Master Universitario en Investigación en Ingeniería del Software y Sistemas Informáticos

Desarrollo de un DSL para el reconocimiento óptico de caracteres en el ámbito de la gestión documental

---

Trabajo de fin de máster

Itinerario de Ingeniería del Software (Código 31105128)

AUTOR: Javier Escribano García

DIRECTOR: Rubén Heradio Gil

Curso Académico 2016-17

Convocatoria de Septiembre

Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos

ITINERARIO: Ingeniería del Software

CÓDIGO DE ASIGNATURA: 31105128

TÍTULO DEL TRABAJO: Desarrollo de un DSL para el reconocimiento óptico de caracteres en el ámbito de la gestión documental.

TIPO DE TRABAJO: Tipo B, proyecto específico propuesto por el alumno.

AUTOR: Javier Escribano García

DIRECTOR: Rubén Heradio Gil

DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO CIENTÍFICO, PARA LA DEFENSA DEL TRABAJO FIN DE MASTER

Fecha: 11/09/2017

Quién suscribe:

Autor(a): Javier Escribano García
D.N.I/N.I.E/Pasaporte.: 45843254E

Hace constar que el autor del trabajo:

Desarrollo de un DSL para el reconocimiento óptico de caracteres en el ámbito de la gestión documental

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores, se han referenciado debidamente en el texto de dicho trabajo.

DECLARACIÓN:

- ✓ Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.
- ✓ Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- ✓ No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Fdo. Javier Escribano

IMPRESO TFDM05_AUTOR
AUTORIZACIÓN DE PUBLICACIÓN
CON FINES ACADÉMICOS

UNED ETS de Ingeniería Informática

Impreso TFdM05_Autor. Autorización de publicación

y difusión del TFdM para fines académicos

Autorización

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los Autor/es

Javier Escribano

Juan del Rosal, 16
28040, Madrid

Tel: 91 398 89 10
Fax: 91 398 89 09

www.issi.uned.es

Resumen

Una de las partes más importantes en la implantación de una gestión documental basada en web, es el reconocimiento óptico de caracteres (OCR) [1] de los documentos para clasificar los archivos automáticamente mediante la interpretación de estos.

Definiremos y desarrollaremos un lenguaje específico de dominio que nos permita facilitar la clasificación automática de documentos (*principalmente facturas de proveedores*). De esta manera, en base a una entrada de documentos digitalizados (*estos documentos siguen un orden lógico en el proceso de digitalización*), se obtendrá un documento PDF identificado que sea útil al cliente, ya sea para guardar y consultar el documento o integrarlo con otras plataformas (*software de gestión documental, planificadores de recursos empresariales, aplicaciones de terceros relacionados con la gestión documental, etc..)*.

La identificación de los documentos se realizará mediante la lectura OCR de campos clave o mediante la lectura de un código de barras introducido en la primera página del documento a clasificar.

Para poder realizar la tarea de identificar campos por reconocimiento óptico de caracteres utilizaremos el motor OCR de Tesseract y lo ampliaremos con una serie de funcionalidades que permitan el tratamiento del conjunto total de documentación.

Palabras clave: Lenguaje específico de dominio, gestión documental, reconocimiento óptico de caracteres, clasificación automática, .NET, C#, Tesseract.

Índice

Listado de figuras y tablas

Figuras

Tablas

1. Introducción

Una de las tareas fundamentales en los procesos de gestión documental web es la digitalización de los documentos para el almacenamiento de estos en un entorno web. Dicha digitalización se suele realizar con máquinas multifunción de alto rendimiento. Resulta imprescindible poder saber de que tipología es cada documento, las hojas por las que está compuesto y cuál es el identificador que lo distingue del resto.

Generalmente, las personas dedicadas a digitalizar estos lotes de documentos deben preocuparse de la calidad del papel (*que no tenga arrugas, que no tenga elementos que eliminen información importante, que el escáner esté en las condiciones adecuadas para no añadir suciedad en la imagen, que la digitalización esté correctamente configurada a nivel de brillo, densidad, dpi, etc…*).

Al introducir los lotes de papel en el escáner, probablemente se incluirán facturas, albaranes, notas… todo en orden lógico. El escáner digitalizará estos archivos y dejará en una carpeta cada imagen escaneada siguiendo el orden introducido en el alimentador. Es tarea nuestra identificar cada documento, agruparlo y crear un PDF distintivo del resto usando la información de cada documento digitalizado. Aquí se encuentra la clave nuestro proyecto.

Mediante un DSL estableceremos pautas para indicar a que tipología de documento (*a partir de ahora lo llamaremos "plantilla")* pertenece cada hoja y, dentro de esta plantilla, saber cuál es el campo identificativo. Este campo identificativo será un **campo del propio documento** (*por ejemplo, el número de la factura*) o un **código de barras** incluido en la primera página de cada documento. Ambas formas de identificación del documento son las más utilizadas en el ámbito de la gestión documental web.

Por lo tanto, seremos capaces de distinguir los archivos digitalizados, reagruparlos en un documento PDF acorde a la plantilla a la que pertenecen y darles un ID que posteriormente permita la clasificación de este.

Para la distinción y reagrupación de dichas imágenes, utilizaremos el motor de reconocimiento óptico de caracteres **Tesseract**. Tesseract es un motor OCR gratuito con licencia Apache 2.0. Ampliaremos las funcionalidades que nos otorga Tesseract para ser capaces de reconocer los campos del documento que nos permitan identificarlo y clasificarlo.

1.1 Ámbitos del proyecto y objetivos

El ámbito de la gestión documental web es muy extenso, existe una gran casuística, por lo que vamos a acotar el ámbito de nuestro proyecto de forma que contemplemos los casos más cotidianos y establezcamos una base sobre la que podamos construir una herramienta que contemple un gran abanico de opciones.

Nuestra materia prima va a ser las imágenes digitalizadas. Contaremos con que estas tendrán el formato PNG, tendrán una resolución mínima de 300dpi y trabajaremos con documentos en las que el texto esté lo suficientemente claro (*grosor del texto, ausencia de ruido o elementos que ensucien la imagen, rotación adecuada…*) para el motor de reconocimiento óptico de caracteres (*parte de estos requisitos forman parte del plan de gestión de calidad estándar de las aplicaciones de gestión documental comerciales*).

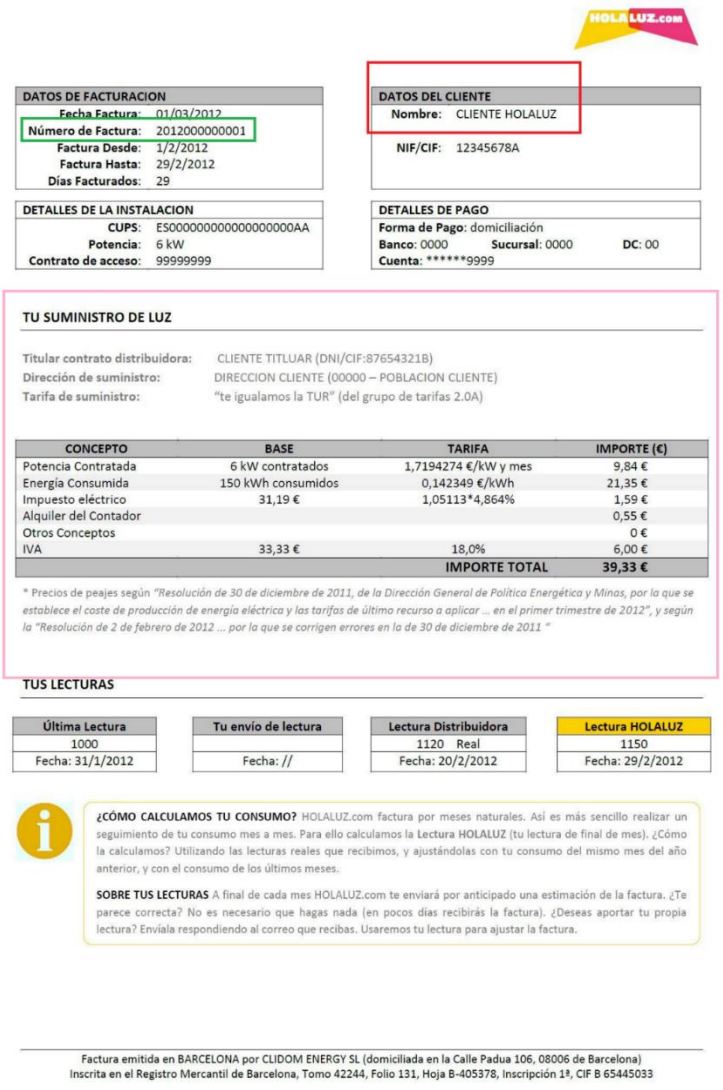
Además, todo el posible lote de imágenes para un DSL establecido deberán compartir las propiedades de digitalización *(si se ha escogido 300dpi, en blanco y negro y un brillo del 5%, todas las imágenes deben compartir las mismas propiedades*). Un cambio de estas propiedades implicará un cambio en el DSL.

