FACULTAD DE CIENCIAS

Facultad de Ciencias

Proyecto Fin de Carrera

DETECTOR DE FRAUDE ACADÉMICO
(Academic Fraud Detector)

Para acceder al Título de

INGENIERO EN INFORMÁTICA

Autor: Jorge Aranda de Miguel
Directora: Cristina Tîrnăucă

Febrero - 2014

Agradecimientos

Este trabajo supone la culminación de un proyecto que ha ocupado cinco años de mi vida. Cinco años que me han formado y me han preparado para lo que supondrá mi vida a partir de ahora. Mi Proyecto Fin de Carrera es el trabajo que completa otro proyecto de futuro aún mayor, mi titulación superior, con todo lo que eso conlleva.

En primer lugar, quiero dar las gracias a mi directora de mi Proyecto Fin de Carrera, *Cristina Tîrnăucă* sin la cual yo jamás hubiera hecho este trabajo. A ella le debo el haber encontrado la temática definitiva del trabajo y los conocimientos necesarios para llevarlo a la práctica. Ha sido mi mentora y la que ha sentado las bases sobre cómo tenía que realizar este trabajo. Gracias.

Hay tantas personas que han estado a mi lado durante la carrera, a pesar de las adversidades y muy a mi profundo pesar no voy a poder nombrarlas a todas. Sin embargo, quiero que esas personas sepan que les estoy muy agradecido por su apoyo en cualquiera de las formas. Ya sea porque han estado acompañandome durante toda la carrera o bien, por haber compartido experiencias, por haber sido compañeros en las prácticas o porque una tarde teníamos que estudiar los dos un examen clave y decidímos hacerlo juntos. Gracias.

Desde luego, y sin dudarlo dos veces he de mencionar a dos compañeros de carrera con los que he vivido aventuras y desventuras y hemos pasado por momentos tensos, agradables, tristes y de trabajo duro. Hablo de *Santos Merino* y *Gonzalo Aragoneses*; especialmente dura fue la enfermedad que sufrió Santos y me alegro enórmemente de que se recuperara y que haya conseguido llegar hasta donde ha llegado. Gracias.

También quiero agradecer a mis cuatro mejores amigos: *Manuel Margallo*, *Lucas Canteras*, *Pablo Pérez* y *Diego González*. Ninguno de ellos compañero de carrera. Siempre que tenía dificultades con mis estudios estaban ahí para

aconsejarme o apoyarme. Gracias.

Dar las gracias a aquellas personas que me hicieron ser consciente de muchas cosas que antes ignoraba. Dar las gracias a mis profesores, pues ellos me han enseñado lo que necesito para triunfar con éxito en la vida. Doy las gracias a mi padre porque gracias a él he podido estudiar una carrera universitaria y a mi madre porque desde niño siempre me ha dicho que tenía que hacer en la vida lo que a mí me gustara; así lo hice. Gracias.

Y por último y no por ello menos importante, quiero dar las gracias a una persona muy importante en mi vida. Una persona que ha sabido sacar y potenciar todo lo bueno que había en mí y demostrarme que era capaz de mucho más de lo que me veía capaz de hacer y trabajar el día antes de conocerla. Que ha conseguido que tenga un último año de carrera exitoso. Muchas gracias *Esther Alcoceba*.

Resumen

En la actualidad, el fraude académico constituye un serio problema para toda la comunidad universitaria. Herramientas de detección automatizadas pueden ser de gran utilidad para los docentes a la hora de detectar estos casos.

El objeto de estudio de este proyecto es tratar de conocer la mejor manera de resolver el problema de la detección del plagio, así como conocer y explicar las diversas técnicas conocidas que existen para analizar el lenguaje humano y lenguajes de programación de alto nivel en busca de patrones que permitan calcular la similitud entre dos documentos de la misma clase.

Dicho estudio ha permitido llevar a la práctica un aplicativo desarrollado en el lenguaje de programacíon C# para la plataforma .NET Framework que lleva a cabo las funciones de análisis y comparación de documentos. Todas las etapas del desarrollo de este aplicativo han sido documentadas en el presente trabajo y se han realizado pruebas que verifican el funcionamiento del aplicativo.

El trabajo concluye con posibles vias de ampliación del aplicativo y cuales serían las dificultades y pasos a seguir si finalmente se decide ampliarlo.

Palabras clave: Csharp, Fraude académico, análisis del lenguaje, LMS, semejanza, .NET Framework

Abstract

Today, academic fraud is a serious problem for the whole academic community. Automatic plagiarism detection tools can help professors discover those cases.

The purpose of this project is to know the best way to solve Plagiarsm detection problem, in addition to explaining the known available methods to analyze human and computer high level languagues looking for patterns which let us estimate the similarity degree between two documents of the same type.

The present document explains the steps taken in order to develop an application in C# languague compatible with .NET Framework Plattform, using these methods and knowledge. All steps have been recorded in this work and the application has been tested to check if the goals had been accomplished.

The work ends with future possibilites to extend this application and explains potential issues that could appear while trying to explore these options.

Keywords: CSharp, Academic Fraud, Languague Analysis, LMS, similarity, .NET Framework

Índice general

1. Introducción **9**
1.1. Inspiración 9
1.2. Propósito 10
1.3. Estado de la cuestión 10

2. Base teórica **12**
2.1. Tipos de análisis del plagio 12
2.1.1. Análisis explícito del plagio 12
2.1.2. Análisis implícito del plagio 12
2.2. Algoritmos empleados 13
2.2.1. Técnica de identificación de patrones 13
2.2.2. Identificación de los bloques 13
2.2.3. Longest Matching Sequence (LMS) 13
2.2.4. Similitud relativa entre dos documentos 14
2.2.5. Otros tipos de análisis 15
2.3. Implementación empleada 16

3. Requisitos IS del proyecto **17**
3.1. Descripción del aplicativo 17
3.2. Requisitos funcionales 17
3.3. Requisitos no funcionales 19

4. Especificación y diseño del proyecto **22**
4.1. Casos de uso 22
4.2. Diagrama de arquitectura 23
4.3. Despliegue 24
4.4. Modelo de datos 24

5. Fase de desarrollo **26**
5.1. Separación de las capas 26
5.2. Satisfacción de los requisitos IS 27

5.3. Problemas encontrados 30
5.4. Medidas adoptadas 31

6. Evaluación y pruebas **32**
6.1. Origen del dataset empleado 32
6.2. Resultados obtenidos 32
6.2.1. Pruebas realizadas con lenguaje humano 32
6.2.2. Pruebas realizadas con lenguajes de alto nivel 33

7. Conclusiones generales y trabajos futuros **36**
7.1. Portado a otras plataformas 36
7.2. Búsquedas de fuentes externas 37
7.3. Análisis léxico y sintáctico 37
7.4. Conclusiones finales 37

Bibliografía **39**

Capítulo 1

Introducción

1.1. Inspiración

Los orígenes de este *proyecto final de carrera* se remontan al año 2012, año en el que cursé la asignatura de Minería de Datos impartida por Cristina Tîrnăucă y Domingo Gómez y más concretamente al *trabajo final de la asignatura*.

La finalidad del trabajo planteado era la realización de un proyecto encaminado a resolver un problema planteado por el usuario utilizando mecanísmos y técnicas basadas en minería de datos. Mi trabajo, denominado *Precrime*, utiliza técnicas de procesamiento del lenguaje natural (mediante el uso de un filtro Bayesiano) para procesar un dataset de mensajes interceptados mediante distintos canales. Cabe destacar que ese proyecto no entra a valorar el origen de dichos mensajes, siendo competencia de las autoridades. La finalidad de *Precrime* era detectar automáticamente aquellos mensajes potencialmente delictivos, y clasificar a los autores de los mensajes en función del grado de peligrosidad, ayudando a las autoridades a centrar sus investigaciones.

El proceso consiste en que las autoridades policiales y judiciales clasifican un subconjunto del dataset de mensajes entre "buenos" y "malos". Una vez entrenado el filtro con suficientes mensajes se podría llegar a precedir con éxito y de forma automatizada la intencionalidad del resto del dataset de mensajes.

Tras la realización de dicho trabajo, planteé a Cristina Tîrnăucă la posibilidad de extender el alcance de aquel trabajo. Después de realizar una búsqueda en repositorios académicos me inspiré en un artículo denominado *Minority Report in Fraud Detection: Classification of Skewed Data* [1]. El artículo sostiene que la forma más eficiente de detectar fraude en tarjetas

de crédito y partes a las compañias de seguro es emplear el uso de tres algoritmos de forma combinada, utilizando siempre el resultado mayoritario y descartando el resultado del discrepante. Dicho artículo inspiró la temática final de mi proyecto fin de carrera, por haberme llevado a la idea de realizar un *Detector de Fraude Académico* con técnicas similares a las planteadas por los autores de aquel trabajo.

