|W(d_1) \cap W(d_2)|}{|W(d_1) \cup W(d_2)|} \qquad Donde: 0 \leq similitudHD \leq 1$$

En el pseudocódigo 6 se muestra el algoritmo para obtener la similitud de huella digital de dos documentos.

```
Función comparaHD(Archivos A1, Archivos A2,int t,int k)
        HD1←HD(A1, t, k)//Se genera huella digital para el archivo 1
        HD2←HD(A2, t, k) //Se genera huella digital para el archivo 2
        tamaño ← HD1.size()+HD2.size()
        Para i ← 0 hasta i<HD1.size()
//Se verifica la coincidencia que tienen cada par de valores hash que
//conforman las huellas digitales que se están comparando
                Para j ←0 hasta j<HD2.size()
                        Si HD1.get(i)-HD2.get(j) es cero
                                HD.add(HD1.get(i))
                                HD.add(HD2.get(j))
                                HD2.remove(j)
                                HD1.remove(i)
                                j ← HD2.size();
                                i ← i-1
                        Fin Si
                Fin Para
        Fin Para
        //Se calcula la similitud de los archivos
```

Pseudocódigo 6: ComparaHD

En el anexo B se puede observar el código fuente de los métodos *HD* y *comparaHD* de la clase *SimilitudHD*.

Algoritmo: Detección de similitud en base a subcadenas comunes más largas

La detección de similitud en cadenas comunes más largas se basa en el algoritmo *Greedy-String-Tiling* [14], el cual para localizar las cadenas comunes más largas en dos documentos, utiliza el concepto de *match*, que es una asociación temporal de un par de cadenas comunes en los documentos d_1 y d_2. Esta asociación no necesariamente es única ya que una de las dos cadenas podría estar involucrada en más de un *match*. También usa el concepto de *tile*, que es un conjunto de *matches* únicos. Un *match* es único, si la longitud de las cadenas que forman el match es máxima. El desarrollo del algoritmo se descompone en dos etapas:

> Etapa 1: Se itera en los textos buscando los *matches* de mayor longitud no menor a *M*. Donde *M* es una constante proporcionada por el usuario que define el número de palabras mínimas de cada *match.* Un valor mínimo *M* debe ser mayor o igual a 3 palabras para considerarse un buen parámetro para obtener las subcadenas comunes más largas.
>
> Etapa 2: Se marcan todos los *matches* de longitud máxima encontrados en la etapa 1 formando el conjunto *tile*, es decir se marcan las palabras del texto que conforman al *match*, para evitar su uso en *matches* posteriores.
>
> Estas dos etapas se repiten hasta no encontrar más *matches* de longitud mayores a *M*. Los *matches* se conformarán por una tripleta *match(a, b, l)* donde *a* denota la posición de comienzo de la subcadena en el texto 1, *b* denota la posición de comienzo de la subcadena en el texto 2 y *l* denota la longitud de la cadena. De la misma manera el conjunto *tile* se conformará por tripletas *match (a, b, l).*

Una vez obtenido el conjunto *tile*, se puede obtener una medida de similitud *similitudSCL* en base a la sumatoria de la longitud de los *matches* que lo conforman, la cantidad de palabras en el texto *T1* y en el texto *T2* como sigue:

$$similitudSCL = \frac{2\sum_{match(a,b,l)\epsilon\ tiles} l}{T1 + T2} \qquad Donde: 0 \leq similitudSCL \leq 1$$

$$Siendo\ 0\ el\ menor\ y\ 1\ el\ máximo\ grado\ de\ similitud$$

En el pseudocódigo 7 se muestra el algoritmo para obtener la similitud en cadenas comunes de dos documentos.

```
Función comparaSCL(Archivo A, Archivo B,int M)
        tiles ← {}//Arreglo contenedor de las cadenas comunes
        Haz
            maxmatch ← M//Tamaño mínimo de cadena
            matches ← {}////Arreglo contenedor de las cadenas comunes
            Para todos los tokens sin marcar Aa en A //Se itera sobre el texto en el archivo A
                Para todos los tokens sin marcar Bb in B //Se itera sobre el texto en el archivo B
                    j ← 0
                    //Se detectan los matches
                    Mientras (Aa+j == Bb+j && SinMarcar(Aa+j) && SinMarcar (Bb+j))
                            j ← j+1
                             Si j == maxmatch //Se valida tamaño mínimo de match
                                matches ← matches ⊕ match(a, b, j) //se agregan matches encontrados
                             Otro Si j >maxmatch //Se encontró match de longitud mayor a tamaño mínimo
                                matches ← {match(a, b, j)} //se agrega match eliminanndo los anteriores
                                maxmatch ← j //Se define nuevo tamaño mínimo de match
                             Fin Si
                    Fin Mientras
                Fin Para
                //Se marca el texto de los matches de máxima longitud
                Para todos los match(a, b, maxmatch) ∈ matches
                    Para j ←0 hasta maxmatch− 1
                         marca(Aa+j)
                        marca(Bb+j )
                    Fin Para
                    //Se agregan los matches encontrados al conjunto tile
                    tiles ←tiles ∪ match(a, b, maxmatch)
                Fin Para
          Mientras (maxmatch > M)//Se itera hasta no encontrar matches de longitud mayor a M
          Regresa tiles
Fin
```

Pseudocódigo 7: comparaSCL

En el anexo C se puede observar el código fuente del método *comparaSCL* de la clase *SimilitudSCL*.

Algoritmo: Detección de similitud en base a trigramas

La detección de similitud en trigramas [12], consiste en formar todos los trigramas distintos de cada documento, detectando las coincidencias de los trigramas. Una vez formados los trigramas distintos para los documentos d_1 y d_2 que se representarán como el conjunto A y B respectivamente, se comparan para obtener una medida de similitud (*similitudT*) de la siguiente forma:

$$similitudT = \frac{Número\ Trigramas\ en\ comun}{Número\ total\ de\ Trigramas} = \frac{A \cap B}{A \cup B} \qquad Donde: 0 \leq similitudT \leq 1$$

En el pseudocódigo 8 se muestra el algoritmo para obtener la similitud en trigramas de dos documentos.

```
Función comparaT(Archivos A1, Archivos A2)
    aux[0] ← A1.getTokens(0)//Primer palabra del trigrama
    aux[1] ← A1.getTokens(1)//Segunda palabra del trigrama
    aux[2] ← A1.getTokens(2)//Tercera palabra del trigrama
    aux[3] ← "0"//Posición del trigrama en el archivo 1, i= inexistente
    aux[4] ← "i"//Posición del trigrama en el archivo 2, i= inexistente
    trigrama.add(aux)
    //Se generan todos los trigramas distintos del archivo 1
    Para i ← 1 hasta i < A1.getToken().size() – 2
        aux[0] ← A1.getTokens(i)
        aux[1] ← A1.getTokens(i + 1)
        aux[2] ← A1.getTokens(i + 2)
        aux[3] ← "" + i
        aux[4] ← "i"
        index ← binarySearch(trigrama, aux)//Se verifica existencia de trigrama
        Si index >= 0//Si existe se agrega la posición de nueva aparición
            aux[0] ← trigrama.get(index)[0]
            aux[1] ← trigrama.get(index)[1]
            aux[2] ← trigrama.get(index)[2]
            aux[3] ← trigrama.get(index)[3] + ":" + i
            aux[4] ← trigrama.get(index)[4]
            trigrama.set(index, aux)
        Otro //De lo contario se agrega el nuevo trigrama
            trigrama.add(aux)
            Sort(trigrama)//Se ordenan alfabéticamente los trigramas
        Fin Si
    Fin Para
    //Se generan todos los trigramas distintos del archivo 2
    Para  i ← 0 hasta i < A2.getTokens().size() – 2
        aux[0] ← A2.getTokens(i)
        aux[1] ← A2.getTokens(i + 1)
        aux[2] ← A2.getTokens(i + 2)
        aux[3] ← "i"
        aux[4] ← "" + i
        index ← binarySearch(trigrama, aux); //Se verifica existencia de trigrama
        Si  index >= 0 //Si existe se agrega la posición de nueva aparición
```

```
                    aux[0] ← trigrama.get(index)[0]
                    aux[1] ← trigrama.get(index)[1]
                    aux[2] ← trigrama.get(index)[2]
                    aux[3] ← trigrama.get(index)[3]
                    Si  trigrama.get(index)[4].equalsIgnoreCase("i")
                            aux[4] ← "" + i
                    Otro
                            aux[4] ← trigrama.get(index)[4] + ":" + i
                    Fin Si
                    trigrama.set(index, aux)
            Otro
                    trigrama.add(aux)
                    Sort(trigrama)
            Fin Si
    Fin Para
    similitudT ← 0
    //Se cuenta el número de trigramas en común
    Para i ←0 hasta i < trigrama.size()
            Si  ( !trigrama.get(i)[3].equalsIgnoreCase("i")  &&
                  !trigrama.get(i)[4].equalsIgnoreCase("i") )
                        similitudT++
            Fin Si
    Fin Para
    similitudT ← similitudT / trigrama.size();//Se calcula similitud de archivos
    //Solo se mantienen los trigramas en común
     Para i ←0 hasta i < trigrama.size()
            Si ( trigrama.get(i)[3].equalsIgnoreCase("i")  ||
                 trigrama.get(i)[4].equalsIgnoreCase("i") )
                        trigrama.remove(i)
                        i ← i-1
            Fin Si
    Fin Para
Fin
```

Pseudocódigo 8: comparaT

En el anexo D se puede observar el código fuente del método *comparaT* de la clase *SimilitudT*.