No vamos a entrar en la optimización de lectura de caracteres ni en el training para conseguir resultados buenos con textos menos legibles.

Nuestro objetivo es elaborar un lenguaje que permita facilitar la automatización y clasificación de archivos sin entrar en el campo del preprocesamiento de imágenes.

Para identificar los documentos vamos a considerar que la información identificativa del documento se encuentra en la primera página de este (caso más común).

Por ejemplo, la siguiente factura presentada en la figura 1 (*todas las facturas incluidas en la memoria no tienen validez legal, ya que están creadas para propósitos de desarrollo*):



HOLALUZ.com

DATOS DE FACTURACION
Fecha Factura: 01/03/2012
Número de Factura: 2012000000001
Factura Desde: 1/2/2012
Factura Hasta: 29/2/2012
Días Facturados: 29

DATOS DEL CLIENTE
Nombre: CLIENTE HOLALUZ
NIF/CIF: 12345678A

DETALLES DE LA INSTALACION
CUPS: ES0000000000000000000AA
Potencia: 6 kW
Contrato de acceso: 99999999

DETALLES DE PAGO
Forma de Pago: domiciliación
Banco: 0000 Sucursal: 0000 DC: 00
Cuenta: ******9999

TU SUMINISTRO DE LUZ

Titular contrato distribuidora: CLIENTE TITLUAR (DNI/CIF:87654321B)
Dirección de suministro: DIRECCION CLIENTE (00000 – POBLACION CLIENTE)
Tarifa de suministro: "te igualamos la TUR" (del grupo de tarifas 2.0A)

CONCEPTO	BASE	TARIFA	IMPORTE (€)
Potencia Contratada	6 kW contratados	1,7194274 €/kW y mes	9,84 €
Energía Consumida	150 kWh consumidos	0,142349 €/kWh	21,35 €
Impuesto eléctrico	31,19 €	1,05113*4,864%	1,59 €
Alquiler del Contador			0,55 €
Otros Conceptos			0 €
IVA	33,33 €	18,0%	6,00 €
		IMPORTE TOTAL	39,33 €

* Precios de peajes según "Resolución de 30 de diciembre de 2011, de la Dirección General de Política Energética y Minas, por la que se establece el coste de producción de energía eléctrica y las tarifas de último recurso a aplicar ... en el primer trimestre de 2012", y según la "Resolución de 2 de febrero de 2012 ... por la que se corrigen errores en la de 30 de diciembre de 2011 "

TUS LECTURAS

Última Lectura	Tu envío de lectura	Lectura Distribuidora	Lectura HOLALUZ
1000		1120 Real	1150
Fecha: 31/1/2012	Fecha: //	Fecha: 20/2/2012	Fecha: 29/2/2012

¿CÓMO CALCULAMOS TU CONSUMO? HOLALUZ.com factura por meses naturales. Así es más sencillo realizar un seguimiento de tu consumo mes a mes. Para ello calculamos la **Lectura HOLALUZ** (tu lectura de final de mes). ¿Cómo la calculamos? Utilizando las lecturas reales que recibimos, y ajustándolas con tu consumo del mismo mes del año anterior, y con el consumo de los últimos meses.

SOBRE TUS LECTURAS A final de cada mes HOLALUZ.com te enviará por anticipado una estimación de la factura. ¿Te parece correcta? No es necesario que hagas nada (en pocos días recibirás la factura). ¿Deseas aportar tu propia lectura? Envíala respondiendo al correo que recibas. Usaremos tu lectura para ajustar la factura.

Factura emitida en BARCELONA por CLIDOM ENERGY SL (domiciliada en la Calle Padua 106, 08006 de Barcelona)
Inscrita en el Registro Mercantil de Barcelona, Tomo 42244, Folio 131, Hoja B-405378, Inscripción 1ª, CIF B 65445033

Figura 1- Formato estándar de factura

En esta factura observamos 3 bloques identificados con cuadros de diferentes colores:

- **Cuadro rojo**: En esta sección se encuentra un campo identificativo de la tipología de la factura. En este caso se trata de una factura del cliente Hola Luz.
- **Cuadro verde:** En esta sección podemos ver el número de la factura. Este campo no es identificativo del tipo de factura, pero si lo es de la factura concreta para un tipo dado. Es decir, con este elemento no podemos saber si la factura de "HolaLuz" o de Telefónica, pero si podemos saber que, dentro de las facturas del tipo HolaLuz, es una factura única e identificada.
- **Cuadro rosa**: Este es el cuerpo de la factura. Dicho cuerpo puede extenderse en varias hojas. Por ejemplo, las líneas de una factura pueden ser lo suficientemente grandes para ocupar 5 hojas. En estas hojas generalmente no hay información útil para identificar el documento, pero si tendremos que tenerlas en cuenta a la hora de formar el PDF final.

Como deducimos, usaremos la primera hoja para identificar la factura e identificarla. El resto de hojas se prepararán para ser agrupadas junto con la primera en un PDF.

Nuestra tarea entonces se convierte en establecer una serie de campos que nos permitan identificar cual es el comienzo de una factura y cuando termina. Consideraremos que una factura termina cuando encontramos otra factura (con sus campos clave identificados) o cuando no quedan más hojas.

En la figura 2 exponemos de forma muy clara lo que hemos explicado. Escaneamos un conjunto de facturas tal y como se muestra a continuación:

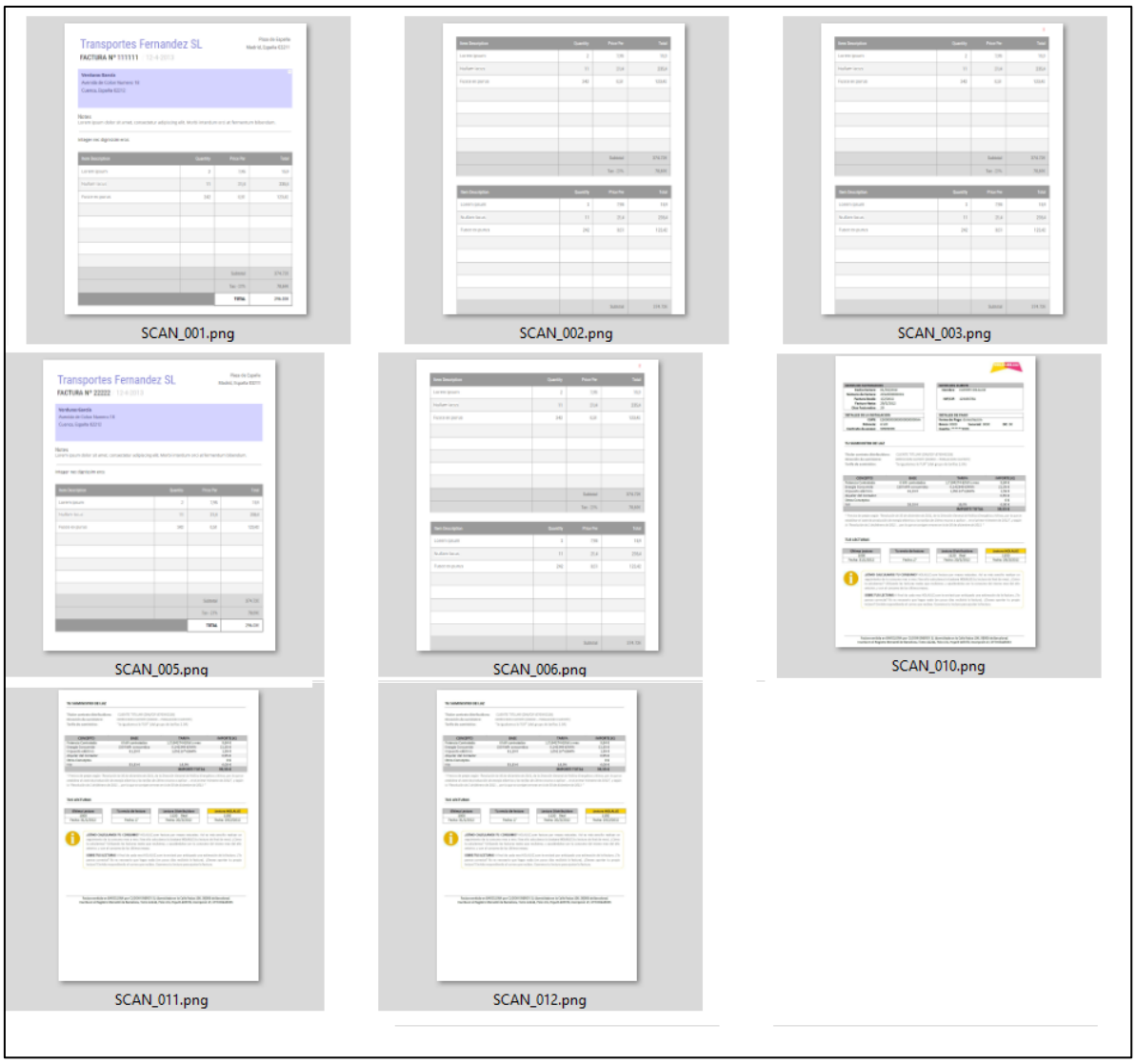


Figura 2. Conjunto de facturas escaneadas

Supongamos que hemos digitalizado 3 facturas. Estas se estructuran como sigue:

FACTURA 1 de TIPO A: SCAN_001.png, SCAN_002.png, SCAN_003.png.

FACTURA 2 de TIPO A: SCAN_005.png, SCAN_006.png.

FACTURA 3 de TIPO B: SCAN_0010.png, SCAN_011.png, SCAN_012.png.

Si nos damos cuenta, las imágenes escaneadas SCAN_001, 005 y 0010 son la primera página de sus respectivas facturas y por tanto las que contendrán la información esencial para su identificación.

Nuestro objetivo, por tanto, consiste en detectar esas imágenes, identificarlas mediante el uso de nuestro DSL y agruparlas en PDF's tal y como sigue:

1- En la figura 3 de muestra la carpeta de salida con una subcarpeta por tipología de factura

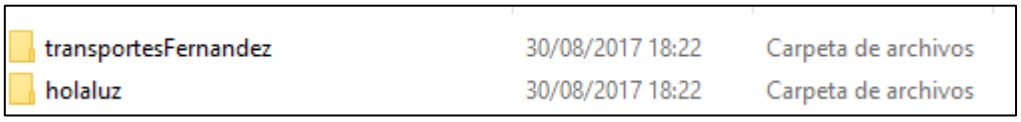


Figura 3. Carpetas ordenas por tipología

2- PDF por cada factura agrupada en la carpeta correspondiente. El nombre del PDF corresponde al campo identificativo del documento. En la figura 4 vemos la sección donde se localiza el número de factura del tipo B, es decir, el campo identificativo. En las figuras 5 y 6 los identificadores para las facturas de tipo A.

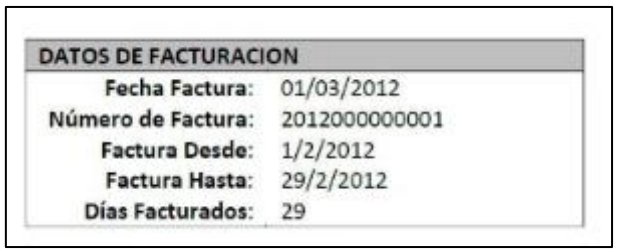


Figura 4. Campo identificativo factura 3

Figura 5. Campo identificativo factura 1

Figura 6. Campo identificativo figura 2

En la figura 7 observamos cómo se han divido las imágenes digitalizadas en agrupaciones de documentos PDF en base a las páginas que lo conforman y renombrado por el identificador correspondiente.

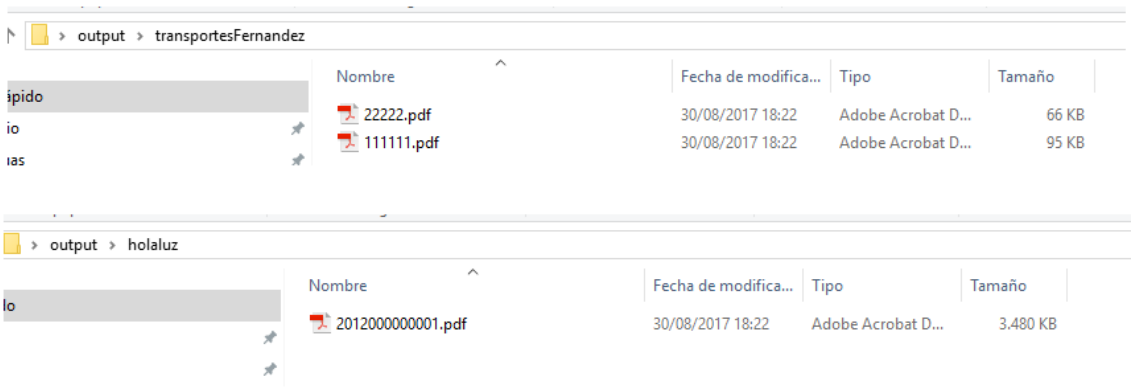


Figura 7. Nombrado del documento PDF y facturas identificadas y ordenadas por carpeta

3- En la figura 8 vemos un ejemplo de PDF tras la ejecución del proceso. Cada PDF contiene todas sus páginas de forma ordenada y renombrada con el identificador capturado por OCR.