1.2. Propósito

El fraude académico, comunmente conocido como plagio, supone un serio problema para la comunidad académica y un verdadero quebradero de cabeza para aquellos docentes que han de detectar estos casos.

El propósito de este proyecto consiste en planificar, especificar, diseñar, codificar e implementar un aplicativo autónomo capaz de detectar posibles plagios cometidos por estudiantes a la hora de realizar sus prácticas o trabajos, ayudando al docente en sus funciones de forma considerable.

Debido a la naturaleza de esta titulación, la idea consiste en poder analizar tanto trabajos escritos en un lenguaje humano como códigos escritos en lenguajes de programación de alto nivel tan comunes en nuestra titulación.

1.3. Estado de la cuestión

Se han encontrado herramientas de similares características al proyecto desarrollado, como por ejemplo las siguientes herramientas en línea: Turitin [2], iThenticate [3], Plagiarism detect [4], Plagius [5], Ephorus [6], Farejador de Plagio [7], JPlag [8] y DOC Cop [9].

Turitin, iThenticate, Plagius y Ephorus pueden realizar búsquedas externas (es decir, fuera de los documentos comparados; por ejemplo, en repositorios académicos) mientras que el resto se limitan únicamente a comparaciones locales entre documentos. A continuación se detallan algunas de las herramientas más destacadas:

Turitin: Se trata de una herramienta de pago capaz de analizar el lenguaje humano con una gran base de datos de toda la comunidad científica. Funciona en la web y es capaz de subir documentos a partir de los distintos provedores cloud como Dropbox o Google Drive.

iThenticate: Muy similar a la anterior y muy utilizada pero excesivamente cara (un dolar por documento analizado). Su base de datos contiene más de 37 millones de artículos.

JPlag: Se trata de una herramienta desarrollada por el *Institute for Program Structures and Data Organization* en Java y tiene la ventaja de ser gratuita, aunque únicamente es capaz de detectar plagio en local.

Sin embargo, todas las herramientas anteriormente mencionadas están enfocadas a la detección de plagio en documentos escritos en lenguajes humanos. Mientras que la finalidad de este proyecto es la de poder ser utilizado también en el análisis de plagio en ficheros de código fuente.

Capítulo 2

Base teórica

2.1. Tipos de análisis del plagio

2.1.1. Análisis explícito del plagio

Es el más sencillo de implementar. Consiste en buscar en una gran base de datos documental coincidencias exactas de los textos. El problema planteado es el coste computacional de dichas búsquedas. En este tipo de análisis, se busca minimizar el impacto de dicho coste todo lo posible.

2.1.2. Análisis implícito del plagio

El análisis implícito del plagio [10] consiste en determinar patrones concretos como el estilo, el léxico empleado y cualquier otro detalle que pueda delatar el estilo personal del autor del texto. El análisis implícito se encuentra íntimamente ligado a la compresión de los textos. Dos textos que deriven en una compresión muy similiar tienen altas probabilidades de ser un plagio uno de otro (al cual se han cambiado palabras sueltas o se ha modificado el orden de las frases).

Este tipo de análisis, en principio, está más enfocado a comparar "documentos locales"(entre dos trabajos presentados por dos alumnos por ejemplo) donde, por ejemplo, uno de los estudiantes haya modificado levemente el contenido del trabajo de otro estudiante.

2.2. Algoritmos empleados

2.2.1. Técnica de identificación de patrones

A la hora de detectar plágio dentro de un documento académico de cualquier índole, es necesario identificar patrones entre documentos de su misma clase (artículos, resúmenes, ensayos, códigos fuente de programación, etc.). En definitiva, comparar bloques independientes de texto entre una pareja de documentos con el objetivo de identificar la "semejanza"de los mismos. Independientemente del tipo de análisis efectuado, es decir, desde el más básico (comparación literal de textos), pasando por otros más sofisticado (análisis sintático, semántico... etc), hasta el más avanzado (reconocimiento de estilos y "marcas de autoría", etc.) todos están basados en comparar dos textos divididos en bloques de texto iguales (o sensiblemente similares) o bien distintos (o sensiblemente diferentes).

2.2.2. Identificación de los bloques

La pregunta es, ¿cómo se identifican adecuadamente dichos bloques dentro de un documento? Existe un algoritmo denominado **Longest Matching Sequence** (la cadena de texto común más larga). El algoritmo trata de dividir dos secuencias de texto en varias subsecuencias hasta que se identifica la más larga de ambas cadenas iniciales. Tras realizar esta operación se pueden obtener las secuencias comunes de cada texto ordenadas de mayor a menor haciendo que el texto común sea identificado, incluso aunque las subcadenas se encuentren en distintas posiciones del texto.

Una vez identificadas esas secuencias, el documento total queda dividido en bloques comunes y distintos con respecto a su documento correlativo. A este proceso se le conoce comunmente como Diff. Nótese que no es necesario que los bloques estén situados en la misma posición en cada uno de los respectivos documentos para ser dentificados como idénticos. Pues su clasificación como idénticos es independiente de su ubicación.

2.2.3. Longest Matching Sequence (LMS)

Este es el algoritmo empleado internamente por Diff para procesar las similitudes entre dos documentos. A continuación, se describe el funcionamiento del algoritmo **LMS** [11]:

Siendo M el número de caracteres de la primera cadena y N el número de caracteres de la segunda cadena, genera un array MAT de tamaño M+1 x N+1 mediante un doble bucle recorre todas las posiciones de la matriz (i, j).

Cuando se encuentra con que M[i]==N[j], es decir, el caracter de la posición i de la primera cadena y el caracter de la posición j es el mismo, realiza lo siguiente:

1. Establece *MAT[i+1,j+1]=MAT[i,j] + 1*. El efecto que esto produce es que si en *MAT[i,j]* no había coincidencia alguna (vale 0), en *MAT[i+1,j+1]* habrá uno (primer caracter coincidente), pero si en *MAT[i,j]* había un 1 ya sabemos que no es el primer caracter coincidente, sino el segundo.

2. Se compara el resultado de *MAT[i+1,j+1]* con el máximo obtenido. Si es igual, se vacia el conjunto de cadenas más largas, encontramos y guardamos la subcadena que va desde i o j hasta el valor de *MAT[i+1,j+1]*. Si el valor máximo obtenido es igual que *MAT[i+1,j+1]* no se vacia el conjunto. Si es menor, no haremos nada.

3. Al concluir el algoritmo retorna el conjunto de las cadenas más largas encontradas.

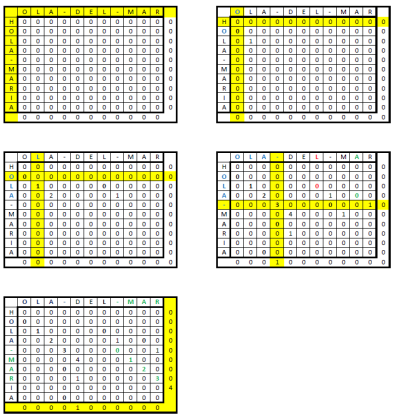

Figura 2.1: Ejemplo de ejecución del algoritmo LMS con dos secuencias de texto. El algoritmo retornará “ola ” y “mar” como cadenas comunes de mayor tamaño.

Nótese que cuando empleamos éste algoritmo dentro de Diff, se suprimen las cadenas de texto comunes con respecto al texto inicial. El algoritmo se repite hasta que sólo quedan las partes distintas.

2.2.4. Similitud relativa entre dos documentos

Una vez identificados los bloques idénticos, al ser los bloques de distinto tamaño implica que el que existan 3 bloques idénticos sobre un total de 4 bloques no significa que la similitud entre ambos sea del 75 %. Es decir,

es necesario tener en cuenta cada caracter dentro del texto para tener una medida única. Se totaliza el número de caracteres contenidos en los bloques distintos de cada documento, obteniendo *doc1-diff* y *doc2-diff*.

Teniendo en cuenta el número total de caracteres de cada documento *doc1* y *doc2* podremos calcular la "similitud relativa" entre documentos que se obtiene de restar al número total de caracteres, aquellos que se han incluido solo en *doc1* y a *doc2* deduciendo los exclusivos de ambos conjuntos.

Finalmente, dividimos el resultado entre el número total de caractéres de ambos documentos. Obteniendo un relativo entre los caracteres presentes en ambos documentos y el número total de caracteres.