4.1.6. Caso de uso: Presentar Resultados

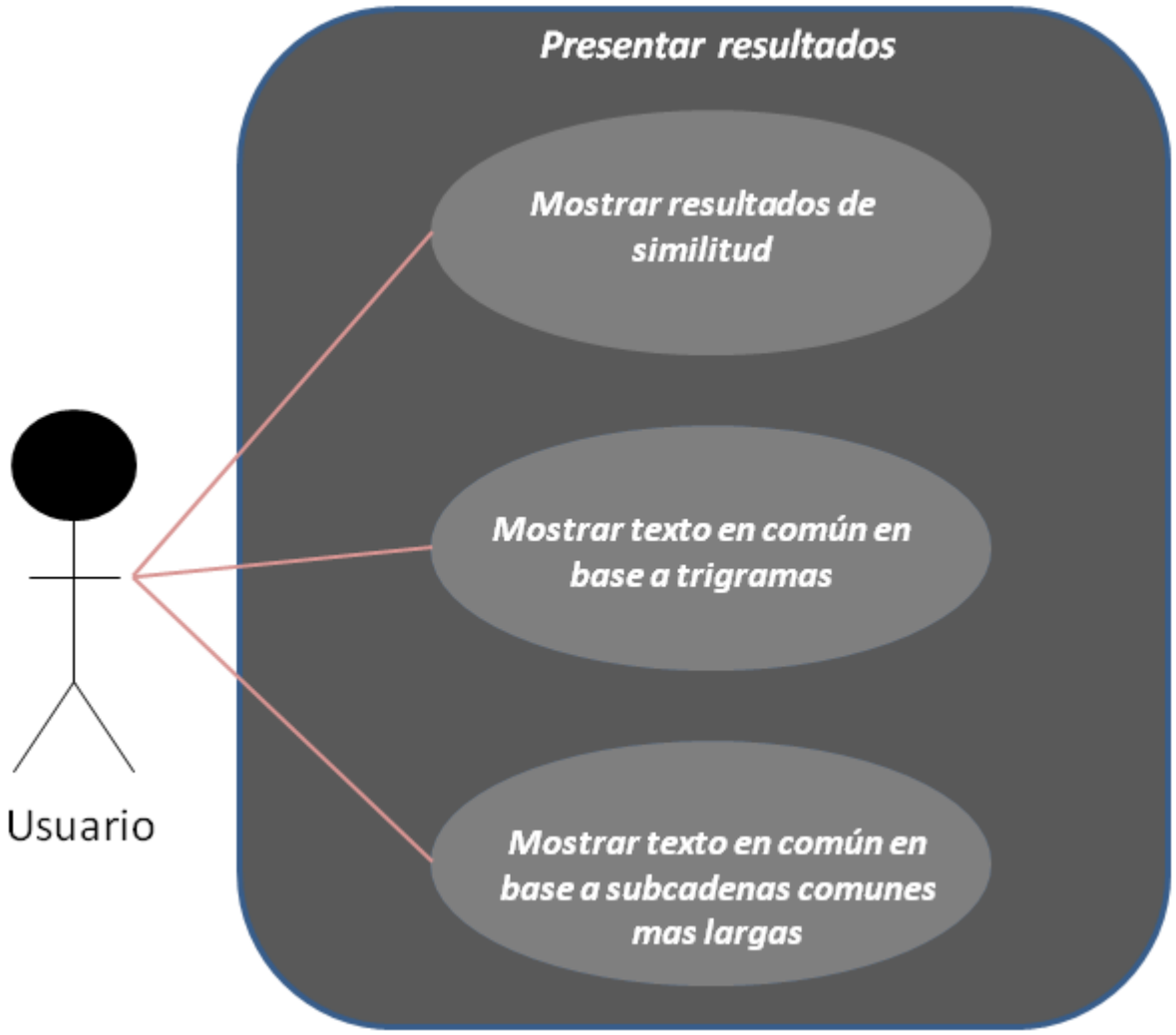

Figura 6: Caso de uso Presentar resultados

El caso de uso *Presentar resultados* representa el módulo 4 del sistema. Este módulo se encarga de la comunicación entre el usuario y el sistema. Se encarga de presentar las funcionalidades del sistema, recuperar los datos de entrada y devolver los resultados al usuario.

Mostrar resultados de similitud: Se encarga de mostrar una lista de los pares de archivos comparados mostrando las medidas de similitud obtenidas.

Mostrar texto en común en base a trigramas: Se encarga de mostrar el texto en común en base a trigramas del par de archivos seleccionado por el usuario realizando un marcado del texto.

Mostrar texto en común en base a subcadenas comunes más largas: Se encarga de mostrar el texto en común en base a subcadenas comunes del par de archivos seleccionado por el usuario, realizando un marcado del texto.

4.1.7. Arquitectura del sistema

La arquitectura del sistema es de tipo *standalone;* es decir, opera en una sola computadora; y el diseño del sistema está basado en el patrón Modelo-Controlador-Vista; esto nos garantiza un bajo acoplamiento entre los elementos que lo conforman. La figura 7 muestra un esquema de la arquitectura.

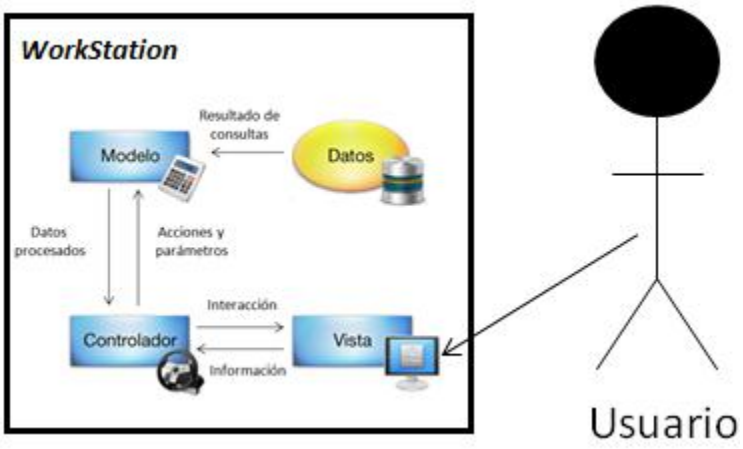

Figura 7: Arquitectura del sistema

4.1.8. Clases del Sistema

A continuación se presentan las clases del sistema las cuales siguen el patrón de diseño Modelo-Controlador-Vista.

Modelo

El modelo es la representación específica de la información con la cual el sistema opera, es decir, define las reglas de negocio. Las clases del modelo son:

La clase *Archivo*, ver figura 8, define los atributos y los métodos para obtener y modificar la información necesaria de los archivos de texto a procesar.

Figura 8: Clase Archivo

La clase *HuellaDigital*, ver figura 9, define los atributos y los métodos para obtener y modificar la información necesaria de la huella digital de un archivo de texto.

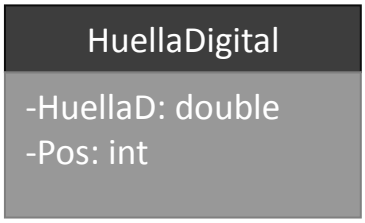

Figura 9: Clase HuellaDigital

La clase *Match*, ver figura 10, define los atributos y los métodos para obtener y modificar la información necesaria para formar las cadenas comunes de cada archivo de texto.

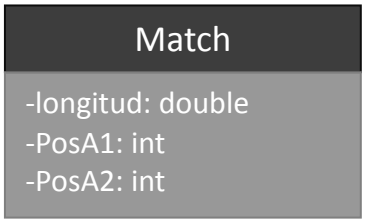

Figura 10: Clase Match

La clase *SCM*, ver figura 11, define los atributos y los métodos para obtener y modificar la información necesaria de cada token para formar las cadenas comunes más largas de cada archivo de texto.

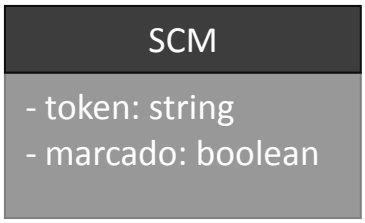

Figura 11: Clase SCM

Controladores

Un controlador es responsable de responder a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y la vista, es decir, contiene las reglas de gestión de eventos. Las clases controladoras son:

La clase *SimilitudEstructura*, ver figura 12, contiene el método para realizar la detección de similitud en estructura de los archivos de texto a procesar. Además, contiene el atributo *similitudE* para representar numéricamente la similitud entre los archivos analizados.

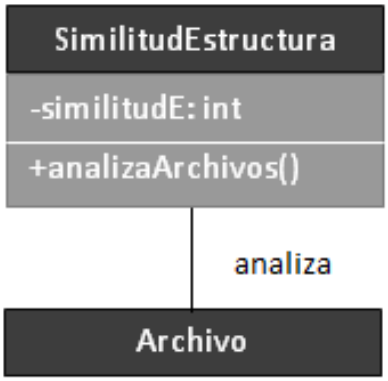

Figura 12: Clase Archivo

La clase *TecnicasPLN*, ver figura 13, contiene el método que se encarga del procesamiento de los archivos de texto de entrada con técnicas PLN de tal forma que los archivos queden listos para la detección de similitud en el texto.

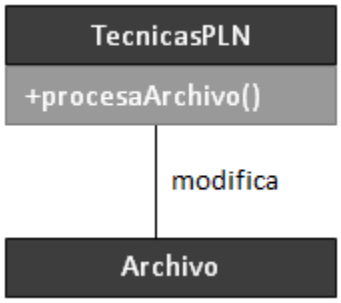

Figura 13: Clase TecnicasPLN

La clase *SimilitudHD*, ver figura 14, contiene el método que se encarga de la detección de similitud en el texto en base a la comparación de huellas digitales, a saber: *comparaHD()*. Además, contiene los atributos *similitudHD* para representar numéricamente el resultado obtenido, y *HD* que representa la huella digital en común de cada par de archivos comparados.

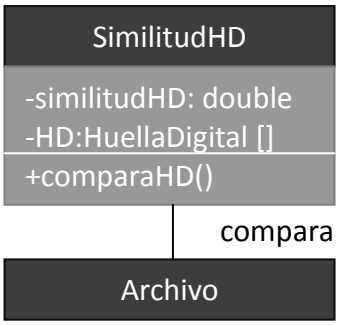

Figura 14: Clase SimilitudHD

La clase *SimilitudSCL*, ver figura 15, contiene el método que se encarga de la detección de similitud en el texto en base a la comparación de subcadenas comunes, a saber: *comparaSCL()*. Además, contiene los atributos *similitudSCL* para representar numéricamente el resultado obtenido, y *tile* que representa las cadenas en común de cada par de archivos comparados.

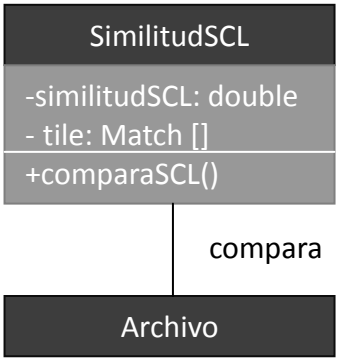

Figura 15: Clase SimilitudSCL

La clase *SimilitudT*, ver figura 16, contiene el método que se encarga de la detección de similitud en el texto en base a la comparación trigramas, a saber: *comparaT()*. Además, contiene los atributos *similitudT* para representar numéricamente el resultado obtenido, y *trigrama* para representar los trigramas en común de los archivos comparados.

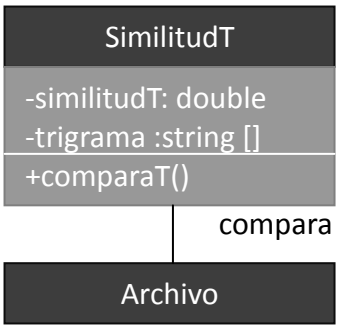

Figura 16: Clase SimilitudT

La clase *Resultados*, ver figura 17, contiene los métodos para presentar los resultados obtenidos de la comparación de los archivos. Esta clase muestra el texto en común de los archivos ya sea en base a las subcadenas comunes más largas o en base a los trigramas en común.