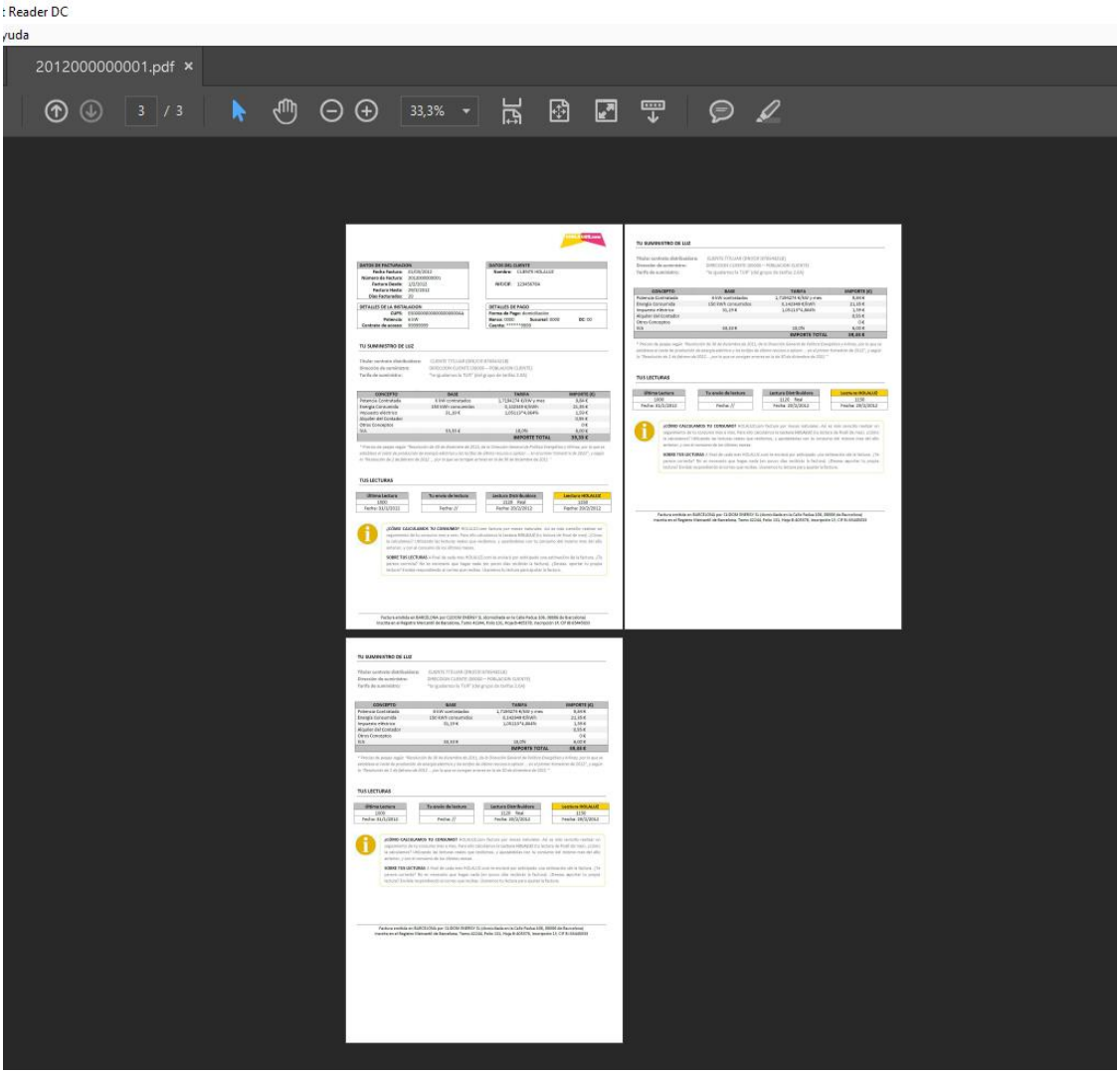


Figura 8. Ejemplo del PDF de la factura 3 renombrado y agrupado.

Visto el ejemplo descriptivo, establecemos, por tanto, la siguiente lista de objetivos:

1- Procesar una serie de imágenes con la suficiente calidad como para reconocer mediante OCR los campos identificativos

2- Reagrupar las imágenes en base a su tipología y generar un PDF nombrado con un campo identificativo del documento.

3- Capacidad para leer los campos, aunque el escaneo los mueva, es decir, ser capaz de reconocer de forma flexible el campo que identifica el documento sin utilizar únicamente regiones estáticas que puedan verse alteradas al escanear el documento "un poco torcido" o algo "escorado".

4- Gestionar los documentos fallidos, es decir, aquellos que no se han podido reconocer, moviéndolos a una carpeta que el usuario podrá monitorizar

5- Dejar archivos de log para verificar y comprobar el proceso seguido.

Cabe destacar la siguiente lista de **potenciales** objetivos (*ver futuras líneas de investigación*) que quedan fuera del alcance del este proyecto:

1- Mejorar la imagen mediante algoritmos de preprocesamiento y así obtener un mejor resultado en el reconocimiento textual.

2- Leer múltiples campos para obtener más información acerca del documento (*fecha de la factura, numero del proveedor, etc..*).

3- Tratar documentos en otro formato que no sea png.

4- Obtener todo el texto del documento para fines relacionados con la gestión documental (*búsqueda por contenido*).

5- Obtener campos identificativos que no estén en la primera página.

6- Leer códigos de barras en otros formatos a los establecidos.

7- Procesar documentos en distinta lengua al castellano.

2. Estado del arte

Actualmente muchas empresas han implantado la gestión documental web [2]. Esto ha conllevado una serie de cambios a la hora de procesar los documentos, almacenarlos y consultarlos.

La gestión documental es el conjunto de tecnologías, normas y técnicas que permiten administrar el flujo de documentos de cualquier empresa u organización. Ello implica agilizar la localización y tramitación de los mismos en base a su información, vincularlos entre sí, compartirlos o determinar su tiempo de conservación.

Para automatizar y simplificar dichas existen diversas aplicaciones y servicios software que aplican todo lo necesario para cumplir con los objetivos de la gestión documental web, ahorrando tiempo, recursos y espacio. [3]

2.1 Criterios en la gestión documental web

Existen diversos criterios esenciales a la hora de trabajar con una aplicación de gestión documental web [4]. Estos se presentan en la Tabla 1.

Almacenamiento	¿Dónde guardaremos nuestros documentos? ¿Cuánto podemos pagar para almacenarlos?
Recuperación	¿Cómo puede la gente encontrar documentos necesarios? ¿Cuánto tiempo se puede pasar buscándolo? ¿Qué opciones tecnológicas están disponibles para la recuperación?
Clasificación	¿Cómo organizamos nuestros documentos? ¿Cómo aseguramos que los documentos estén archivados siguiendo el sistema más apropiado?
Seguridad	¿Cómo evitamos la pérdida de documentos, evitar la violación de la información o la destrucción no deseada de

	documentos? ¿Cómo mantenemos la información crítica oculta a quién no debiera tener acceso a ella?
Custodia	¿Cómo decidimos qué documentos conservar? ¿Por cuánto tiempo deben ser guardados? ¿Cómo procedemos a su eliminación (expurgo de documentos)?
Distribución	¿Cómo distribuimos documentos a la gente que la necesita? ¿Cuánto podemos tardar para distribuir los documentos?
Workflow	¿Si los documentos necesitan pasar a partir de una persona a otra, cuáles son las reglas para el flujo de estos documentos?
Creación	¿Si más de una persona está implicada en creación o modificación de un documento, cómo se podrá colaborar en esas tareas?
Autenticación	¿Cómo proporcionamos los requisitos necesarios para la validación legal al gobierno y a la industria privada acerca de la originalidad de los documentos y cumplimos sus estándares para la autentificación?

Tabla 1. Criterios en la gestión documental web

El ámbito de nuestro proyecto se centra (*directa e indirectamente*) en los tres primeros puntos: almacenamiento, recuperación y clasificación.

Generalmente existen dos vías principales de obtener documentos útiles para nuestra gestión documental web:

- Documentos digitales: Cualquier documento digital (email, aplicaciones de terceros, documento previamente digitalizado, etc...).

- Documentos en papel: Estos documentos constituyen hoy en día el grueso de la documentación de la pequeña y mediana empresa.

Sea cual sea la tipología de documento con la que tratemos, necesitaremos poder clasificarlos de forma que podemos posteriormente recuperarlos de la forma más eficiente posible, ya que constituye una tarea muy importante en el día a día de las empresas.

Aproximadamente entre un 14%-20% del día de un empleado se invierte en buscar documentos y modificarlos. [5]

2.2 Clasificación de un documento

En las aplicaciones de gestión documental web, una de las tareas a automatizar de la mejor forma posible es la de procesar los documentos, en formato digital o papel, identificarlos y clasificarlos de forma que lléveme el menor trabajo manual posible y, sobre todo, el menor índice de error. Existen 3 formas de clasificar documentos empresariales:

- Manualmente: Este ha sido el proceso más aplicado en años anteriores, donde habían empleados que identificaban y clasificaban el papel en archivadores de forma manual. Estos archivadores se ordenaban y se marcaban para intentar reducir el tiempo de acceso a ellos. Esta forma de gestionar la documentación conlleva diversos problemas obvios de almacenamiento, mantenimiento, recuperación y control.
- Automatizada: Existen varias herramientas que reconocen el documento por OCR y usan su semántica para clasificarlos automáticamente en la aplicación de gestión documental. El empleado solamente tiene que proveer la documentación al aplicativo, ya se proporcionando la fuente digital de los archivos o bien digitalizando el papel.