La fórmula a aplicar sería la siguiente:

$$similarity_{relative} = \frac{doc_1 + doc_2 - (doc_1 diff + doc_2 diff)}{doc_1 + doc_2}$$

Sin embargo, la fórmula enunciada sólo tiene aplicación para evaluar lo "comunes" que resultan dos documentos entre sí. A veces, interesa obtener únicamente el relativo de partes de *doc1* que pertenecen a *doc2*. Por ejemplo, si se desea conocer si partes de *doc1* han sido obtenidas de *doc2*. Únicamente es conveniente tener en cuenta los caracteres comunes entre ambos documentos y relativizarlas con respecto al tamaño total de *doc1*.

La fórmula a aplicar sería la siguiente:

$$similarity_{doc_1 \subset doc_2} = \frac{doc_1 - doc_1 diff}{doc_1}$$

2.2.5. Otros tipos de análisis

Una de las ventajas de dicho algoritmo es que es posible construir otros comparadores introduciendo capas de análisis sintatico y gramatical con el fin de buscar estructuras independiente del léxico empleado en los documentos. Esta funcionalidad tiene propiedades muy interesantes, por ejemplo, nos permite "ignorar" cambios en los nombres de las variables entre dos documentos de código fuente o analizar exclusivamente la estructura gramatical del documento, lo que considerará como coincidencia la sustitución de palabras sinónimas.

Como paso inicial a la hora de clasificar las palabras según su tipo gramatical (sustantivo, verbo, adjetivo) es necesario separar la raíz del resto de la palabra. El algoritmo PorterStemmer realiza esta función para la lengua inglesa. Existe otra variante de dicho algoritmo (denominado Snowball Stemmers [12]) para otros idiomas como el alemán, el español, el rumano, italiano, etc.

2.3. Implementación empleada

Puesto que la implementación de un algoritmo diff (búsqueda automatizada de diferencias entre dos textos) ya existe, para la realización de esta parte del proyecto se procedió a la búsqueda de una implementación en C#.

Se encontraron dos alternativas: DiffPlex [13] y DiffEngine [14]. Entre las dos opciones, se optó finalmente por emplear DiffEngine debido a su mayor versatilidad (tres niveles de exploración del algoritmo, código abierto, ejemplos, documentación) en vez de la primera opción encontrada, DiffPlex, debido a que esta primera tenía una peor documentación y no hacía exactamente lo requerido en el proyecto.

Capítulo 3

Requisitos IS del proyecto

En este capítulo se expone el documento de requisitos que constituye una descripción técnica y detallada de lo que se espera del aplicativo.

3.1. Descripción del aplicativo

Aplicación desarrollada en C# para la plataforma *.NET Framework* con GUI gráfica basada en *Windows Forms* que permite a un docente verificar entre documentos de la misma clase y temática el grado de similitud entre estos, así como comprobar qué partes son idénticas.

A continuación, se detallan los requisitos funcionales y no funcionales que cumple el aplicativo derivado de la realización de este proyecto.

3.2. Requisitos funcionales

1. **REQUISITOS SOBRE LA PLATAFORMA DE LA APLICACIÓN:**

   R1.01: La aplicación será desarrollada para ser compatible con la plataforma .NET Framework 4 o compatibles.

   R1.02: La aplicación será codificada en el lenguaje de programación C# 4.

   R1.03: La GUI de la aplicación será desarrollada para ser compatible con la librería gráfica de Windows Forms.

   R1.04: El funcionamiento de la aplicación ha de ser óptimo en plataformas compatibles con Windows que tengan instalada la .NET Framework 4 o compatibles.

2. **REQUISITIOS SOBRE LA GUI DE LA APLICACIÓN**:

   R2.01: La GUI tendrá prioridad de proceso sobre otras cargas de trabajo a fin de mejorar la experiencia del docente en la utilización de la misma.

   R2.02: La aplicación deberá permitir al docente seleccionar el grado de similitud tolerada entre documentos de la misma temática.

   R2.03: La GUI permitirá seleccionar de forma limpia distintas temáticas de trabajo que contendrán documentos de diferentes autores relacionados con la misma temática de forma ordenada.

   R2.04: La GUI se diseñará con soporte multilíngüe.

3. **REQUISITOS SOBRE LA ENTRADA (OBTENCIÓN) DE DOCUMENTOS PROPIOS**:

   R3.01: La aplicación deberá permitir al docente cargar documentos almacenados en su propio sistema en ficheros de texto plano.

   R3.02: La aplicación deberá permitir al docente cargar documentos mediante la función de copiar/pegar del sistema operativo de documentos en texto plano.

   R3.03: La aplicación deberá permitir al docente cargar de forma masiva documentos mediante la autenticación en la plataforma Moodle y posterior selección de la actividad de la cual se desean descargar masivamente documentos.

4. **REQUISITOS SOBRE LA ENTRADA (OBTENCIÓN) DE DOCUMENTOS MEDIANTE FUENTES INTERNAS**:

   R4.01: La aplicación llevará a cabo un historial de documentos presentados en anteriores cursos de la misma temática.

   R4.02: La aplicación realizará a petición del docente comparaciones de los documentos mencionados en R4.01 con los que se desea comparar.

5. **REQUISITOS SOBRE LA COMPARACIÓN DE DOCUMENTOS**:

   R5.01: Los documentos a analizar serán de texto plano.

   R5.02: Se distinguen dos tipos de documento: Documentos de texto humano y Documentos de código fuente.

R5.03: A la hora de comparar unos documentos, se especificará como parámetro, el márgen de similitud tolerada entre documentos de la misma temática.

6. **REQUISITOS SOBRE EL CÁLCULO DE SIMILITUD ENTRE DOCUMENTOS A LA HORA DE COMPARAR DOCUMENTOS**:

   R6.01: La similitud de documentos será medida teniendo en cuenta la siguiente fórmula:

$$similarity_{doc_1 \subset doc_2} = \frac{doc_1 - doc_1 diff}{doc_1}$$

7. **REQUISÍTIOS SOBRE LOS RESULTADOS:**

   R7.01: Los resultados de las comparaciones serán ordenados de mayor a menor según el grado de similitud.

   R7.02: Se mostrarán aquellos bloques de texto que mayor similitud presenten en ambos documentos una vez seleccionada la pareja de documentos a comparar.

8. **REQUISITOS SOBRE EL ALMACENAMIENTO DE LA INFORMACIÓN RECOPILADA:**

   R8.01: La información se almacenará en una base de datos tipo SQL Lite ubicada en el espacio del usuario del sistema base.

   R8.02: Salvo petición contraria del docente o falta de espacio del usuario del sistema base, la aplicación recopilará documentos de años anteriores para su posterior comparación con nuevos trabajos de la misma temática descritos en R4.01.

   R8.03: Los resultados de las comparaciones también podrán ser almacenados con fines de expediente o para mejoras futuras de la información (estos datos no serán reemitidos al equipo de desarrollo mediante procedimientos de recopilación automatizada)

3.3. Requisitos no funcionales

La prioridad será tenida en cuenta en caso de conflicto con otro RNF (requisito no funcional).

RNF1: ESTABILIDAD [PRIORIDAD: CRÍTICA]

RNF1.1: La aplicación pasará por un proceso de verificación con cargas exhaustivas de trabajo y escenarios hostiles para descartar la probabilidad de fallos.

RNF1.2: La aplicación ha de permitir el procesamiento de trabajos en paralelo con la finalidad de no congelar la GUI y otras cargas de trabajo.

RNF2: INTEGRIDAD [PRIORIDAD: CRÍTICA]

RNF2.1: La aplicación garantizará en todo momento la integridad de los datos almacenados en la misma.

RNF3: USABILIDAD [PRIORIDAD: ELEVADA]

RNF3.1: La aplicación deberá ser intuitiva para el docente.

RNF3.2: La aplicación deberá ser sencilla de aprender utilizar para el docente.

RNF3.3: La aplicación deberá facilitar las funciones solicitadas en los requisitos funcionales.

RNF3.4: La aplicación mantendrá ordenados los contenidos para facilitar la labor del docente.

RNF3.5: La aplicación tendrá soporte para múltiples idiomas.

RNF4: COMPATIBILIDAD [PRIORIDAD: MUY ALTA]

RNF4.1: La aplicación será desarrollada para ser compatible con la plataforma .NET Framework 4 o compatibles.

RNF4.2: El funcionamiento de la aplicación ha de ser óptimo en plataformas compatibles con Windows que tengan instalada la .NET Framework 4 o compatibles.