Figura 17: Clase Resultados

Vista

La vista presenta el modelo en un formato adecuado para interactuar con el usuario mediante la interfaz grafica. La interfaz gráfica está definida por las siguientes clases:

La clase *ResultadosG*, ver figura 18, contiene el método que se encarga de controlar la interfaz grafica del usuario, donde se muestran los resultados del análisis.

Figura 18: ResultadosG

La clase *Principal*, ver figura 19, lleva el control general del sistema. Además aquí se realiza la selección de los archivos a analizar y la detección de similitud en estructura.

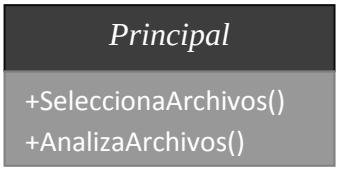

Figura 19: Principal

4.1.9. Diagrama de paquetes

La figura 20 muestra el diagrama general de paquetes con la distribución de las clases correspondientes a cada uno de los elementos del patrón Modelo-Controlador-Vista.

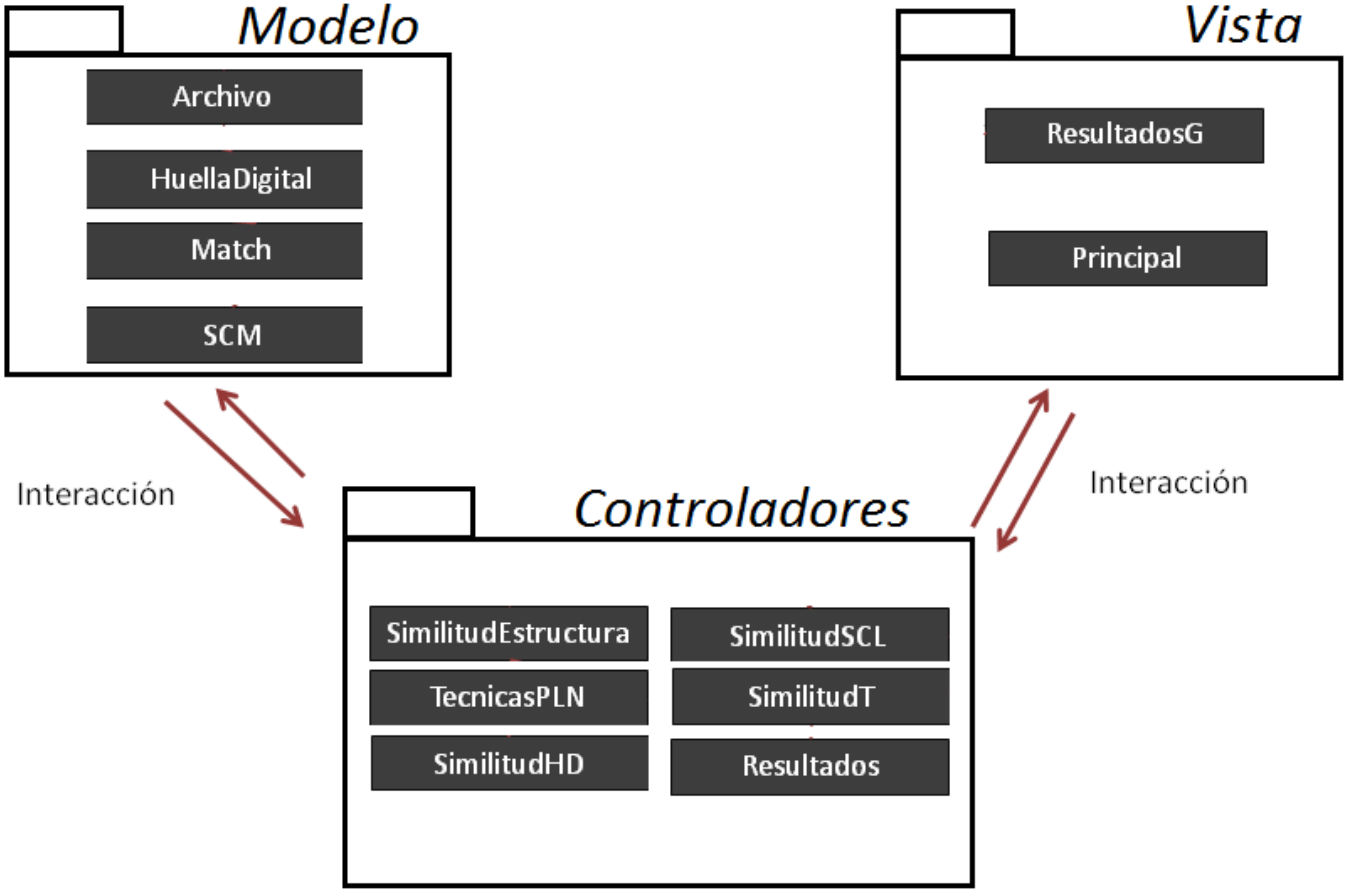

Figura 20: Diagrama de paquetes

4.1.10. Diagrama general de clases

La figura 21 muestra el diagrama general de clases, las relaciones y la cardinalidad entre las clases que conforman el sistema.

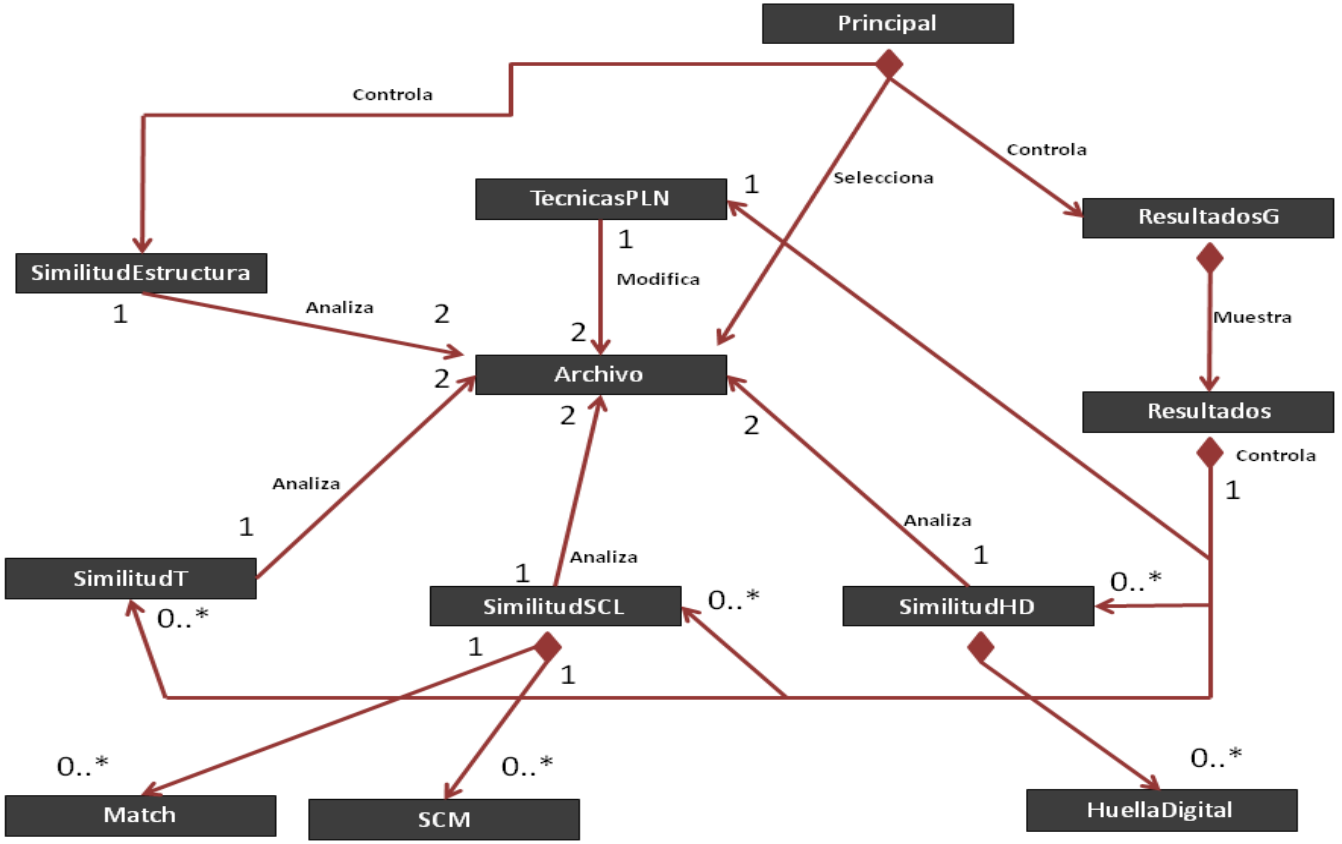

Figura 21: Diagrama de clases

4.1.11. Estructura de la Base de Datos

En la figura 22 podemos observar la estructura de la base de datos WordNet, de la cual se extraen los lemas y sinónimos necesarios en el preprocesamiento del texto de los documentos.