Pese a que se intenta optimizar e implantar este tipo de soluciones, ya que a priori son las más optimas en cuanto a dinero/tiempo, la

tecnología de reconocimiento de texto y la inteligencia artificial aún no es capaz de resultar útil/económica a la pequeña y mediana empresa. Resulta muy complejo clasificar automáticamente los documentos sin la intervención de un usuario y que esta clasificación sea la más eficiente para el uso diario por parte de los trabajadores.

- Híbrida: **Nuestro proyecto se sitúa aquí**, en la identificación y clasificación de documentos de forma híbrida. Es una mezcla de las dos anteriores reduciendo los problemas de la gestión manual y aprovechando la tecnología de OCR. Muchas empresas dedicadas a la gestión documental web optan por esta solución.

  Básicamente se trata de identificar campos esenciales en los documentos para poder clasificarlos y posteriormente reconocer estos campos mediante OCR. Previamente, la información detallada de cada documento se ha introducido en un ERP, de forma que la empresa ya dispone de una fuente de información. El único paso que se necesita es vincular el papel a esa fuente de información, y para ello tratamos de establecer campos de nexo que automáticamente vinculen la fuente de papel/digital con la información acerca del documento disponible en la empresa.

  Para poder obtener este campo identificativo, y en el caso de las facturas de proveedores, se suelen usar dos vías:

  -Hibrida: **Lectura de un campo presente en el documento**: En el caso de los documentos que vayan a tener una buena calidad de digitalización, el papel se conserve en buen estado y/o el archivo digital es fácilmente reconocible *(entendemos por fácilmente reconocible al hecho de que la información se presente clara, sin ruido, sin suciedad y de forma localizada)* trataremos de leer mediante OCR dicho campo y obtener su valor. Suele ser ampliamente utilizado el número de la factura y algún campo que identifique la tipología del documento.

Si para una factura de proveedores de Telefónica somos capaces de encontrar "Telefónica", "Factura" y "Numero de factura: 0001" seremos de establecer la tipología de la factura (*en este caso Telefónica*) y el número identificativo (*el 0001*). De esta forma se vinculará el 0001 con los datos que tenga la empresa para así finalmente obtener todos los metadatos asociados a esa factura desde la base de datos, eliminando errores OCR y aumentando la coherencia entre el documento y la información empresarial.

Resumiendo:

1- Se detecta el tipo de documento mediante OCR
2- Se detecte el campo identificativo del documento mediante OCR
3- Se identifica este documento (*generalmente un PDF*)
4- La herramienta encargada de procesar este artefacto vinculará el campo identificativo con la base de datos para obtener el resto de campos (*cliente, fecha de la factura, líneas de la factura, totales, IVAs, bases imponibles, etc…)*

-Hibrida: **Uso de código de barras:** En los casos en los que la fuente no sea lo suficientemente clara o estructurada como para ser capaces de leer un campo se opta por la inclusión de un código de barras. Este código de barras puede venir directamente en el documento o "pegarse" en la primera hoja, de forma que reducimos bastante la complejidad en la lectura OCR porque la aplicación solamente tiene que leer un código de barras con la información necesaria.

Si tenemos una serie de facturas de Empresa García SL y no tienen las condiciones necesarias para leer el campo de numero de factura, previamente a la digitalización del papel, se pega un código de barras en la primera página de cada

factura y así tenemos un campo altamente legible que permita identificar, trocear y clasificar el lote de facturas.

En el propio código de barras podría introducirse mediante una clave el tipo de documento y su identificador.

Hemos incluido a continuación una imagen descriptiva de ambos procesos (*ver figura 9*) que permita la compresión del proceso que pretendemos tratar:

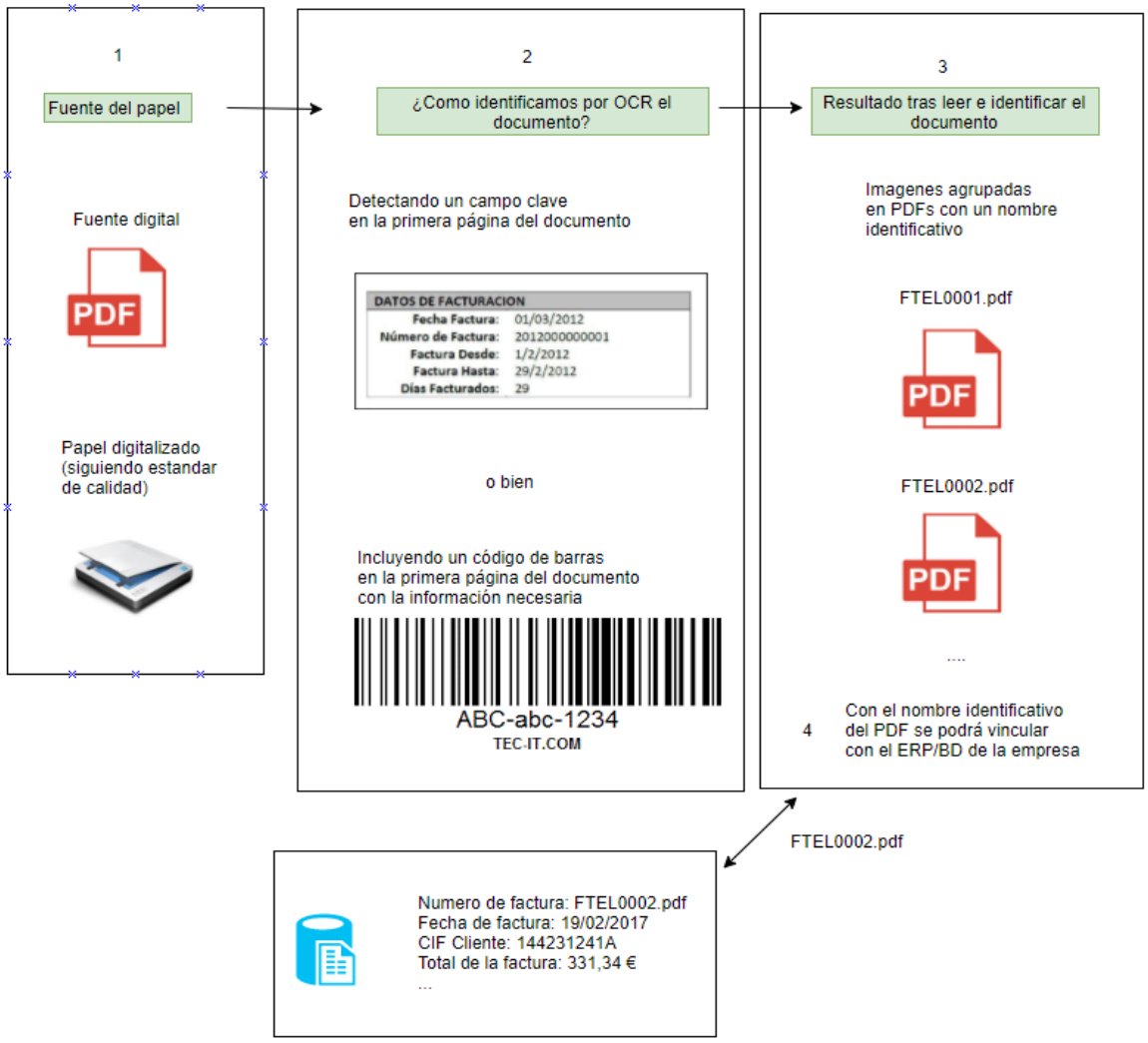


Figura 9 Proceso híbrido de clasificación documental

Como podemos observar, todo se reduce a identificar el documento para reconocer que tipo es, que páginas tiene y producir un artefacto que pueda seguir el flujo de la aplicación de gestión documental web.