RNF4.3: La GUI de la aplicación será desarrollada para ser compatible con la librería gráfica de Windows Forms.

RNF5: RENDIMIENTO [PRIORIDAD: MUY ALTA]

RNF5.1: La aplicación tratará de que la ejecucción de los algoritmos pesados como "diff"se realice de forma óptima.

RNF5.2: La GUI de la aplicación no deberá congelarse por la realización de trabajos pesados.

RNF5.3: La aplicación deberá ser capaz de utilizar técnicas multihilo para utilizar toda la capacidad de trabajo de la máquina host (multiprocesos en distintos procesadores físicos o lógicos).

RNF6: CONCURRENCIA [PRIORIDAD: ALTA]

RNF6.1: La GUI de la aplicación deberá controlar que el usuario pueda interactuar con otras partes de la interfaz cuando existan cargas de trabajo pesadas en segundo plano.

RNF7: DISPONIBILIDAD [PRIORIDAD: ALTA]

RNF7.1: La aplicación deberá estar disponible en modo local cuando el docente solicite su uso.

RNF8: ESCALABILIDAD, REUTILIZACIÓN Y MANTENIMIENTO [PRIORIDAD: ELEVADA]

RNF8.1: La aplicación ha de ser altamente reutilizable en otros proyectos.

RNF8.2: La aplicación se documentará, de tal forma que su mantenimiento y ampliación sean sencillos para el desarrollador o equipo de desarrollo.

RNF8.3: La aplicación se desarrollará siguiendo el patrón de diseño de Modelo-Vista-Controlador (MVC)

RNF9: PORTABILIDAD [PRIORIDAD: MEDIA]

RNF9.1: La aplicación debería diseñarse teniendo en cuenta su posibilidad de portado a otras plataformas y entornos. Especialmente en su GUI (entorno web, aplicaciones para móviles... etc).

RNF9.2: Con el objetivo de facilitar el RNF9.1, la distribución de la aplicación y su código fuente, así como su documentación se realizará mediante una licencia GPL académica.

RNF10: INTEROPERATIBILIDAD [PRIORIDAD: BAJA]

RNF10.1: La aplicación no deberá interferir con ningún otro sistema funcional del computador del docente. Es conveniente tener en cuenta que por la propia naturaleza y alcance de la aplicación este requisito es poco o nada relevante.

RNF11: INTEGRACIÓN [PRIORIDAD: NINGUNA]

RNF11.1: Esta aplicación no tiene requisitos de integración propios debido a su sencillez e interactividad con la capa de llamadas al sistema y la plataforma sobre la que se ejecutará que llevará a cabo sus funciones de integración.

Capítulo 4

Especificación y diseño del proyecto

En el capítulo anterior se expuso lo que constituye el documento de requisitos del aplicativo. En este capítulo se presentan los diagramas que permiten describir en lenguaje UML las características y funcionalidad del aplicativo.

4.1. Casos de uso

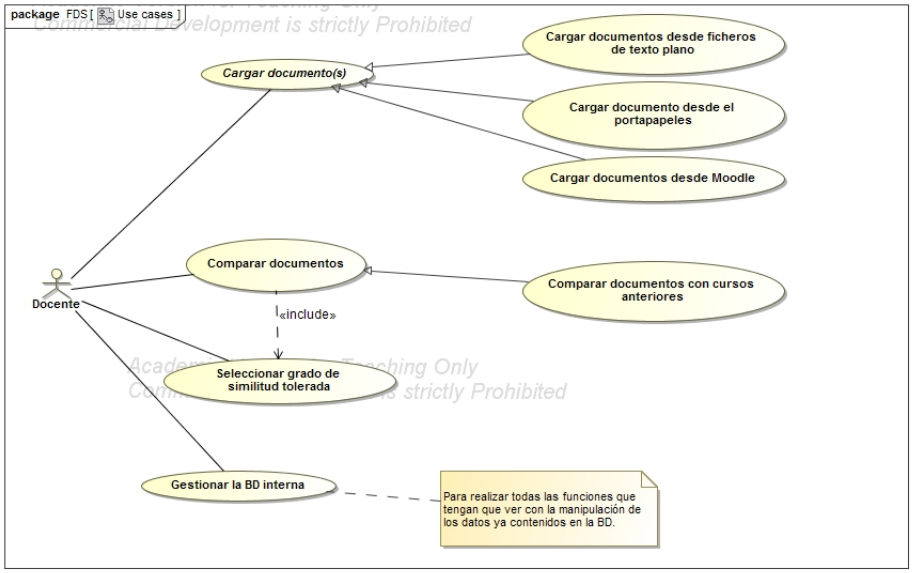

Figura 4.1: Diagrama de casos de uso

Nótese que el caso de uso *Cargar documento(s)* es abstracto porque una

misma funcionalidad puede ser provista de hasta tres formas distintas. Se ha planteado un caso de uso genérico para dar de alta, modificar y dar de baja asignaturas, ejercicios, cursos, etc. Este caso de uso es propio de cualquier aplicación de tipo gestión.

El grado de similitud tolerada será seleccionado en la vista de comparación de documentos.

4.2. Diagrama de arquitectura

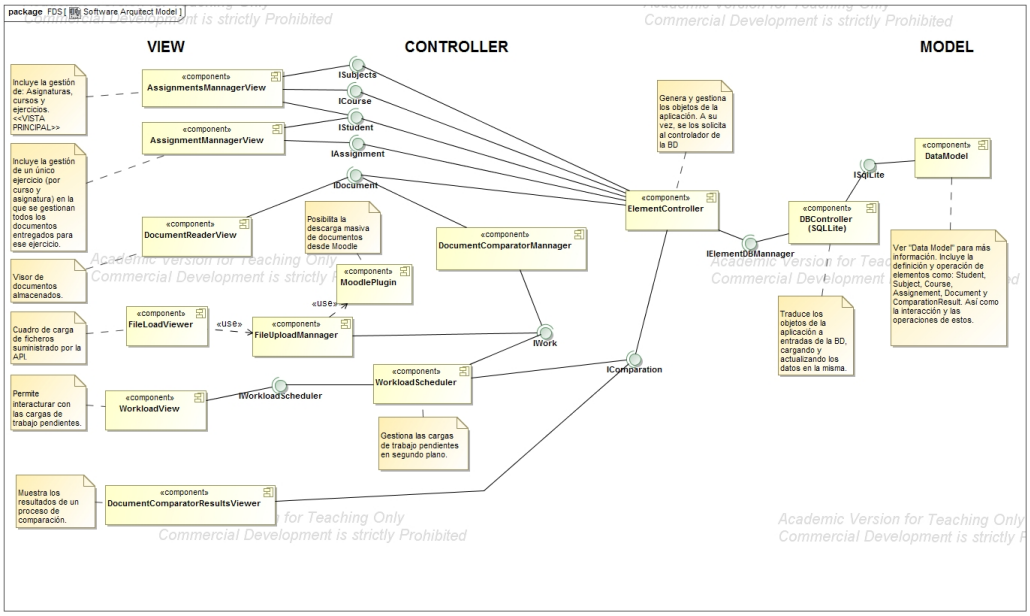

Figura 4.2: Diagrama de arquitectura

Lo primero que hay que destacar de esta arquitectura es que implementa un patrón de Modelo-Vista-Controlador tambien conocido como MVC. Tal y como menciona el Documento de Requisitos, la idea es crear una vista basada en *Windows Forms*, pero se desea que tanto el controlador, como el modelo puedan ser reutilizados a la hora de crear nuevas vistas para otras plataformas (por ejemplo: dispositivos móviles, interfaz web, interfaz GTK+ Mono [15] para Mac o Linux, etc.).

Se ha incluido un planificador de la carga de trabajo, *WorkloadScheduler*, para poder garantizar la independencia de la GUI con los procesos de larga ejecución.

Todos los componentes utilizan objetos generados por el aplicativo, es decir, en ningún caso cargarán datos directamente contra la BD. Un compo-

nente clave denominado *DBController* hace las llamadas contra la BD recibiendo como parámetros las referencias de los objetos manejados y definidos por el aplicativo.

El sistema solamente presenta un único subsistema denominado *Moodle-Plugin*, encargado de la descarga automatizada de trabajos presentados por los estudiantes a través de la plataforma Moodle. Además este subsistema presenta el inconveniente de ser monolítico. Es decir, que no sigue el patrón MVC anteriormente descrito, por las razones que se justificarán en la Sección 5.3 acerca de los *problemas encontrados*.