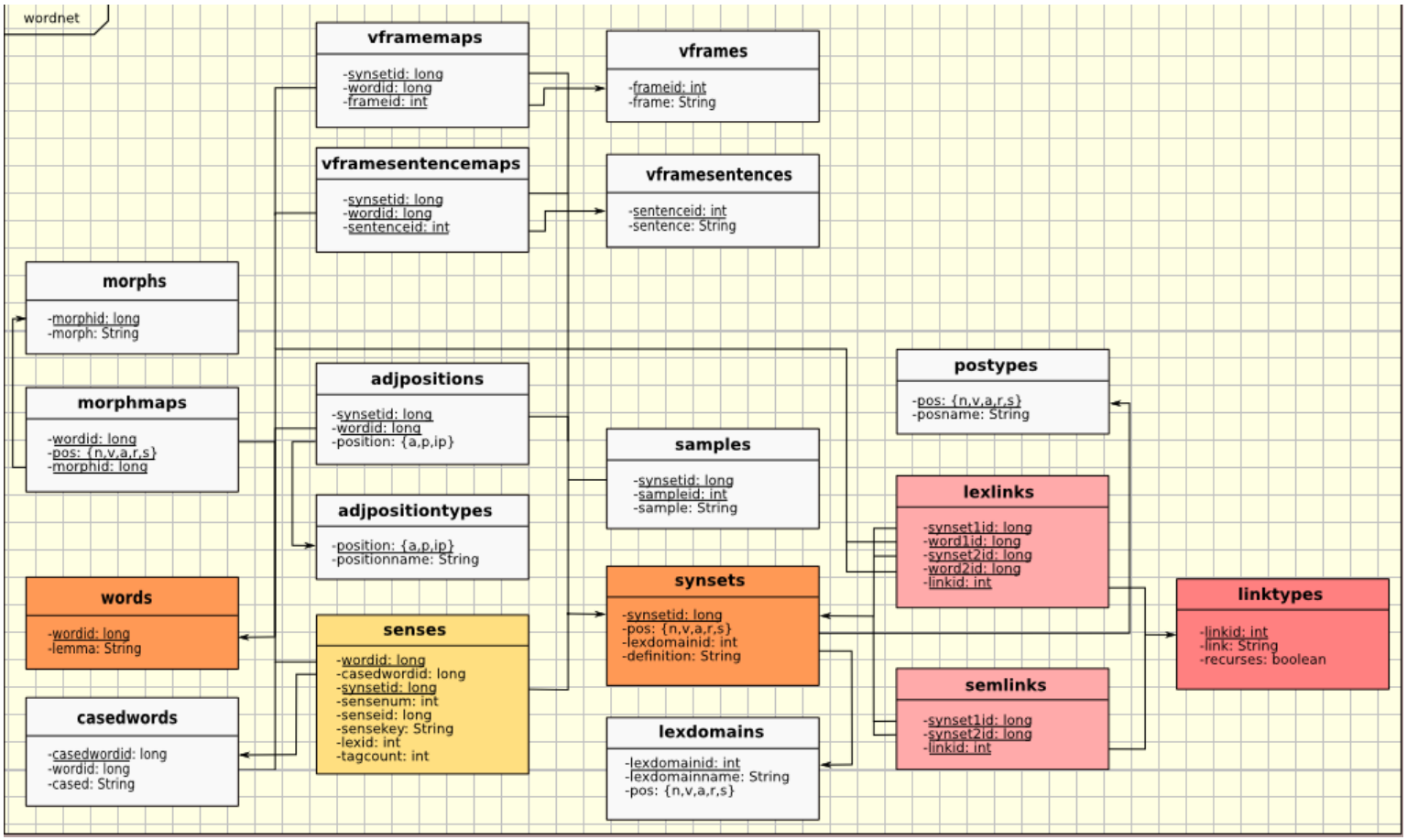

Figura 22: Esquema de WordNet [9]

Es importante mencionar que no se emplean todas las tablas de la base WordNet, ya que la aplicación no requiere todas las propiedades de las palabras que esta base proporciona. Concretamente, el módulo 2 del sistema emplea las tablas: *words* y *synsets,* las cuales contienen un conjunto de lemas y un conjunto de sinónimos respectivamente.

4.2. Implementación

4.2.1. Tecnología Empleada

Para la realización del proyecto se utilizó el siguiente hardware:

- HP pavilion dv4-2145dx entertainment notebook pc con procesador AMD Turion(tm) II Dual-Core Mobile M520 2.30GHz, memoria Ram de 4.00 gigabytes y disco duro de 320 gigabytes.

El software utilizado para el desarrollo del proyecto es de código abierto, evitando así problemas de licenciamiento:

- Sistema operativo Debian GNU/Linux.
- NetBeans IDE 7.2: Entorno de desarrollo integrado de código abierto [9].
- MySQL: Manejador de bases de datos de código abierto [10].

Debian GNU/Linux es un sistema operativo estable, y es de libre distribución.

NetBeans es un entorno de desarrollo integrado, fácil de usar, cómodo y de excelente calidad y es de libre distribución. Además cuanta con herramientas de depuración que facilita el desarrollo de software.

MySQL es un sistema gestor de bases de datos estable, y cuenta con suficiente documentación en su página Web oficial; además, la versión de WordNet utilizada está orientada a operar en MySQL.

Se utilizó Apache POI[3] en su versión 3.8, ya que es la versión más estable hasta el momento; además se cuenta con documentación disponible en la página oficial de Apache. POI es una API de java para la manipulación de documentos de Microsoft. Lo cual se aprovechó para extraer el texto de documentos de Microsoft Word.

También se utilizó JDOM[4] en su versión 2.0.4, ya que es la versión más estable hasta el momento; además se cuenta con documentación disponible en su página oficial. JDOM es

[3] Apache POI es una API para manipular distintos formatos de archivo basados en el formato de Microsoft OLE2.
[4] JDOM es una API basada en Java para acceder, manipular y generar datos en XML desde el código Java.

una API de java para acceder, manipular y crear documentos XML. Lo cual fue utilizado para extraer el texto de documentos generados con OpenOffice.

Finalmente se utilizó MCR WordNet 3.0 como base de palabras del idioma español, cuya versión está disponible gratuitamente, además cuenta con una versión para MySQL.

4.2.2. Funcionalidades

Antes de abordar en detalle cada funcionalidad del sistema de detección de plagio, explicaremos el proceso que sigue el sistema, a partir de que un usuario le proporciona los archivos.

1. Para cada par de archivos se utiliza la clase *SimilitudEstructura* para realizar la comparación estructural de los archivos. Si la medida de similitud estructural es mayor o igual a 0.5, entonces se prosigue con el paso 2; de lo contrario, se prosigue en el paso 6.
2. El sistema llama al método *procesaArchivo* de la clase *TecnicasPLN*, para procesar el texto de cada archivo con técnicas PLN. El sistema pasa como argumento al método *procesaArchivo* cada uno de los documentos a analizar.
3. Una vez procesado el texto de los documentos el sistema llama al método *comparaHD* de la clase *SimilitudHD*, para realizar la detección de similitud en base a huella digital. El sistema pasa como argumentos los dos archivos a analizar, *t* y *k* al método *comparaH*.
4. Ya concluida la detección de similitud en base a huella digital el sistema llama al método *comparaSCL* de la clase *SimilitudSCL*, para realizar la detección de similitud en base a cadenas comunes. El sistema pasa como argumentos los dos archivos a analizar y *M* al método *comparaSCL* .
5. Una vez concluida la detección de similitud en base a cadenas comunes, el sistema llama al método *comparaT* de la clase *SimilitudT*, para realizar la detección de similitud en base a trigramas. El sistema pasa como argumentos los dos archivos a analizar al método *comparaT*.
6. Ya teniendo el análisis de todo los pares posibles de archivos se muestran los resultados obtenidos en la interfaz gráfica. Si el usuario desea ver el texto en común de un par de archivos, sea en base a cadenas comunes o trigramas, el sistema llama al método *textoEnComunSCL* ó al método *textoEnComunT* de la clase *Resultados*. Con estos métodos se realiza el marcado con color amarillo del texto en común para que el usuario lo distinga.

Ingresar al sistema

Para ingresar al sistema, basta con iniciar la aplicación, no es necesario autenticarse con el sistema de detección de plagio. En la pantalla principal (ver figura 23) se puede visualizar el botón de *Selecciona archivos* y una etiqueta que muestra la cantidad de archivos seleccionados.

Al pasar el ratón por encima del botón *Seleccionar archivos* se muestra una leyenda que informa al usuario que se deben seleccionar al menos dos archivos para proceder al análisis de detección de plagio y un máximo de 50. Este valor de 50 archivos se escogió porque la cantidad de veces que se requiere analizar a los pares de archivos tiene un orden cuadrático $O(n^2)$; las pruebas revelaron que al seleccionar como máximo 50 archivos, que no exceden 10 cuartillas, el sistema mantiene un tiempo de respuesta aproximado de 19 minutos.

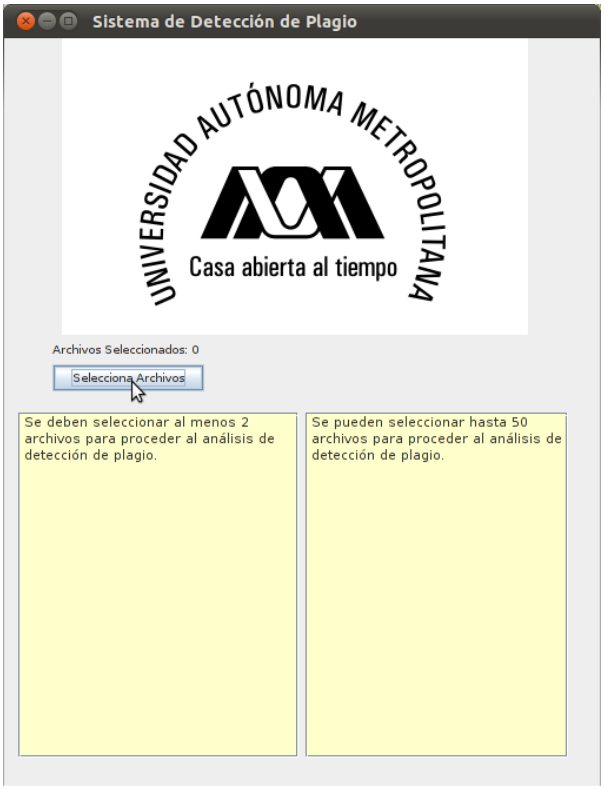

Figura 23: Inicio del sistema

Seleccionar los documentos

Al oprimir el botón de *Seleccionar archivos*, como se observa en la figura 24, se abre una ventana que permite navegar entre los subdirectorios hasta encontrar los archivos a analizar. Es importante mencionar que los archivos pueden existir en distintas carpetas. El sistema filtra los archivos con extensión .doc, .docx, .odt ó .txt.

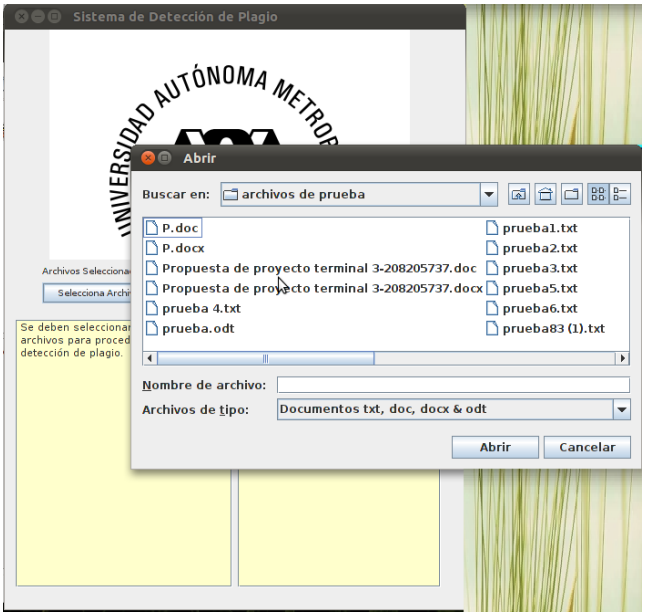

Figura 24: Selección de archivos

Después de haber seleccionado 2 ó más archivos, se visualizan las casillas para ingresar los parámetros numéricos para los algoritmos de detección de similitud *huella digitales* y *cadenas comunes*. Existe una leyenda de ayuda que explica los parámetros que se ingresan para los algoritmos de comparación, la cual describe el significado de cada parámetro y los valores recomendados. Los valores que se recomiendan para huella digital son k=7 y t=27 y para cadenas comunes se recomienda $5 \leq M \leq 9$ ya que en las pruebas realizadas se obtuvo un mejor resultado con estos valores, ver figura 25.