2.3 Motores OCR

Existen varios motores OCR actualmente. Básicamente lo más importante a la hora de seleccionar uno es su licencia, si tiene un buen soporte, si ofrece un SDK para nuestro lenguaje de trabajo y, sobre todo, que sea capaz de tratar con nuestra tipología de documentos de forma adecuada. A continuación, presentamos en la figura 10 los principales motores del mercado en 2017 [6]:

Name	Latest stable version	License	Windows	Mac OS X	Linux	Programming language	SDK?	Languages	Output Formats
Tesseract	3.04.01	Apache	Yes	Yes	Yes	C++, C	Yes	100+[1]	Text, hOCR,[2] PDF, others with different user interfaces[3] or the API
ABBYY FineReader	14	Proprietary	Yes	Yes	Yes	C/C++	Yes	192[8]	DOC, DOCX, XLS, XLSX, PPTX, RTF, PDF, HTML, CSV, TXT, ODT, DjVu, EPUB, FB2[9]
Asprise OCR SDK	15	Proprietary	Yes	Yes	Yes	Java, C#,VB.NET, C/C++/Delphi	Yes	20+[11]	Plain text, searchable PDF, XML[12]
AnyDoc Software	?	Proprietary	Yes	No	No	VBScript	?	?	
LEADTOOLS[16]	19.0	Proprietary	Yes	Yes	Yes	C/C++, .NET, Objective-C, Java, JavaScript	Yes	56[18]	PDF, PDF/A, DOC, DOCX, XLS, XPS, RTF, HTML, ANSI Text, Unicode Text, CSV[19]
CuneiForm	1.1	BSD variant	Yes	Yes	Yes	C/C++	Yes	28	HTML, hOCR, native, RTF, TeX, TXT[22]
Dynamsoft OCR SDK	8.2	Proprietary	Yes	No	No	C/C++	Yes	40+[23]	PDF, TXT
OmniPage	19.2	Proprietary	Yes	Yes	No	C/C++, C#[24]	Yes	125[25]	DOC/DOCX XLS/XLSX PPTX RTF PDF PDF/A Searchable PDF HTML Text XML ePUB MP3
FreeOCR	4.2	Proprietary	Yes	No	No	?	?	?	
GOCR	0.50	GPL	Yes	Yes	Yes	C	?	?	
Puma.NET	?	BSD	Yes	No	No	C#	Yes	28	
OCRFeeder	0.7.11	GPL	No	No	Yes	Python	?	?	
OCRopus	1.0	Apache	No	No	Yes	Python	?	?	hOCR, HTML, TXT[29]

Figura 10 Principales motores OCR

Entre los motores más destacables se encuentran Tesseract y Abbyy:

Tesseract: El motor de Tesseract [7] fue creado por HP y actualmente estaba bajo desarrollo de Google. La licencia es Apache y además es multiplataforma. Dispone de varios SDK para varios lenguajes, una comunidad detrás que ha implementado varias soluciones usando este motor y además un sistema de entrenamiento (*serie de procesos que permiten preparar Tesseract para un conjunto de caracteres específico*) para poder reconocer diversas fuentes y lenguajes. Ofrece un gran rendimiento y es altamente personalizable. Sin embargo, no ofrece la gama de productos de Abbyy [8].

Abbyy : Empresa multinacional de Moscú. Una de las soluciones comerciales de OCR más utilizadas. Tiene 10 oficinas regionales y más de 1000 empleados en todo el mundo. El número total de usuarios de productos y tecnologías de la empresa supera 30 millones en más de 132 países del mundo. Ofrece varios productos para facilitar y permitir el trabajo con documentos de forma visual, es decir, con interfaces que asisten al desarrollo. Los más importantes son FineReader [9] y Flexicapture [10].

- FineReader: Es un sistema de reconocimiento óptico de caracteres (OCR). El producto convierte fácilmente cualquier tipo de PDF, documentos digitales y ficheros de imagen, incluyendo las fotos digitales, en documentos editables y localizables.
- FlexiCapture: es un software de captura de datos escalable para varios tipos de documentos (*especialmente facturas, contratos, formularios, cuestionarios, etc…)*. Los escenarios de uso principal de este software son la extracción de datos, indexación y clasificación de documentos, archivado de imágenes y PDFs searchable (*PDF con capa de texto para copiar, pegar, buscar etc.*.). En la figura 11 vemos una captura en la elaboración de una plantilla.

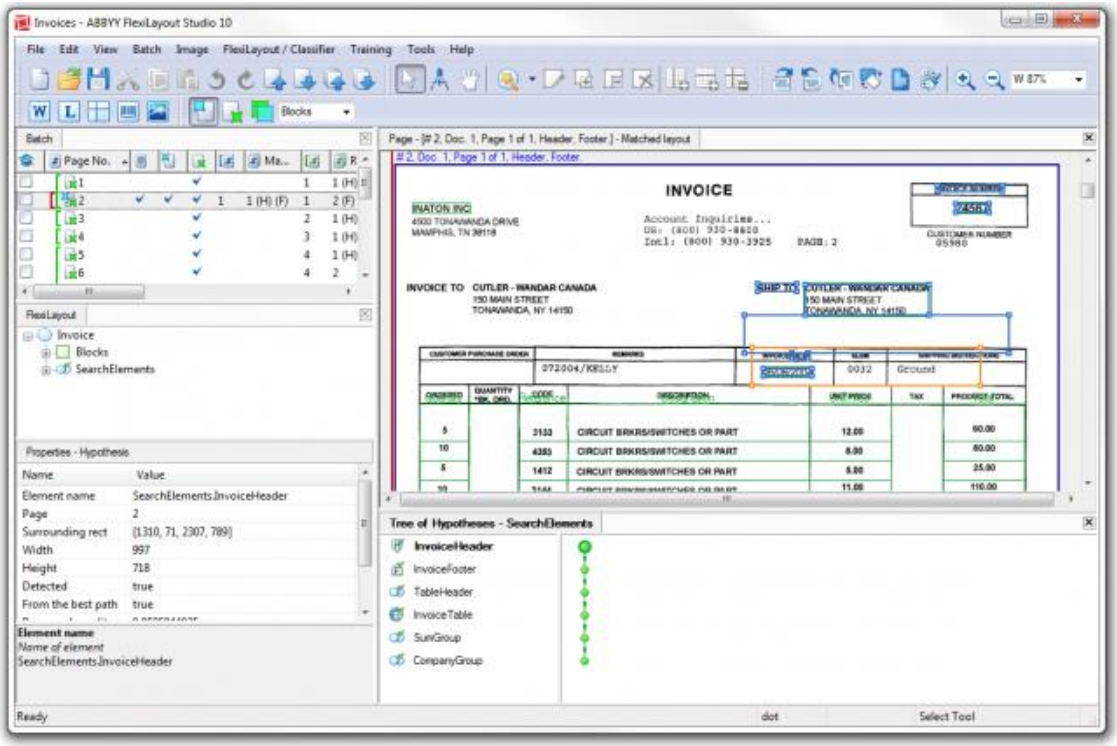


Figura 11 Creando una plantilla flexible con Abbyy

Abbyy gestiona muy bien el concepto de "plantilla estática" y "plantilla flexible". Veamos ambos conceptos, ya que resultará muy útil para la comprensión de este documento:

Concepto de **plantilla**: Una plantilla es una serie de instrucciones e indicaciones para transmitir a un motor OCR que, donde y como tiene que encontrar la información deseada. Por ejemplo, si en un documento queremos encontrar un número de factura, una "pseudoplantilla" sería la siguiente:

header: "Nº Factura"
value: string max 10 char
valuePosition: right of 10px

Es decir, indicar al motor OCR que queremos encontrar un texto que se parezca bastante a "Nº Factura" y que a su derecha debe encontrar una cadena de 10 caracteres como máximo.