4.3. Despliegue

El despliegue de esta aplicación tal y como especifica el documento de requisitos es muy simple, pues al tratarse de un aplicativo local en la máquina del docente y monousuario no se necesita planificar una "estrategia de despliegue" más allá de prever un asistente de instalación.

4.4. Modelo de datos

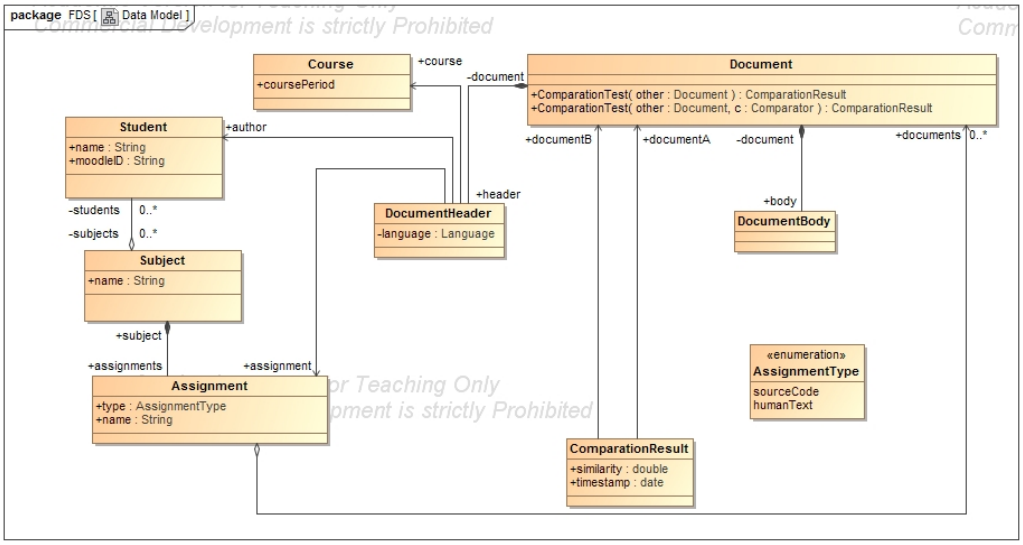

Figura 4.3: Modelo de datos

La clase más importante es el *Document*. Esta se ha decidido dividir en dos partes: *DocumentHeader* y *DocumentBody* para separar los datos que

identifican al documento en cuestión (su autor, el curso al que corresponde, el ejercicio, etc.) del contenido del mismo. Esto permite reutilizar la clase DocumentBody extendiéndola y dotándola de futuras funcionalidades como analizadores léxicos y sintácticos.

Una asignatura se define como el conjunto de ejercicios que la constituyen. Los ejercicios son independientes de los cursos, pero cada documento pertenece a un único curso, a un único estudiante y a un único ejercicio. De esta forma el docente podrá seleccionar todos los trabajos presentados por los estudiantes dentro de ese curso; o almacenarlos para futuras comparaciones.

Las comparaciones se hacen siempre entre dos documentos y retornan un resultado entre 0,0 y 1,0 calculado conforme a la fórmula expresada en el Documento de Requisitos.

Capítulo 5

Fase de desarrollo

5.1. Separación de las capas

Para facilitar la separación de las capas MVC (Modelo/Vista/Controlador) se han creado varios proyectos .NET (cada proyecto se corresponde con una libreria o un ejecutable).

FraudDetectionGUI (ejecutable): Constituye por si mismo toda la capa de la Vista y en caso de desarrollar una nueva vista para otras plataformas será el único proyecto (junto con MoodlePlugin) que habrá que portar. Consta de las clases: AssignmentEditForm, AssignmentMannagerForm, ComparationDetailsForm, DocumentComparationForm, DocumentView y MainForm. Todas ellas formularios (o vistas) del aplicativo.

DifferenceEngine (librería): Contiene la implementación del algoritmo Diff desarrollada por Michael Potter [14] bajo licencia GPL. Su librería se ha respetado sin modificarse, será distribuida bajo los mismos términos y cuando ha sido necesario modificar la entrada o salida de la misma se ha encapsulado bajo clases propias de este aplicativo.

DBController (librería): Posibilita el paso de datos entre el controlador de datos y la base de datos SQLite [16]. Si en un futuro se implementara otro sistema de base de datos únicamente sería necesario reemplazar esta librería.

MoodlePlugin (librería): Contiene todo el subsistema que permite descargar automáticamente trabajos de estudiantes subidos a la plataforma Moodle.

FDSKernel (libería): Esta libería contiene las partes del modelo y del controlador que han sido desarrolladas específicamente para este aplicativo. Sin embargo, las partes del controlador se han almacenado en ficheros de código separados del modelo de datos y además los *namespace* (similiares a los paquetes de Java) son distintos. A continuación se detallan los ficheros de código:

Comparators.cs: Pertenece al MODELO DE DATOS. Contiene la definición de un resultado de una comparación

DataController.cs: Pertenece al CONTROLADOR. Contiene la parte del aplicativo que gestiona el acceso a la BD y define las operaciones del gestor de objetos. Cualquier operación realizada por el usuario o por el gestor de trabajos pasa primero por el gestor de objetos, garantizando que se cumplen los requisitos del modelo.

DataModel.cs: Pertenece al MODELO DE DATOS. Define las clases del ejercicio, asignatura, curso y estudiante.

DocumentDataModel.cs: Pertenece al MODELO DE DATOS. Define las clases relacionadas con el documento.

Tools.cs: Pertenece al CONTROLADOR. Facilita herramientas y métodos para que el gestor de objetos y el gestor de trabajos pueda utilizar los algoritmos especificados.

WorkloadScheduler.cs: Pertenece al CONTROLADOR. Gestiona todas las operaciones en segundo plano y se encarga de enviar eventos a las vistas para indicar que ha finalizado los trabajos pendientes.

5.2. Satisfacción de los requisitos IS

Se han realizado pruebas de funcionamiento en las condiciones de plataforma pedidas en el Documento de Requisitos sin que se hayan detectado fallos gráficos o malfuncionamiento, por lo que se entiende que los requisitos sobre la plataforma de la aplicación están cubiertos.

Se ha buscado simplificar al máximo posible la interfaz para hacerla más amigable. Lo que facilita una buena experiencia de uso para el docente, puesto que tendrá las funciones que necesita siguiendo los pasos que la propia GUI le señala y con los documentos perfectamente organizados.

En la Figura 5.1 se tiene una vista principal de la aplicación. Se observa en el cuadro de diálogo adyacente cómo se especifican los parámetros al crear o modificar un ejercicio. La GUI tiene completo soporte para Inglés y Español

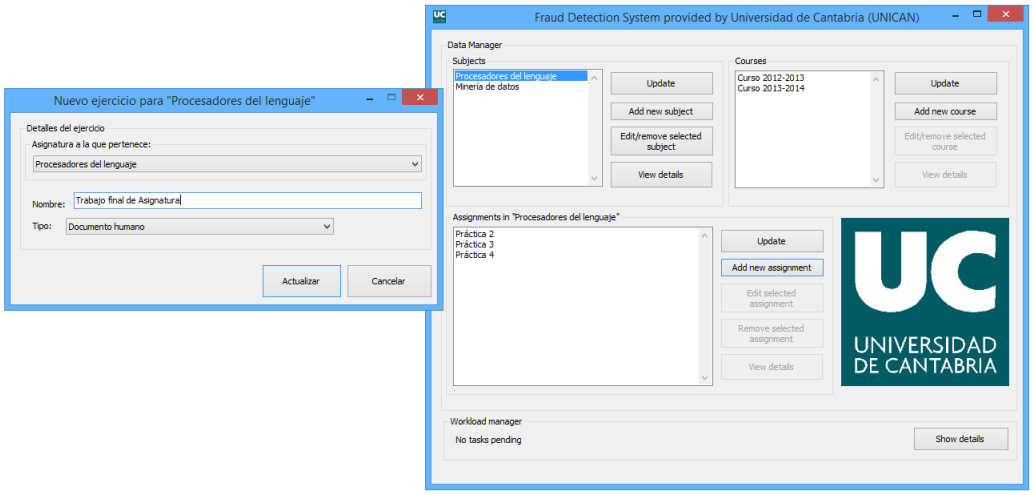

Figura 5.1: Vista principal de la aplicación.

pudiéndose fácilmente incorporar nuevos idiomas sin necesidad de modificar el código de la aplicación. La selección del idioma depende de la preferencia de idioma configurada en el entorno del sistema operativo base.