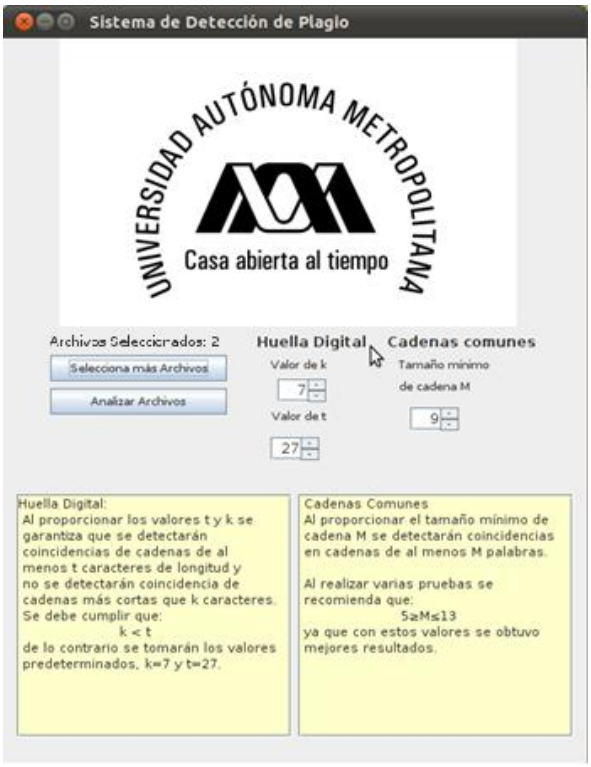

Figura 25: Parámetros de algoritmos

También existe una leyenda de ayuda que da una descripción de los pasos consistentes del análisis de los archivos la cual contiene la información siguiente:

> Se analizarán todos los archivos seleccionados todos contra todos.
> El análisis completo de un par de archivos consiste de cuatro posibles comparaciones:
>
> 1. Se comparan los archivos en base a su estructura, es decir en base a su número de palabras y párrafos . Si la similitud en base a su estructura es mayor al 50% se continúa con las siguientes etapas.
> 2. Se comparan los archivos en base a los trigramas en común.
> 3. Se comparan los archivos en base a su huella digital.
> 4. Se comparan los archivos en base a sus cadenas comunes.

Esta ayuda se muestra al pasar el ratón por encima del botón *Analizar Archivos*, ver figura 26.

Figura 26: Descripción de análisis

Análisis y resultados

Al oprimir el botón de *Analizar Archivos* el sistema procede a analizar por pares todos los archivos seleccionados. Al finalizar el análisis, el sistema muestra la ventana de resultados, ver figura 27, donde se despliega una lista con los nombres de cada par de archivos analizados, los cuales pueden estar marcados en tres colores distintos: blanco, rosa y rojo.

- El color blanco indica que la similitud entre los archivos es nula;
- El color rosa indica que la similitud entre los archivos es baja; y
- El color rojo indica que la similitud entre los archivos es alta.

Al seleccionar un par de archivos específico se muestran los resultados sobre el grado de similitud calculada a través de: la estructura, cadenas comunes, huella digital y trigramas comunes; además, se puede observar el texto en común de los archivos ya sea en base a trigramas o en base a cadenas comunes si es que la similitud estructural fue mayor o igual al 0.5 (ver figura 28). En este caso el texto en común de los archivos se marca de color amarillo; además, se muestra una lista de los trigramas y cadenas en común; al seleccionar una de ellas, se muestra su ubicación en el texto marcándolas de color verde.

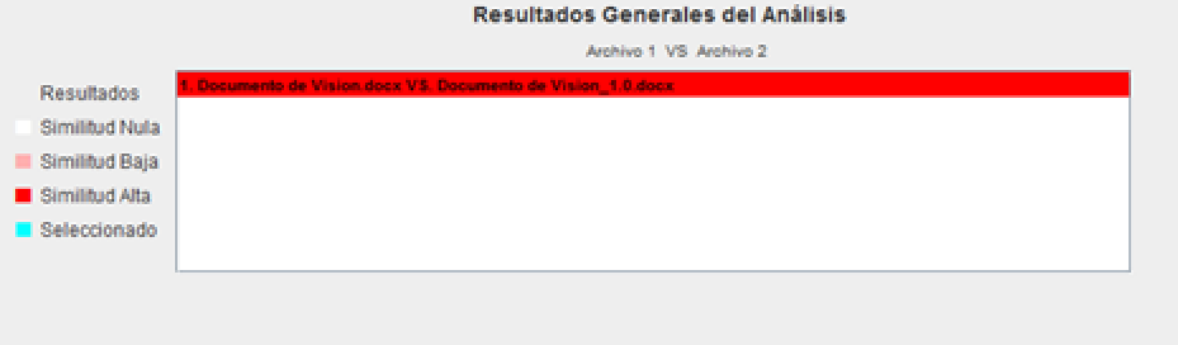

Figura 27: Resultados generales

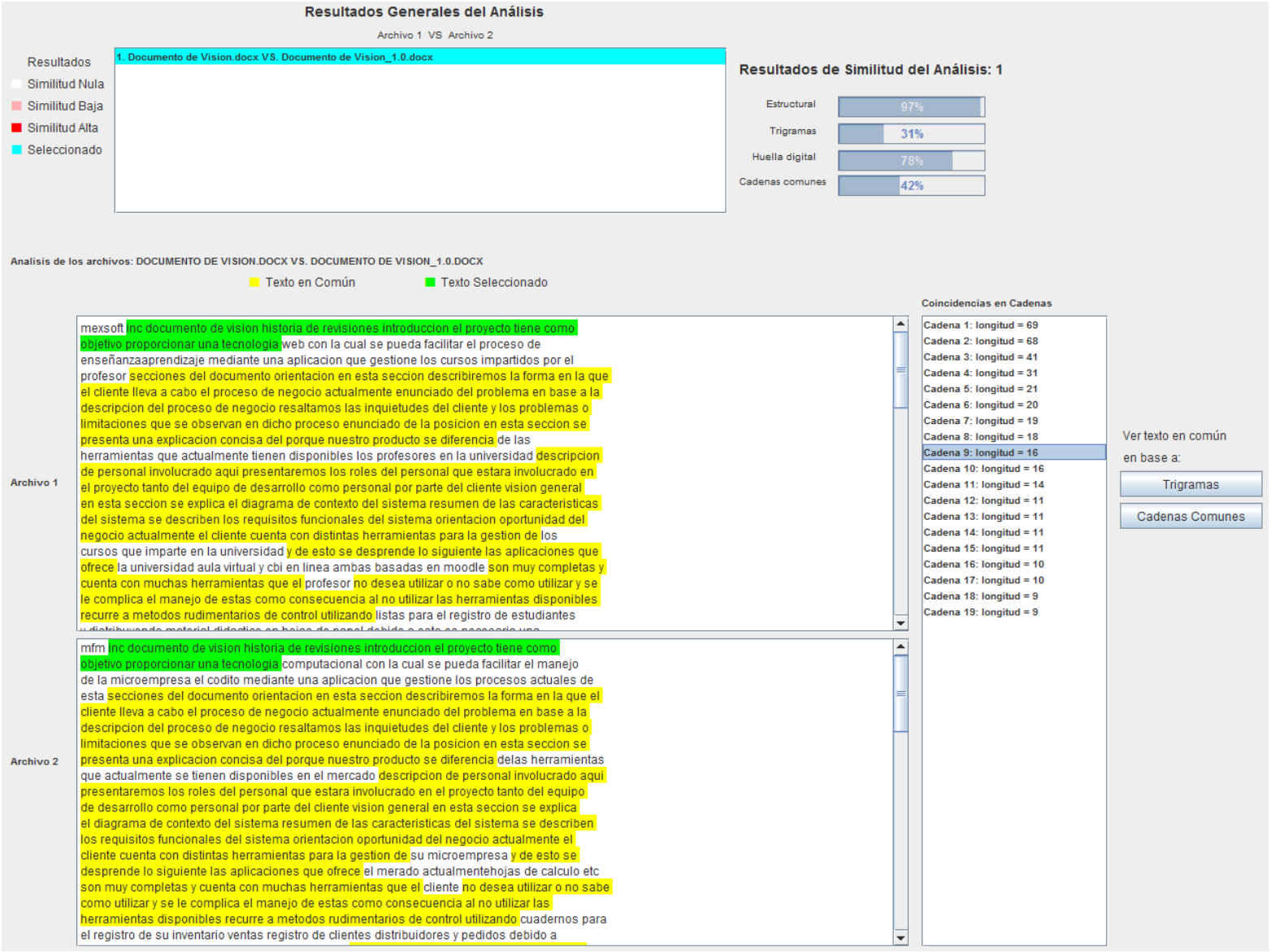

Figura 28: Resultados específicos

En el anexo E se muestra el contenido de los dos archivos analizados con los que se realizaron las pruebas y cuyos resultados del análisis se muestran en las figuras 28 y 29.

5. Conclusiones y perspectivas del proyecto

El objetivo del proyecto terminal fue desarrollar un *sistema de detección de plagio en archivos de texto*; este sistema analiza pares de archivos y proporciona como resultado cuatro medidas de similitud en base a: estructura, huella digital, cadenas y trigramas comunes. El sistema está dividido en cuatro módulos: el primer módulo se encarga de seleccionar los documentos a analizar y detectar su similitud a nivel de su estructura. El segundo módulo se encarga de procesar el texto de los documentos con técnicas de procesamiento de lenguaje natural. El tercer módulo se encarga de detectar la similitud en el texto en base a tres comparaciones: por trigramas, por huella digital y por subcadenas comunes más largas. El cuarto módulo tiene como tarea presentar al usuario los resultados del análisis así como el texto en común de los archivos analizados, a través de una interfaz gráfica.

Los objetivos particulares planteados al inicio del proyecto se cumplieron totalmente, estos son:

- Diseñar e implementar un módulo que permita decidir si es posible que exista similitud entre dos archivos de texto por medio de su estructura.
- Diseñar e implementar un módulo que aplique técnicas de procesamiento de lenguaje natural sobre los archivos de texto a comparar.
- Diseñar e implementar un módulo que realice la comparación de trigramas, huellas digitales y subcadenas comunes más largas de dos archivos de texto.
- Diseñar una interfaz grafica que muestre los resultados, a saber: las medidas de similitud y el texto en común entre los archivos comparados.