Concepto de **plantilla estática**: Una plantilla estática es una plantilla que utiliza regiones estáticas para encontrar información en un documento. Supongamos el típico formulario digital o un test de universidad, donde todos los campos se encuentran siempre en el mismo sitio, no hay posibilidad que aparezcan en otro sitio. Muy importante: ni siquiera cabe la posibilidad de que el movimiento del papel en el espejo o el alimentador de un escáner pueda alterar la posición del campo.

Veamos un documento de ejemplo en la figura 12:

Figura 12. Hoja de marcas para test

Una plantilla estática, para obtener el "Nom i cognoms" buscaría una cadena de forma estática en la siguiente región mostrada en la figura 13:

Figura 13 Hoja de marcas para test con región

El cuadrado verde constituye la región:

- Comienzo eje X: *230px*
- Comienzo eje Y:*70px*
- Alto: *83px*
- Ancho: *1400px*

Entonces, dado que dicho formulario de test no va a cambiar de posición y, al ser digital, no se verá alterado en coordenadas, siempre se obtendrá el valor en dicha región. Veamos, entonces, el concepto de flexibilidad en una plantilla:

Concepto de **plantilla flexible**: **Nuestro proyecto se centra en este concepto**. Una plantilla flexible es una plantilla donde se esperan encontrar campos clave en una zona del documento amplia y, a partir de estos campos clave, determinar el valor deseado de forma flexible, es decir, que con independencia de donde se encuentre en el documento el campo deseado seremos capaces de obtener su valor. Esto es extremadamente útil en el caso de documentos en papel escaneados en un alimentador, donde el valor de un campo puede moverse.

Veamos un ejemplo. Tenemos dos facturas digitalizadas, una mostrada en la figura 14 y la segunda en la figura 15:

Figura 14 Ejemplo plantilla flexible 1

DATOS DE FACTURACION	
Fecha Factura:	01/03/2012
Número de Factura:	2012000000003
Factura Desde:	1/2/2012
Factura Hasta:	29/2/2012
Días Facturados:	29

DATOS DEL CLIENTE	
Nombre:	CLIENTE HOLALUZ
NIF/CIF:	12345678A

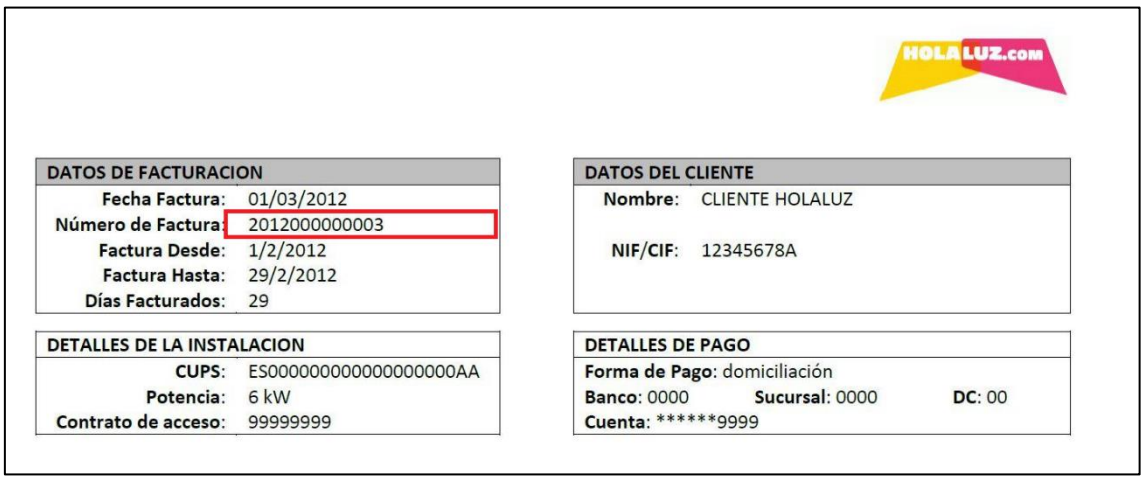
DETALLES DE LA INSTALACION	
CUPS:	ES0000000000000000000AA
Potencia:	6 kW
Contrato de acceso:	99999999

DETALLES DE PAGO		
Forma de Pago: domiciliación		
Banco: 0000	Sucursal: 0000	DC: 00
Cuenta: ******9999		

Figura 15 Ejemplo plantilla flexible 2

Como observamos, por el motivo que sea, la factura 2 se ha escaneado movida, es decir, esta desplazada arriba a la derecha.

Nótese que no aparece ni el logotipo de "holaluz.com". Si aplicáramos el concepto anterior de plantilla estática e intentáramos obtener el valor de "Numero de factura" mediante una región estática veríamos en el caso de la factura 2 no lo obtendríamos. En la figura 16 vemos la región en rojo donde se localiza el número de la factura. En la figura 17 vemos como esa región no sería válida para un documento "movido":



DATOS DE FACTURACION	
Fecha Factura:	01/03/2012
Número de Factura:	2012000000003
Factura Desde:	1/2/2012
Factura Hasta:	29/2/2012
Días Facturados:	29

DATOS DEL CLIENTE	
Nombre:	CLIENTE HOLALUZ
NIF/CIF:	12345678A

DETALLES DE LA INSTALACION	
CUPS:	ES0000000000000000000AA
Potencia:	6 kW
Contrato de acceso:	99999999

DETALLES DE PAGO		
Forma de Pago: domiciliación		
Banco: 0000	Sucursal: 0000	DC: 00
Cuenta: ******9999		

Figura 16 Ejemplo plantilla flexible 1 región seleccionada

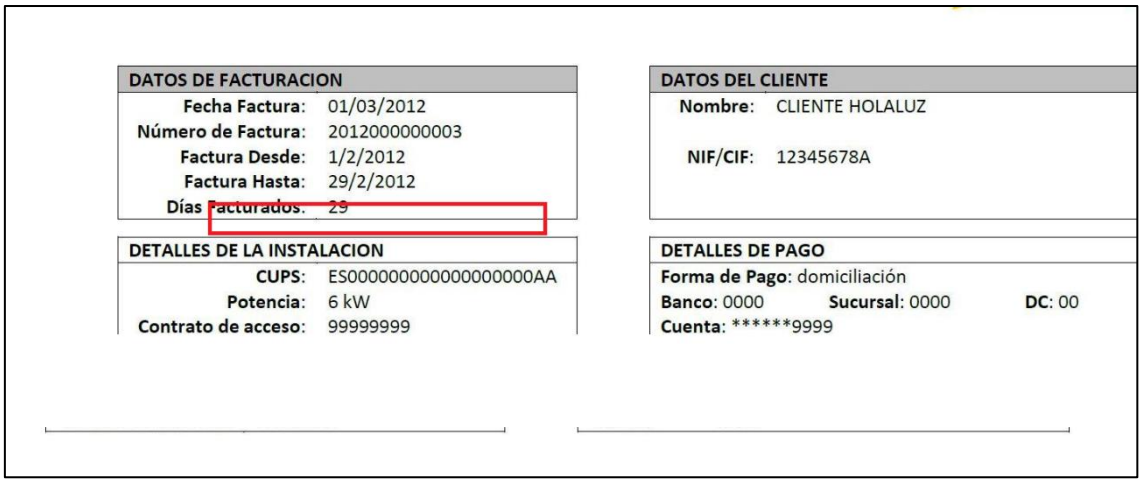


Figura 17 Ejemplo plantilla flexible 2 región seleccionada

Por lo tanto, para obtener el valor usando el concepto de plantilla flexible deberíamos:

1- Localizar el campo identificativo de la plantilla en la zona superior derecha. "CLIENTE HOLALUZ"
2- En referencia a este campo y a su localización en el documento, buscar la cabecera "Número de factura"
3- En base a las coordenadas de esta cabecera, buscar un valor a su derecha de x tamaño.
4- Este valor contendrá "2012000000000003".

Dado que trabajaremos con esta aproximación, se detallará más adelante este punto con ejemplos de código.