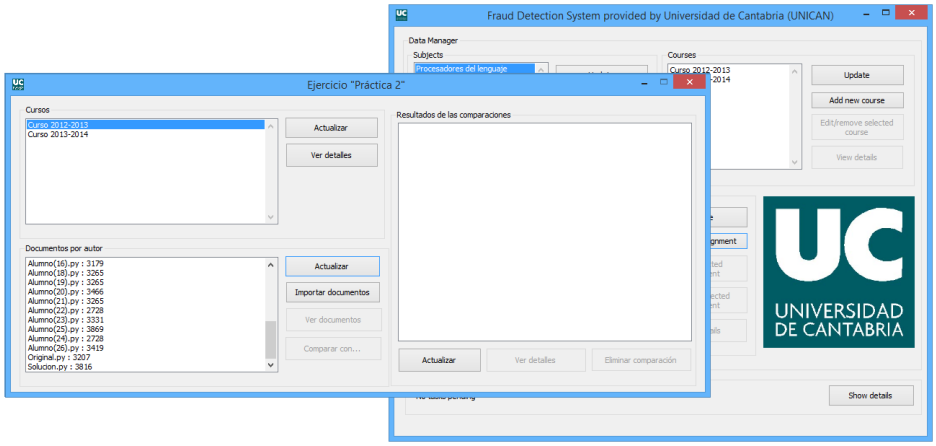

Figura 5.2: Formulario del ejercicio. Se visualizan documentos y comparaciones del mismo por cursos.

Como se observa en la Figura 5.2, dependiendo del curso seleccionado aparecen unos u otros documentos y comparaciones ya realizadas. La aplicación almacena todos los documentos que el docente haya importado a través de las tres opciones disponibles en el botón *Importar documentos*, ofreciéndose en todo momento la posibilidad de comparar documentos de otros cursos con los del curso actual tal y como se aprecia en la Figura 5.3. En defini-

tiva, el docente puede seleccionar qué documentos quiere analizar y contra qué documentos quiere hacerlo. Puede hacerlo contra documentos de ese año o de un año anterior. Las comparaciones se muestran en el cuadro blanco de la derecha ordenadas de mayor a menor grado de similitud.

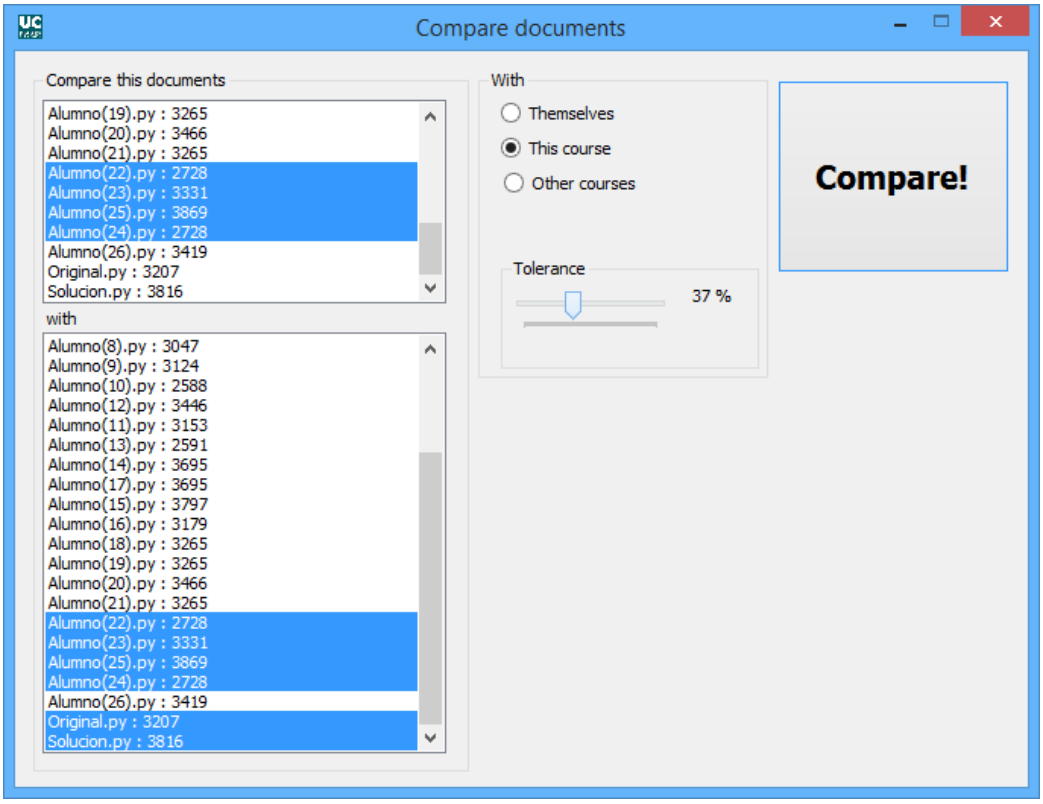

Figura 5.3: Ventana para efectuar las comparaciones.

Cada documento pertenece a un único trabajo. Es decir, si el trabajo original está realizado en otro formato que no sea texto plano hay que convertirlo primero a dicho formato. Si contiene varios ficheros hay que combinarlos en uno solo. La justificación es simple: a la hora de comparar no nos importa el formato (dos documentos con distinto formato e idéntico contenido para nosotros son idénticos). Esta es la razón de haber escogido texto plano como formato para el análisis del plagio.

Para las entregas de los trabajos de tipo código fuente se recomienda utilizar una plantilla del ejercicio con la solución escrita por el profesor de tal forma que el docente pueda utilizar la tasa de similitud obtenida entre la plantilla de control y esta plantilla como límite inferior a la hora de identificar plagios.

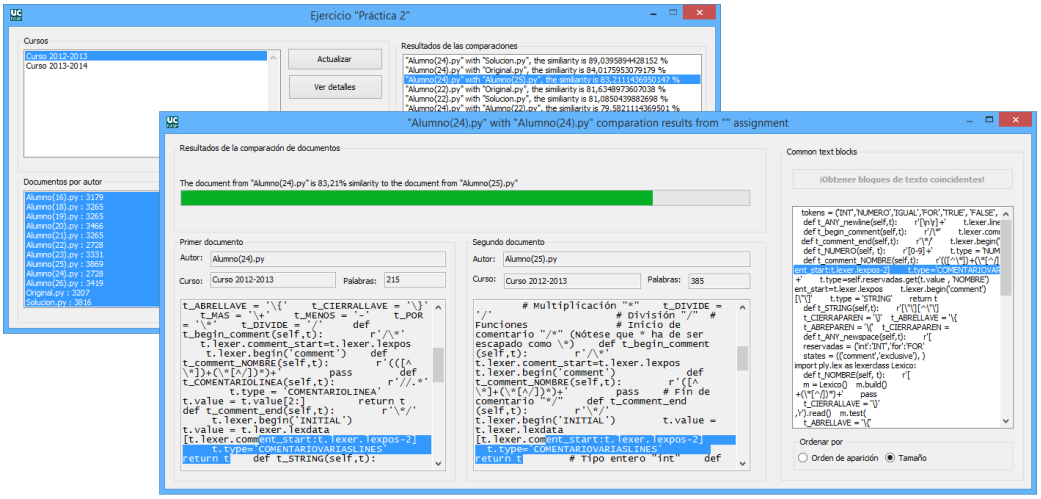

Figura 5.4: Cuadro de resultado de una comparación.

En la Figura 5.4 se aprecia el cuadro de resultado de comparación, en la lista de la parte de la derecha se puede seleccionar uno de los bloques comunes a ambos documentos y automáticamente se selecciona en cada documento el bloque común para que el docente pueda determinar si existe plagio entre los documentos.

5.3. Problemas encontrados

El principal problema que surgió durante la realización del proyecto es que aunque la API .NET sea muy completa ésta no dispone de un navegador "embebido" para poder realizar peticiones HTML por debajo. Únicamente cuenta con un navegador en la interfaz que, aunque permite realizar peticiones HTML automatizadas, éstas tienen que realizarse forzosamente sobre la vista dado que el control *WebBrowser* que la API nos facilita es de este tipo.

Otro problema surge a la hora de almacenar los resultados de una comparación. Pues lo "ideal" sería poder almacenar en la BD todos los bloques que hayan resultado ser iguales en ambos documentos. Pero no se puede hacer esto para todos los documentos puesto que la BD se saturaría y en la mayor parte de los casos este almacenamiento resultaría completamente inútil.

Finalmente, se ha comprobado que si se trata de comparar ficheros binarios, debido a su alto nivel de entropía ésta comparación tarda mucho tiempo en converger. Éste problema no sucede en ningun caso con ficheros que contienen texto en lenguaje humano.