El sistema realizado puede hacer lo siguiente:

- Recuperar el texto de archivos con extensión .doc, .docx, .odt y .txt.
- Pueden analizar hasta 50 archivos por pares. Esto implica: siendo n el número de archivos ingresados al sistema se realizarán $\frac{n^2-n}{2}$ análisis de archivos.
- Para cada una de las comparaciones el sistema genera una medida de similitud estructural en base al número de palabras y párrafos de los archivos. Para cada comparación de archivos con una medida de similitud estructural mayor o igual a 0.5:
- Aplica al texto de los archivos las técnicas PLN de *minusculización, remplazo de números, limpiado, lematización* y *remplazo de sinónimos.*

- Genera los trigramas distintos de los dos archivos de texto. Recupera los trigramas en común guardando su ubicación en ambos archivos y generando una medida de similitud.
- Genera la huella digital de los dos archivos de texto recuperando las coincidencias en ellas de las cuales genera una medida de similitud.
- Itera sobre el texto de ambos archivos encontrando cadenas comunes. Guarda la ubicación y longitud de las cadenas; a partir de esta última generará una medida de similitud.
- Presenta un listado con el resultado del análisis al usuario de todos los archivos comparados mostrando las cuatro medidas de similitud generadas.
- Muestra el texto en común de los archivos comparados ya sea en base a trigramas o cadenas en común realizando un marcado del texto en color amarillo.
- Muestra un listado de los trigramas y cadenas comunes de los archivos con la opción de visualizarlos en el texto, es decir marcándolos en el texto al seleccionar un trigrama o una cadena especifica muestra su ubicación marcando el texto en color verde.

Cabe mencionar que en la propuesta del proyecto terminal, se planteó utilizar la base de palabras MCR WordNet 3.0 que es una versión de WordNet en español, como base para la lematización y remplazo de sinónimos en el módulo 2 del sistema. Sin embargo, esto se logró parcialmente, porque MCR WordNet 3.0 en su versión SQL para MySQL contiene solo una parte de la base total. Como resultado, las palabras remplazadas por las técnicas de lematización y remplazo de sinónimos fueron limitadas.

Trabajo a futuro

Al sistema se le pueden agregar módulos para mejorar o ampliar su funcionalidad. Algunos módulos que se podrían agregar son:

- Módulo para enriquecer la base de palabras.
- Módulo para realizar búsquedas de páginas Web para que el sistema determine si existe plagio de ellas en tareas de los alumnos.

Se podría proporcionar una base de palabras que tenga una mayor cantidad de entradas para mejorar el proceso de lematización y remplazado de sinónimos. Además se podría expandir el tipo de archivos que acepta el sistema como por ejemplo archivos con extensión .pdf.

Anexos

Anexo A: Clase SimilitudEstructura

```
public class SimilitudEstructura {
   double similitudE;

   public double getSimilitudE() {
      return similitudE;
   }
   public Archivos analizarArchivo(Archivos A1) throws FileNotFoundException, IOException, Exception {
      String contenido="";
      String aux;
      ArrayList<String> palabrasArchivo;
      int parrafos=0;
      int i,j;
      File f = new File(A1.getUbicacion());
      if(A1.getUbicacion().endsWith("txt")){
         Scanner sc = new Scanner(f,"ISO-8859-1");
         sc.useDelimiter("\n");
         while (sc.hasNext()) {
            if ( (contenido.length()>3) &&(contenido.length()!=0) && contenido.substring(contenido.length()-3,
contenido.length()-2).endsWith("."))parrafos++;
            contenido=contenido+sc.next()+" ";
         }
         if(parrafos==0)parrafos=1;
         sc.close();
         sc = new Scanner(f,"ISO-8859-1");
         palabrasArchivo = new ArrayList<String>();
         while (sc.hasNext()) palabrasArchivo.add(sc.next());
         sc.close();
         A1.setNumParrafos(parrafos);
         A1.setNumPalabras(palabrasArchivo.size());
         A1.setTokensArch(palabrasArchivo);
      }
      else if(A1.getUbicacion().endsWith("doc")){
         File myFile = new File(A1.getUbicacion());
         FileInputStream inputStream = new FileInputStream(myFile);
         POIFSFileSystem fs = new POIFSFileSystem(inputStream);
         HWPFDocument docc = new HWPFDocument(fs);
         WordExtractor we = new WordExtractor(docc);
         contenido = we.getText();
         for(i=2;i<contenido.length();i++) if(contenido.charAt(i)=='\n'&&contenido.charAt(i-2)=='.') parrafos++;
         if(parrafos==0)parrafos=1;
         i=0;
         palabrasArchivo = new ArrayList<String>();
         while(i<contenido.length()){
            j=i+1;
            while(j<contenido.length()&& contenido.charAt(j)!=' ' && contenido.charAt(j)!='\n'&& contenido.charAt(j)!='\t')j++;
```

```
aux=contenido.substring(i, j);
            palabrasArchivo.add(aux);
            i=j+1;
          }
          A1.setTokensArch(palabrasArchivo);
          A1.setNumPalabras(palabrasArchivo.size());
          A1.setNumParrafos(parrafos);
       }
       else if(A1.getUbicacion().endsWith("docx")){
          InputStream in=new FileInputStream(A1.getUbicacion());
          XWPFDocument doc = new XWPFDocument(in);
          XWPFWordExtractor ex = new XWPFWordExtractor(doc);
          contenido = ex.getText();
          for(i=2;i<contenido.length();i++) if(contenido.charAt(i)=='\n'&&contenido.charAt(i-1)=='.' ) parrafos++;
          if(parrafos==0)parrafos=1;
          i=0;
          palabrasArchivo = new ArrayList<String>();
          while(i<contenido.length()){
            j=i+1;
            while(j<contenido.length()&& contenido.charAt(j)!=' ' && contenido.charAt(j)!='\n'&& contenido.charAt(j)!='\t')j++;
            aux=contenido.substring(i, j);
            palabrasArchivo.add(aux);
            i=j+1;
          }
          A1.setTokensArch(palabrasArchivo);
          A1.setNumPalabras(palabrasArchivo.size());
          A1.setNumParrafos(parrafos);
       }
       else if(A1.getUbicacion().endsWith("odt")){
          contenido = new OpenOfficeParser().getTexto(A1.getUbicacion());
          for(i=2;i<contenido.length();i++) if(contenido.charAt(i)=='\n'&&contenido.charAt(i-1)=='.' ) parrafos++;
          if(parrafos==0)parrafos=1;
          i=0;
          palabrasArchivo = new ArrayList<String>();
          while(i<contenido.length()){
            j=i+1;
            while(j<contenido.length()&& contenido.charAt(j)!=' ' && contenido.charAt(j)!='\n'&& contenido.charAt(j)!='\t')j++;
            aux=contenido.substring(i, j);
            palabrasArchivo.add(aux);
            i=j+1;
          }
          A1.setTokensArch(palabrasArchivo);
          A1.setNumPalabras(palabrasArchivo.size());
          A1.setNumParrafos(parrafos);
       }
       return A1;
    }
    public void similitud(Archivos A1,Archivos A2){
       double SP,SW;
       SP=Math.abs(A1.getNumParrafos()-A2.getNumParrafos());
       SP=1-(SP/(Math.max(A1.getNumParrafos(), A2.getNumParrafos())));
       SW=Math.abs(A1.getNumPalabras()-A2.getNumPalabras());
       SW=1-(SW/(Math.max(A1.getNumPalabras(), A2.getNumPalabras())));
       this.similitudE=(0.2*SP)+(0.8*SW);
    }
}
```

Anexo B: Clase SimilitudHD

```
public class SimilitudHD {
   double similitudHD;
   ArrayList<HuellaDigital> HD;
   public ArrayList<HuellaDigital> getHD() {
      return HD;
   }
   public double getSimilitudHD() {
      return similitudHD;
   }
   public void comparaHD(Archivos A1,Archivos A2,int t,int k) {//t debe ser mayor a k
      String texto="";
      double hashValues[];
      ArrayList<HuellaDigital> HD1,HD2;
      int i,j;
      HuellaDigital aux;
      if ((t<=0 || k<=0) || (t<=k)){
         if (Math.max(A1.getTokensArch().size(), A2.getTokensArch().size())>500){
            t=27;
            k=7;
         }
      }
      for (i=0; i<A1.getTokensArch().size(); i++) texto=texto+A1.getTokensArch().get(i);
      if (texto.length()%k!=0)hashValues= new double[(texto.length()/k)+1];
      else hashValues= new double[texto.length()/k];
      for(i=0;i<texto.length()%k;i++) texto=texto+" ";
      j=0;
      for(j=0;j<texto.length()/k;j++){
         hashValues[j]=0;
         for(i=0;i<k;i++)hashValues[j]=hashValues[j]+Math.pow((7*texto.charAt(j+i)), k-(i+1));
         hashValues[j]=hashValues[j]%999999937;
      }
      HD1=new ArrayList<HuellaDigital>();
      for(i=0;i<hashValues.length;i++){
         aux= new HuellaDigital();
         aux.setHuellaD(hashValues[i]);
         for(j=1;j<t-k+1;j++)if(i+j<hashValues.length) aux.setHuellaD(Math.min(aux.getHuellaD(),hashValues[i+j]));
         for (j=i;j<hashValues.length;j++){
            if (hashValues[j]==aux.getHuellaD()){
               i=j;
               aux.setPos(i);
               j=hashValues.length;
            }
         }
         HD1.add(aux);
      }
      texto="";
      for (i=0; i<A2.getTokensArch().size(); i++) texto=texto+A2.getTokensArch().get(i);
      if (texto.length()%k!=0)hashValues= new double[(texto.length()/k)+1];
      else hashValues= new double[texto.length()/k];
      for(i=0;i<texto.length()%k;i++) texto=texto+" ";
```

```
      j=0;
      for(j=0;j<texto.length()/k;j++){
         hashValues[j]=0;
         for(i=0;i<k;i++)hashValues[j]=hashValues[j]+Math.pow((7*texto.charAt(j+i)), k-(i+1));
         hashValues[j]=hashValues[j]%999999937;
      }
      HD2=new ArrayList<HuellaDigital>();
      for(i=0;i<hashValues.length;i++){
         aux=new HuellaDigital();
         aux.setHuellaD(hashValues[i]);
         for(j=1;j<t-k+1;j++)if(i+j<hashValues.length) aux.setHuellaD(Math.min(aux.getHuellaD(),hashValues[i+j]));
         for (j=i;j<hashValues.length;j++){
            if (hashValues[j]==aux.getHuellaD()){
               i=j;
               aux.setPos(i);
               j=hashValues.length;
            }
         }
         HD2.add(aux);

      }
      this.HD=new ArrayList<HuellaDigital> ();
      aux=new HuellaDigital();
      aux.setHuellaD(HD1.size()+HD2.size());
      for(i=0;i<HD1.size();i++){
         for(j=0;j<HD2.size();j++){
            if(HD1.get(i).getHuellaD()-HD2.get(j).getHuellaD()==0){
               HD.add(HD1.get(i));
               HD.add(HD2.get(j));
               HD2.remove(j);
               HD1.remove(i);
               j=HD2.size()+1;
               i--;
            }
         }
      }
      this.similitudHD=(aux.getHuellaD()-(HD1.size()+HD2.size()))/aux.getHuellaD();
      aux.setHuellaD(-1);
      aux.setPos(k);
      HD.add(aux);
   }
}
```

Anexo C: Clase SimilitudSCL

```
public class SimilitudSCL {
   private double similitudSCL;
   private ArrayList<Match> tile;

   public ArrayList<Match> getTile() {
      return tile;
   }

   public double getSimilitudSCL() {
      return similitudSCL;
   }

   public void comparaSCL(Archivos A1, Archivos A2,int m) {
      ArrayList<SCM> textoA1=new ArrayList<SCM>();
      ArrayList<SCM> textoA2=new ArrayList<SCM>();
      ArrayList<Match> matches;
      Match match;
      int i,j,k,MaxMatch;
      for (i=0;i<A1.getTokensArch().size();i++){
         if(!A1.getTokensArch().get(i).equalsIgnoreCase("")){
            SCM aux=new SCM();
            aux.setToken(A1.getTokensArch().get(i));
            aux.setMarcado(false);
            textoA1.add(aux);
         }
      }
      for (i=0;i<A2.getTokensArch().size();i++){
         if(!A2.getTokensArch().get(i).equalsIgnoreCase("")){
            SCM aux=new SCM();
            aux.setToken(A2.getTokensArch().get(i));
            aux.setMarcado(false);
            textoA2.add(aux);
         }
      }
      tile= new ArrayList<Match>();
      do{
         MaxMatch=m;
         matches= new ArrayList<Match>();
         for(i=0; i<textoA1.size();i++){
            if(!textoA1.get(i).isMarcado()){
               for(k=0; k<textoA2.size();k++){
                  if(!textoA2.get(k).isMarcado()){
                     j=0;
                     while((i+j<textoA1.size())&&(k+j<textoA2.size()) &&
textoA1.get(i+j).getToken().equalsIgnoreCase(textoA2.get(k+j).getToken()) && !textoA1.get(i+j).isMarcado() &&
!textoA2.get(k+j).isMarcado())j++;
                     if (j==MaxMatch){
                        match=new Match();
                        match.setPosA1(i);
                        match.setPosA2(k);
```

```
                    match.setLongitud(j);
                    matches.add(match);
                }
                else if(j>MaxMatch){
                    matches.removeAll(matches);
                    match=new Match();
                    match.setPosA1(i);
                    match.setPosA2(k);
                    match.setLongitud(j);
                    matches.add(match);
                    MaxMatch=j;
                }
              }
            }
          }
        }
        for(i=0;i<matches.size();i++){
          for(j=0;j<MaxMatch-1;j++){
            textoA1.get(matches.get(i).getPosA1()+j).setMarcado(true);
            textoA2.get(matches.get(i).getPosA2()+j).setMarcado(true);
          }
          tile.add(matches.get(i));
        }
      }while(MaxMatch>m);
      j=0;
      for (i=0;i<tile.size();i++)j=j+tile.get(i).getLongitud();
      similitudSCL=2*j;
      similitudSCL=similitudSCL/(textoA1.size()+textoA2.size());
    }
}
```

Anexo D: Clase SimilitudT

```
public class SimilitudT {
   private double similitudT;
   private ArrayList<String[]> trigrama;
   public ArrayList<String[]> getTrigrama() {
      return trigrama;
   }
   public double getSimilitudT() {
      return similitudT;
   }
   public void comparaT(Archivos A1, Archivos A2) {
      int i, index;
      String aux[] = new String[5];
      trigrama = new ArrayList<String[]>();
      if (A1.getTokensArch().size() >= 3 && A2.getTokensArch().size() >= 3) {
         aux[0] = A1.getTokensArch().get(0);
         aux[1] = A1.getTokensArch().get(1);
         aux[2] = A1.getTokensArch().get(2);
         aux[3] = "0";
        aux[4] = "i";
```

```
trigrama.add(aux);
for (i = 1; i < A1.getTokensArch().size() - 2; i++) {
   aux = new String[5];
   aux[0] = A1.getTokensArch().get(i);
   aux[1] = A1.getTokensArch().get(i + 1);
   aux[2] = A1.getTokensArch().get(i + 2);
   aux[3] = "" + i;
   aux[4] = "i";
   index = binarySearch(trigrama, aux);
   if (index >= 0) {
      aux[0] = trigrama.get(index)[0];
      aux[1] = trigrama.get(index)[1];
      aux[2] = trigrama.get(index)[2];
      aux[3] = trigrama.get(index)[3] + ":" + i;
      aux[4] = trigrama.get(index)[4];
      trigrama.set(index, aux);
   } else trigrama.add(aux);
   trigrama = insertionSort(trigrama);
}
for (i = 0; i < A2.getTokensArch().size() - 2; i++) {
   aux = new String[5];
   aux[0] = A2.getTokensArch().get(i);
   aux[1] = A2.getTokensArch().get(i + 1);
   aux[2] = A2.getTokensArch().get(i + 2);
   aux[3] = "i";
   aux[4] = "" + i;
   index = binarySearch(trigrama, aux);
   if (index >= 0) {
      aux[0] = trigrama.get(index)[0];
      aux[1] = trigrama.get(index)[1];
      aux[2] = trigrama.get(index)[2];
      aux[3] = trigrama.get(index)[3];
      if (trigrama.get(index)[4].equalsIgnoreCase("i")) {
         aux[4] = "" + i;
      } else {
         aux[4] = trigrama.get(index)[4] + ":" + i;
      }
      trigrama.set(index, aux);
   } else trigrama.add(aux);
   trigrama = insertionSort(trigrama);
}
similitudT = 0;
for (i = 0; i < trigrama.size(); i++) {
   if (!trigrama.get(i)[3].equalsIgnoreCase("i") && !trigrama.get(i)[4].equalsIgnoreCase("i"))  similitudT++;
}
similitudT = similitudT / trigrama.size();
for (i = 0; i < trigrama.size(); i++) {
   if(trigrama.get(i)[3].equalsIgnoreCase("i") || trigrama.get(i)[4].equalsIgnoreCase("i")){
                trigrama.remove(i);
                i--;
   }  } }
```

```
private int binarySearch(ArrayList<String[]> t, String k[]) {
   int start, end, midPt;
   start = 0;
   end = t.size() - 1;
   while (start <= end) {
      midPt = (start + end) / 2;
      if (t.get(midPt)[0].compareTo(k[0]) == 0 && t.get(midPt)[1].compareTo(k[1]) == 0 && t.get(midPt)[2].compareTo(k[2]) == 0) {
         return midPt;
      } else if (t.get(midPt)[0].compareTo(k[0]) < 0) {
         start = midPt + 1;
      } else if (t.get(midPt)[0].compareTo(k[0]) == 0 && t.get(midPt)[1].compareTo(k[1]) < 0) {
         start = midPt + 1;
      } else if (t.get(midPt)[0].compareTo(k[0])==0 && t.get(midPt)[1].compareTo(k[1])==0 && t.get(midPt)[2].compareTo(k[2])<0) {
         start = midPt + 1;
      } else {
         end = midPt - 1;
      }
   }
   return -1;
}
private ArrayList<String[]> insertionSort(ArrayList<String[]> t) {
   int j=t.size()-2;
   String temp[]=new String[3];
   temp[0]=t.get(t.size()-1)[0];
   temp[1]=t.get(t.size()-1)[1];
   temp[2]=t.get(t.size()-1)[2];
   while (j >= 0 && t.get(j)[0].compareTo(temp[0]) > 0)j--;
   while (j >= 0 && t.get(j)[0].compareTo(temp[0]) == 0 && t.get(j)[1].compareTo(temp[1]) > 0) j--;
   while(j >= 0 && t.get(j)[0].compareTo(temp[0])==0 && t.get(j)[1].compareTo(temp[1])==0&&t.get(j)[2].compareTo(temp[2])>0)j--;
   t.add(j+1, t.get(t.size()-1));
   t.remove(t.size()-1);
   return t;
}
```

Anexo E: Contenido de los archivos de prueba

Archivo 1

Mexsoft Inc.
Documento de Visión
Historia de Revisiones
Versión de Documento: 3.0
Fecha: 25/09/2011
Descripción de la Versión: Inclusión del caso de uso Reportes
Introducción
El proyecto tiene como objetivo proporcionar una tecnología web con la cual se pueda facilitar el proceso de Enseñanza-Aprendizaje mediante una aplicación que gestione los cursos impartidos por el profesor.
Secciones del documento
Orientación: En esta sección describiremos la forma en la que el cliente lleva a cabo el proceso de negocio actualmente.
Enunciado del problema: En base a la descripción del proceso de negocio, resaltamos las inquietudes del cliente y los problemas o limitaciones que se observan en dicho proceso.
Enunciado de la posición: En esta sección se presenta una explicación concisa del porqué nuestro producto se diferencia de las herramientas que actualmente tienen disponibles los profesores en la universidad.
Descripción de personal involucrado: Aquí presentaremos los roles del personal que estará involucrado en el proyecto, tanto del equipo de desarrollo, como personal por parte del cliente.
Visión General: En esta sección se explica el diagrama de contexto del sistema.