5.4. Medidas adoptadas

Para el primer problema se ha optado por constituir un subsistema totalmente independiente para la parte del aplicativo que descarga los trabajos de los estudiantes de la plataforma Moodle, minimizando el impedimento intrínseco de la API .NET. Por desgracia, si alguien trata de portar la aplicación a otras plataformas ya mencionadas anteriormente tendrá que buscar una alternativa a esta funcionalidad. Este problema no afecta al resto del sistema.

Para el problema de almacenamiento de los bloques idénticos entre dos documentos se ha optado por no almacenarlas, calculando únicamente el índice de similaridad en un primer momento. Sin embargo, en el cuadro de comparación puede solicitarse que se carguen en memoria volátil los bloques comunes volviendo a llamar al algoritmo (en segundo plano) y únicamente para la comparación que se desea revisar.

Los ficheros binarios no suponen ningún problema porque no constituyen el propósito de este proyecto. Si el usuario "accidentalmente" solicita la comparación de un fichero binario no congelaría la interfaz principal porque la tarea quedaría ejecutándose en segundo plano hasta que convergiera el algoritmo.

Capítulo 6

Evaluación y pruebas

6.1. Origen del dataset empleado

Para la realización de las pruebas he de dar las gracias a mi directora de proyecto Cristina Tîrnăucă y a Domingo Gómez por facilitarme los trabajos entregados para correción y así poder evaluar los resultados arroyados por el aplicativo construido. Este dataset está compuesto por una serie de ejercicios variados de programación. Debido a la sensibilidad de la información, los autores de dichos trabajos, así como el curso y la asignatura a la que pertenecen serán totalmente ofuscados. Además en ningun caso se mostrará el contenido total de los trabajos; sólamente las partes necesarias para la validación de las pruebas.

También se han realizado pruebas con textos en lenguaje humano, aunque en este caso han sido generados para la ocasión.

6.2. Resultados obtenidos

6.2.1. Pruebas realizadas con lenguaje humano

Las pruebas realizadas con textos escritos en lenguaje humano han consistido en analizar un ejercicio sobre la biografía de Steve Jobs. Se han generado cuatro documentos:

Wikipedia.txt: Es el artículo original de la Wikipedia copiado y pegado de la web entero que utilizará el docente como referencia. Es el mayor de todos los ficheros.

Copiado.txt: Fragmento obtenido de la Wikipedia del cual se han eliminado únicamente partes que hacían que el texto no fuera presentable (por

ejemplo, el número de apartado que aparece en Wikipedia)

Cambiado.txt: Fragmento del texto copiado al cual se le han cambiado algunas palabras.

Reescrito.txt: Texto escrito directamente por el autor tras la lectura del artículo de la Wikipedia.

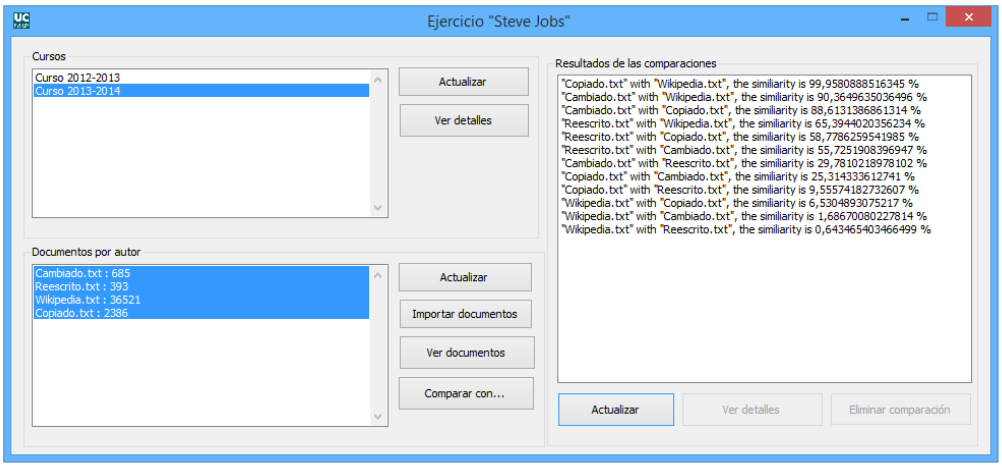

Figura 6.1: Resultados de la comparación de los cuatro documentos.

Como se puede apreciar en la Figura 6.1, el documento *Copiado.txt* se asemeja con la Wikipedia en casi un 100 %. El aplicativo ha sido capaz de identificarlo claramente como copia a pesar de los pequeños cambios de formato.

El documento *Cambiado.txt* también se identifica en un 90 % con el original y al visualizar la comparación en la Figura 6.2 se aprecian tres grandes bloques de texto copiados y varias frases copiadas y pegadas. Sin embargo, la semejanza de *Reescrito.txt* con el texto de la Wikipedia es solo del 65 %.

Finalmente, Wikipedia con respecto a todos los demás saca tasas de similitud bajas porque el tamaño del artículo original es mucho más amplio que los otros y la posible implicación del plagio en relación a su tamaño se minimiza.

6.2.2. Pruebas realizadas con lenguajes de alto nivel

Para la realización de las pruebas con lenguajes de programación de alto nivel se han utilizado trabajos reales.

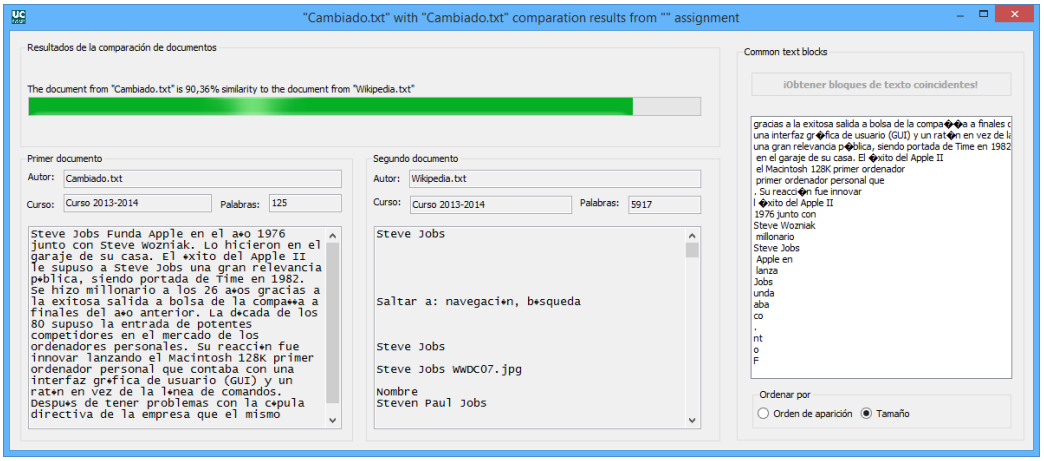

Figura 6.2: Resultados al comparar el documento "editado" con el de la Wikipedia

Lo primero que se deduce de los resultados que aparecen en la Figura 6.3 es que tres alumnos han entregado exactamente la misma práctica sin modificar (el 18, 19 y 21).

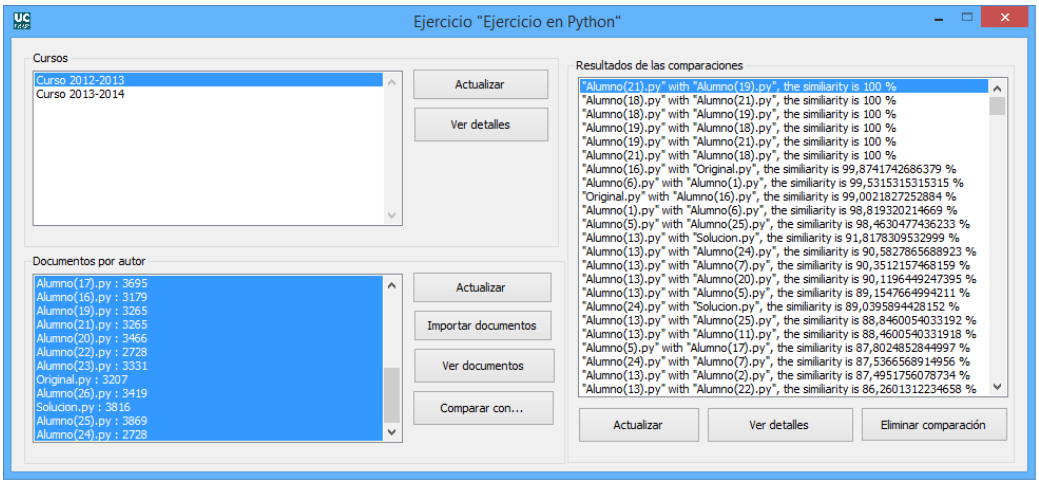

Figura 6.3: Resultados de la comparación de todos los trabajos de los alumnos.