Resumen de las características del sistema: Se describen los requisitos funcionales del sistema.
Orientación *Oportunidad del negocio*
Actualmente el cliente cuenta con distintas herramientas para la gestión de los cursos que imparte en la universidad y de esto se desprende lo siguiente: Las aplicaciones que ofrece la universidad (Aula Virtual y CBI en línea, ambas basadas en Moodle) son muy completas y cuenta con muchas herramientas que el profesor no desea utilizar o no sabe cómo utilizar y se le complica el manejo de estas, como consecuencia, al no utilizar las herramientas disponibles, recurre a métodos rudimentarios de control, utilizando listas para el registro de estudiantes y distribuyendo material didáctico en hojas de papel. Debido a esto es necesaria una tecnología web que no sea tan compleja, pero tampoco tan simple, para que se pueda llevar a cabo una buena gestión de los cursos.
Enunciado del problema
Las aplicaciones que la universidad ofrece a sus profesores para la gestión de cursos, son complejas y requiere de tiempo para familiarizarse con todas las herramientas que estas ofrecen y algunos profesores optan por no utilizarlas, como consecuencia de esto, la gestión de cursos se torna complicada y la demanda de recursos y tiempo aumenta.
Enunciado de la posición
La aplicación web que se va a desarrollar está dirigida a cualquier profesor integrante de la Universidad Autónoma Metropolitana – Unidad Azcapotzalco, que tenga la necesidad de utilizar una herramienta sencilla pero que a la vez cumpla con sus expectativas para una buena gestión de cursos. El proyecto a desarrollar se diferencia de las aplicaciones disponibles porque no es tan complejo, pero tampoco muy simple, ya que se entrevista personalmente al usuario final y se atienden sus principales necesidades.
Una vez registrado en el sistema, este proporciona dos vistas con distintas herramientas que son suficientes para una buena gestión de cursos: Vista de estudiante y Vista de Profesor.
Vista de Estudiante
Listado de cursos a los que está inscrito el alumno. Inscribirse a un curso.
Material didáctico proporcionado por el profesor.
*Vista de Profes*or
Alta/Baja y Consulta de un curso. Alta/Baja y Consulta de un estudiante. Alta/Baja y Consulta de un recurso. Reporte de cursos impartidos. Reporte de alumnos inscritos.
Personal por parte del cliente
Dra. Beatriz Adriana González Beltrán – *Responsable del proyecto*
Alumnos y Profesores – *Usuarios primarios*
Objetivos de nivel de usuario
Los usuarios necesitan una plataforma para satisfacer sus objetivos:
Estudiantes: Obtener material didáctico sin necesidad de que el profesor se las proporcione, conocer con anticipación el tema de clase, conocer la tarea dejada en clase sin necesidad de asistir.
Profesores: Gestionar a los estudiantes, gestionar los cursos, gestionar los recursos, que son utilizados en los salones de clases.
Visión General del Sistema Computacional.
Perspectiva del sistema
La aplicación web se encontrará alojada en un servidor de la universidad, el cual proporcionara servicios a profesores y estudiantes que colaborarán en la gestión de cursos.
Resumen de los beneficios del sistema
Características a Soportar y Beneficio del personal por parte del cliente.
Funcionalmente, el sistema proporcionará servicios que faciliten la gestión de cursos impartidos en la universidad, incluyendo manejo de listas de alumnos, manejo de recursos didácticos, etc. Funcionalmente, el sistema proporcionará servicios que faciliten la gestión de cursos impartidos en la universidad, incluyendo manejo de listas de alumnos, manejo de recursos didácticos, etc.
Registro de Alumnos a los cursos vía internet. Herramienta para facilitar el acceso a recursos didácticos.
Reporte de estudiantes matriculados, con descripción del curso al que pertenecen.
Mayor organización sobre los estudiantes matriculados a distintos grupos.
Reporte de cursos impartidos. Se tiene información de los cursos que se imparte y los alumnos que pertenecen a estos
Resumen de las características del sistema
Registro de usuarios
Perfiles: una vez registrado un usuario en el sistema, este presenta dos perfiles: Alumno y Profesor.
Alta/Baja de cursos
Alta/Baja de alumnos
Compartir Material/Tarea
Inscribirse a un curso
Retirarse de un curso
Descarga de material
Autenticar usuario: El sistema solicita autenticarse a los usuarios para poder ingresar.
Reportes de alumnos y de cursos

Archivo 2

MFM Inc. Documento de Visión Historia de Revisiones Versión de Documento: 1.0 Fecha: 15/05/2012
Descripción de la Versión: Creación de documento de visión.
Introducción
El proyecto tiene como objetivo proporcionar una tecnología computacional con la cual se pueda facilitar el manejo de la microempresa "El Codito" mediante una aplicación que gestione los procesos actuales de esta.
Secciones del documento
Orientación: En esta sección describiremos la forma en la que el cliente lleva a cabo el proceso de negocio actualmente.
Enunciado del problema: En base a la descripción del proceso de negocio, resaltamos las inquietudes del cliente y los problemas o limitaciones que se observan en dicho proceso.
Enunciado de la posición: En esta sección se presenta una explicación concisa del porqué nuestro producto se diferencia delas herramientas que actualmente se tienen disponibles en el mercado.
Descripción de personal involucrado: Aquí presentaremos los roles del personal que estará involucrado en el proyecto, tanto del equipo de desarrollo, como personal por parte del cliente.
Visión General: En esta sección se explica el diagrama de contexto del sistema.
Resumen de las características del sistema: Se describen los requisitos funcionales del sistema.
Orientación
Oportunidad del negocio
Actualmente el cliente cuenta con distintas herramientas para la gestión de su microempresa y de esto se desprende lo siguiente: Las aplicaciones que ofrece el merado actualmente(Hojas de cálculo, etc.) son muy completas y cuenta con muchas herramientas que el cliente no desea utilizar o no sabe cómo utilizar y se le complica el manejo de estas, como consecuencia, al no utilizar las herramientas disponibles, recurre a métodos rudimentarios de control, utilizando cuadernos para el registro de su inventario, ventas , registro de clientes , distribuidores y pedidos. Debido a esto es necesaria una tecnología computacional que no sea tan compleja, pero tampoco tan simple, para que se pueda llevar a cabo una buena gestión de la microempresa.
Enunciado del problema
Las aplicaciones que ofrece el mercado actualmente para nuestro cliente para la gestión de su microempresa, son complejas y requiere de tiempo para familiarizarse con todas las herramientas que estas ofrecen y debido a esto opta por no utilizarlas, como consecuencia de esto, la gestión de se torna complicada y la demanda de recursos y tiempo aumenta, así como la perdida de ventas por no tener un inventario actualizado y la fuga de dinero por no tener todas las ventas registradas.
Enunciado de la posición
La aplicación que se va a desarrollar está dirigida específicamente al giro de negocio de la microempresa "El Codito", la cual tiene la necesidad de utilizar una herramienta sencilla pero que a la vez cumpla con sus expectativas para una buena gestión de sus procesos. El proyecto a desarrollar se diferencia de las aplicaciones disponibles porque no es tan complejo, pero tampoco muy simple, ya que se entrevista personalmente al usuario final y se atienden sus principales necesidades.
Una vez registrado en el sistema, este proporciona una vista distinta al negocio con herramientas que son suficientes para una buena gestión de sus procesos.
Vista del sistema
Listado del Inventario.
Control de ventas y pedidos.
Registro de clientes y proveedores.
Descripción del personal involucrado
Personal por parte del cliente
Alberto Isidro Alanis – *Responsable y dueño de la microempresa (Usuario Primario).*
Empleados, Clientes y Proveedores – *Usuarios Secundarios*
Objetivos de nivel de usuario
Los usuarios necesitan una aplicación para satisfacer sus objetivos:
Clientes y proveedores: Ser atendidos de manera más rápida y eficiente.
Empleados: Gestionar las ventas, gestionar los pedidos y gestionar el inventario.
Dueño: Generación de reportes de ventas, pedidos e inventario.
Visión General del Sistema Computacional.
Perspectiva del sistema

La aplicación se encontrará alojada en el equipo de cómputo de la microempresa, el cual proporcionara servicios a empleados que colaborarán en la gestión de los procesos del negocio.
Resumen de los beneficios del sistema
Características a Soportar y Beneficio del personal por parte del cliente.
Funcionalmente, el sistema proporcionará servicios que faciliten la gestión del negocio “El Codito”, incluyendo manejo de ventas, pedidos, inventario, etc. Se disminuye el uso de recursos como papel, tinta, etc. Además de facilitar los procesos del negocio y tener un mejor control.
Registro de proveedores y clientes. Herramienta para facilitar el registro de las ventas y pedidos.
Reporte de las ventas en general, por cliente o por artículo y por fecha. Mayor organización sobre las ventas que permitirán un mejor administración del negocio.
Reporte del Inventario. Se tiene información de los productos y el total en bodega.
Resumen de las características del sistema
Registro de clientes y proveedores.
Alta/Baja de clientes y proveedores
Alta/Baja de productos
Control de ventas
Control de pedidos
Inventario
Reportes de de ventas e inventarios

Bibliografía

[1] C. Lyon, R. Barrett y J. Malcolm. (2003) Experiments in Electronic Plagiarism Detection. [En línea]. Disponible: https://uhra.herts.ac.uk/dspace/bitstream/2299/1774/1/S90.pdf

[2] L. E. García Rodríguez, "Obtención de una medida cuantitativa de similitud de códigos fuente escritos en lenguaje C", proyecto terminal, División de CBI, Universidad Autónoma Metropolitana Azcapotzalco, México, 2007.

[3] S. L. Pérez Pérez, "Adaptación del detector de copias para códigos fuente escritos en C++ y Java", proyecto terminal, División de CBI, Universidad Autónoma Metropolitana Azcapotzalco, México, 2008.

[4] WCopyfind (2012, Octubre 24). WCopyfind Software and Instructions [En line]. Disponible en: http://plagiarism.bloomfieldmedia.com/z-wordpress/software/wcopyfind/

[5] Ferret (2012, Octubre 24). Plagiarism Detection Research Group [En línea]. Disponible en: http://homepages.stca.herts.ac.uk/~pdgroup/

[6] Plagiarism Checker (2012, Octubre 24). The Plagiarism Checker [En línea]. Disponible en: http://www.dustball.com/cs/plagiarism.checker/

[7] B. Coppin, "Understanding Language" in Artificial intelligence illuminated, Canada: Jones and Bartlett Publishers, 2004.

[8] WordNet 3.0 (2012, Noviembre 20). Multilingual Central Repository [En línea]. Disponible en: http://adimen.si.ehu.es/web/MCR.

[9] WordNet (2013, Marzo). Schema[En linea]. Disponible en: http://wnsql.sourceforge.net/schema.html.

[10] NetBeans (2012, Octubre 25). NetBeans IDE 7.2 Release Information [En línea]. Disponible en: http://netbeans.org/community/releases/72/index.html

[11] MySQL (2012, Octubre 25). About MySQL [En línea]. Disponible en: http://www.mysql.com/about/

[12] C. Lyon, R. Barrett y J. Malcolm. (2004, Junio) A theoretical basis to the automated detection of copying between texts, and its practical implementation in the Ferret plagiarism and collusion detector. [En línea]. Disponible: http://homepages.stca.herts.ac.uk/~comqcml//

[13] J. Parapar y A. Barreiro. (2008, Octubre) Winnowing-based text clustering. [En línea]. Disponible: http://www.dc.fi.udc.es/~barreiro/publications/cikm0496-final-parapar.pdf

[14] M. J. Wise (1993, Marzo). Running Karp-Rabin Matching and Greedy String Tiling. [En línea]. Disponible: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.137.7965&rep=rep1&type=pdf

[15] R. M. Karp, M. O. Rabin, Efficient randomized pattern-matching algorithms, IBM Journal of Research and Development 31 (2) (1987) 249–260.