El alumno 16 ha entregado la práctica sin hacer, siendo prácticamente la misma que el guión original. El alumno 6 y el alumno 1 también han copiado porque tienen índices de similaridad mutuos del orden del 100 %. Lo que contrasta bastante con el hecho de que el alumno que mayor semejanza ha obtenido con la práctica de control ha sido de un 91 %. Los alumnos que

se aproximaron a la solución correcta comparten índices de similaridad que oscilan entre el 90 % y el 80 %. Esto se debe a que los trabajos entregados en código fuente serán más similares que los trabajos en lenguaje humano.

Capítulo 7

Conclusiones generales y trabajos futuros

7.1. Portado a otras plataformas

Como señala el *Documento de Requisitos*, uno de los objetivos del aplicativo desarrollado es ser fácilmente portable a otros entornos modificando únicamente la parte de la vista. A la hora de llevar a cabo este proceso es recomendable repasar el diseño y código de la parte de la vista debido a que en ella se encuentra toda la lógica de la presentación de los datos y las llamadas al Controlador de Datos y al Planificador de Trabajos (ambos reutilizables al 100 % por cualquier entorno que implemente el estandar ECMA C# versión 4), así como al Plugin de Moodle.

El Plugin del Moodle es necesario volver a desarrollarlo para una nueva plataforma debido a su caracter monolítico adoptado por las razones ya mencionadas. Sin embargo, su integración con el resto del sistema está garantizada siempre y cuando se cumpla el parámetro de la llamada: *MoodlePlugin.exe [ruta_ficheros]*, siendo *[ruta_ficheros]* la ubicación temporal de descarga de los trabajos en Moodle.

En el caso de una implementación para interfaz web es conveniente tener en cuenta ciertos riesgos de seguridad. Pues el subsistema de descarga automatizada de Moodle tendría que descargar los ficheros en el lado del servidor, con el riesgo que esto supone si un usuario malintencionado solicita la descarga de ficheros potencialmente peligrosos.

7.2. Búsquedas de fuentes externas

Una extensión natural del aplicativo sería la posiblidad de contrastar los trabajos con fuentes externas tales como *GitHub* [17] o *dblp* [18]. Dicha funcionalidad sería constituida como un extension point del caso de uso *Comparar documentos* y constituiría por si mismo un subsistema. El objetivo sería reducir al máximo posible la cantidad de documentos a buscar en fuentes externas puesto que estos pueden ser cientos de miles de millones siendo la mayor parte irrelevantes. Una opción sería pedir al docente que introdujera una serie de palabras clave para realizar las comparaciones en los repositorios académicos contra los trabajos presentados o automatizar dicho proceso localizando las palabras más comunes entre todos los trabajos presentados y utilizarlas como palabras clave en las búsquedas contra los repositorios.

7.3. Análisis léxico y sintáctico

Como ya se ha mencionado en el *Capítulo 2* se distinguen dos tipos de análisis en el plagio. El explícito (más rápido) y el implícito (más profundo). En este proyecto se ha materializado el análisis explícito del plagio, pero el aplicativo se ha desarrollando pensando en futuras ampliaciones orientadas al análisis implícito del plagio. Por ejemplo, dando un peso a la estructura gramatical en la fórmula de similaridad se puede dar por "coincidente" palabras sinónimas cambiadas intencionadamente para despistar al docente.

En la parte de lenguajes de alto nivel el análisis implícito es muy útil, dado que con un analizador del léxico del lenguaje de programación se pueden "tokenizar" los elementos del código como los nombres de las variables y conservar únicamente la estructura del código de la misma forma que hace el compilador. De esta forma un cambio del nombre de una variable nuestro analizador lo considerará como coincidencia siempre y cuando la funcionalidad sea idéntica.

7.4. Conclusiones finales

Las herramientas de detección de plagio desarrolladas hasta el momento no han tenido en cuenta el análisis del plagio en ficheros de código fuente. Este proyecto ha tratado este problema, sin dejar de lado el análisis de textos convencionales y tras la culminación del mismo se ha construido un aplicativo de fácil uso para el docente que ha demostrado tener muy buenos resultados. Sin embargo, es importante tener en cuenta que al final, el que

determina qué estudiante ha cometido plagio es el docente. Cualquier herramienta, incluida esta, sólo puede suponer una ayuda y en ningun caso tener la última palabra. Por eso es importante buscar la manera de mostrar los datos más relevantes para el docente en dicho análisis y no limitarse a decir qué documentos son más similiares entre sí, mostrando, por ejemplo y como hace este aplicativo, aquellos puntos de los trabajos de los estudiantes que resulten comunes o similares.

Finalmente y para concluir, este proyecto ha supuesto un repaso muy completo la mayoría de las materias vistas durante la carrera. Desde las más básicas de programación donde aprendí los fundamentos de la programación orientada a objetos y la planificación de buenos algoritmos hasta asignaturas como Minería de Datos o Multiprocesadores, pasando por las asignaturas de Bases de datos que me ayudaron mucho a realizar buenas planificaciones de modelos de datos. También, la asignatura de Sistemas Operativos, la cual me dio una perspectiva sobre cómo trabajar con las llamadas al sistema y a realizar algoritmos concurrentes con memoria compartida. Por otro lado, la asignatura de Procesadores del lenguaje me ha servido para aprender sobre técnicas de análisis del lenguaje. Y finalmente, y no por ello menos importante, mencionar la asignatura de Ingienería del Software I, una de las más duras de la carrera. Ingienería del Software I no sólamente me enseño cómo se planifica adecuadamente cualquier proyecto de software que se plantée como objetivo; también me enseñó, junto a otras asignaturas como Bases de Datos Avanzadas o Modelos de Negocio y Estrategias Web a planificar mi propio tiempo y mis recursos a la hora de elaborar un trabajo de cualquier índole.

Sin todo el conocimiento que he ido adquiriendo desde que ingresé en esta titulación en Septiembre de 2008, éste trabajo no se hubiera podido llevar a la práctica.

Bibliografía

[1] Clifton Phua, Damminda Alahakoon, and Vincent Lee. Minority report in fraud detection: Classification of skewed data. 2004.

[2] *Turitin*. «http://www.turnitin.com/», consultado en Nov 2013.

[3] *iThenticate*. «http://www.ithenticate.com/», consultado en Nov 2013.

[4] *Plagiarism detect*. «http://plagiarism-detect.com/», consultado en Nov 2013.

[5] *Plagius*. «http://www.plagius.com/s/en/Default.aspx», consultado en Nov 2013.

[6] *Ephorus*. «https://www.ephorus.com/», consultado en Nov 2013.

[7] Maximiliano Zambonatto Pezzin. *Farejador de Plagio*. «http://www.farejadordeplagio.com.br/», consultado en Nov 2013.

[8] Jörgen Graf and Mathias Landhäußer. *JPlag*. «https://jplag.ipd.kit.edu/», consultado en Nov 2013.

[9] Mark McCrohon. *DOC Cop*. «http://www.doccop.com/index.html», consultado en Nov 2013.

[10] Lean Seaward and Stan Matwin. Intrinsic plagiarism detection using complexity analysis. 2009.

[11] David Maier. The complexity of some problems on subsequences and supersequences. 1978.

[12] Tartarus. *Snowball Stemmer*. «http://snowball.tartarus.org/», consultando en Nov 2013.

[13] Matthew Manela. *Diffplex*. «http://diffplex.codeplex.com/», consultado en Nov 2013.

[14] Michael Potter. *A Generic Reusable Diff Algorithm in C-II*. «http://www.codeproject.com/Articles/6943/A-Generic-Reusable-Diff-Algorithm-in-C-II/», consultado en Nov 2013.

[15] *Mono Project*. «http://www.mono-project.com/», consultado en Feb 2014.

[16] D. Richard Hipp. *SQLite*. «http://www.sqlite.org/», consultado en Feb 2014.

[17] *GitHub*. «https://github.com/», consultado en Feb 2014.

[18] Marcel R. Ackermann Oliver Hoffmann Michael Wagner Stefanie von Keutz Michael Ley, Marc Herbstritt and Katharina Hostert. *dblp*. «http://www.informatik.uni-trier.de/~ley/db/», consultado en Feb 2014.