2.4 Tesseract OCR

Tesseract es un motor OCR open source. Fue desarrollado originalmente por Hewlett Packard como software propietario entre 1985 y 1995. Tras diez años sin ningún desarrollo, fue liberado como código abierto en el año 2005 por Hewlett Packard y la Universidad de Nevada, Las Vegas. Tesseract es desarrollado actualmente por Google y distribuido bajo la licencia Apache, versión 2.0.

Ofrece soporte unicode (UTF-8) y puede reconocer más de 100 lenguajes "out of the box" mediante entrenamiento (*Tesseract Training*) [11]. Además está considerado como uno de los motores OCR open source con mayor precisión.

Entre sus características funcionales para conseguir dicha precisión destacamos las siguientes: [7].

- Búsqueda de líneas: Los algoritmos de búsqueda de líneas están diseñados para que una imagen con desvió pueda tratarse sin preprocesamientos de ajuste, ahorrando así la pérdida de calidad.
- Ajuste de las líneas bases: Una vez se han localizado las líneas en la imagen Tesseract es capaz de afinar y detectar precisamente las líneas base.
- Detección y corte: Tesseract usa las líneas de texto para determinar, en base al trazo, cuándo debe cortar y separar caracteres en palabras. Además, es capaz de asociar cadenas rotas, como por ejemplo una "N" a la que le falte un segmento lateral.
- Búsqueda proporcional de palabras: Tesseract es capaz de gestionar el espacio entre palabras e incluso el tamaño de estas de forma que no se pierda la relación existente entre ellas.
- Análisis lingüístico: Tesseract cuenta con un set de análisis lingüístico que permite en base a un diccionario de información escoger la cadena adecuada en la detección de un segmento textual dado.

Teniendo en cuenta las características mencionadas, vamos a implementar un lenguaje específico de dominio que nos permita clasificar, identificar y agrupar en PDF diversos documentos mediante el uso de plantillas flexibles con Tesseract OCR.

Abbyy ofrece muchas herramientas para llevar a cabo este trabajo, y posee opciones bastante verticales que permiten hilar en detalle, pero dado su carácter comercial, su coste de económico y su curva alta de aprendizaje, hemos decidido dotar a Tesseract de cierta versatilidad para afrontar el reconocimiento sistemático de documentos.

Actualmente no existe ningún proyecto que permita trabajar con documentos estructurados de forma personalizada y 100% flexible con Tesseract.

En este proyecto estableceremos una base para ir creciendo en funcionalidades y poder disponer de una herramienta basada en Tesseract que nos provea ciertas funcionalidades más comerciales como el tratamiento de plantillas flexibles.

De esta forma, tendremos un producto abierto, personalizable y escalable, basado en un buen motor OCR, gratuito, que permita flexibilizar y hacer llegar a más empresas la fase de identificación y almacenamiento automático de documentos.

Nuestra solución nos permitirá integrarnos fácilmente con otras soluciones del ámbito de la gestión documental web, pieza básica para los vagones del tren de la gestión documental, ese tren que empieza digitalizando documentos y termina accediendo a ellos en segundos, desde cualquier lugar, buscando por cualquier campo.

3. Lenguaje específico de dominio

A diferencia de un lenguaje de uso general, un **lenguaje específico de dominio** (DSL) [12] está diseñado para expresar las instrucciones en un determinado problema de espacio o dominio. La mayor parte de los lenguajes de programación no se pueden considerar DSL ya que no están diseñados para resolver un conjunto específico de problemas, sino para resolver cualquier tipo de problema. Son pues, lenguajes genéricos

Nuestro objetivo fundamental con la aplicación de un DSL es mejorar la calidad, productividad, mantenibilidad y reusabilidad de nuestro proyecto. Un DSL nos va aportar la capacidad de abstraernos de un lenguaje genérico y tener un lenguaje en el cual todas las personas implicadas en el proceso de gestión documental puedan comprender el dominio y saber detalladamente que se está realizando en el proceso.

Nuestro dominio se va a centrar en aspectos detallados como:

- Creación de plantillas flexibles.
- Identificación de las plantillas.
- Identificación de las cabeceras.
- Identificación del campo clave del documento.
- Rutas para la gestión de los documentos.
- Especificación de códigos de barras.
- Coordenadas de posicionamiento relativo.

Los DSL son comúnmente clasificados como internos o externos:

- Los DSL internos son escritos en un lenguaje de programación de propósito general, cuya sintaxis se ha inclinado a parecerse más al lenguaje natural.
- Los DSL externos son expresiones gráficas o textuales de un lenguaje, aunque los DSL textuales tienden a ser más comunes que los gráficos. Las expresiones textuales pueden ser

> procesadas por una cadena de herramientas que incluyen léxico, un analizador, un transformador de modelo, generadores, y cualquier otro tipo de post-procesamiento.

Necesitamos, entonces, hacer uso de un lenguaje específico de dominio externo. Desarrollaremos un DSL y proporcionaremos un IDE que soporte la creación, modificación y gestión de nuestro lenguaje. Este IDE nos permitirá también crear un ejecutable que ponga en marcha el proceso especificado.

Existen ciertos inconvenientes en la implantación de nuestro DSL tales como:

- Periodo de aprendizaje de las partes interesadas en el DSL.
- Tiempo invertido en la especificación e implementación.
- Necesidad de seguir avanzando y desarrollando el DSL para ir cubriendo más aspectos (ver *Futuras líneas de investigación*).

Para minimizarlos lo máximo posible realizaremos lo siguiente:

- Usar un DSL sencillo y autodocumentado que a la vista resulte comprensible.
- Asumir el tiempo invertido como una inversión retornable dada la potencial mejora que vamos a obtener en el proceso de identificación y clasificación de documentos.
- Especificar claramente cuáles son los siguientes objetivos que cubrir para que cada desarrollo incremental suponga un beneficio claro y preciso.

Para el desarrollo de nuestro DSL vamos a elegir el meta lenguaje XML [13]. La elección de este metalenguaje se debe a que es ampliamente utilizado, nos da bastante flexibilidad para la estructuración, hay muchas librerías para manejarlo y a nivel visual es bastante comprensible entender su estructura.

En esta sección vamos a especificar nuestro DSL, a detallar el significado de cada nodo y las restricciones asociadas.

3.1 ¿Qué vamos a representar en nuestro DSL?

Como hemos mencionado anteriormente, nuestro DSL tiene que ser capaz de permitir:

- Creación de plantillas flexibles.
- Identificación de las plantillas.
- Identificación de las cabeceras.
- Identificación del campo clave del documento.
- Rutas para la gestión de los documentos.
- Especificación de códigos de barras.
- Coordenadas de posicionamiento relativo.

Bien, a nivel genérico tenemos que modelar lo siguiente:

- Una **ruta de entrada** que contenga las imágenes a procesar.
- Una **ruta de fallidos** para los documentos en los que el proceso haya fallado.
- Representar el **concepto de plantilla**. Una plantilla es una forma de indicar al motor OCR que, donde y como queremos reconocer los campos.
- **Nombre de la plantilla**. Si la plantilla está vinculada con facturas de telefónica, nombramos la plantilla como “Facturas Telefónica”.
- **Ruta de salida**. Necesitamos indicar una ruta de salida para dejar los PDF’s agrupados e identificados.
- **Tipo de reconocimiento**. Como indicamos en apartados anteriores, se va a reconocer una plantilla en base a un **campo clave** o a un **código de barras**.
- En caso de reconocer por código de barras, será necesario establecer una **expresión regular** para validar que código de barras es válido para la plantilla (ya que puede ser que en una

misma hoja haya varios) y especificar el **tipo de código de barras** (*QR, CODE_39, EAN, etc..).*

- En caso de reconocer por campo clave, necesitaremos especificar que **campos nos indican el tipo de factura**, que campo "**cabecera**" vamos a utilizar para localizar el campo identificativo y el propio **campo identificativo**.

Veamos el siguiente diagrama (*figura 18*) genérico de nuestro DSL donde, a lo largo de esta sección, iremos expandiendo hasta el definitivo:

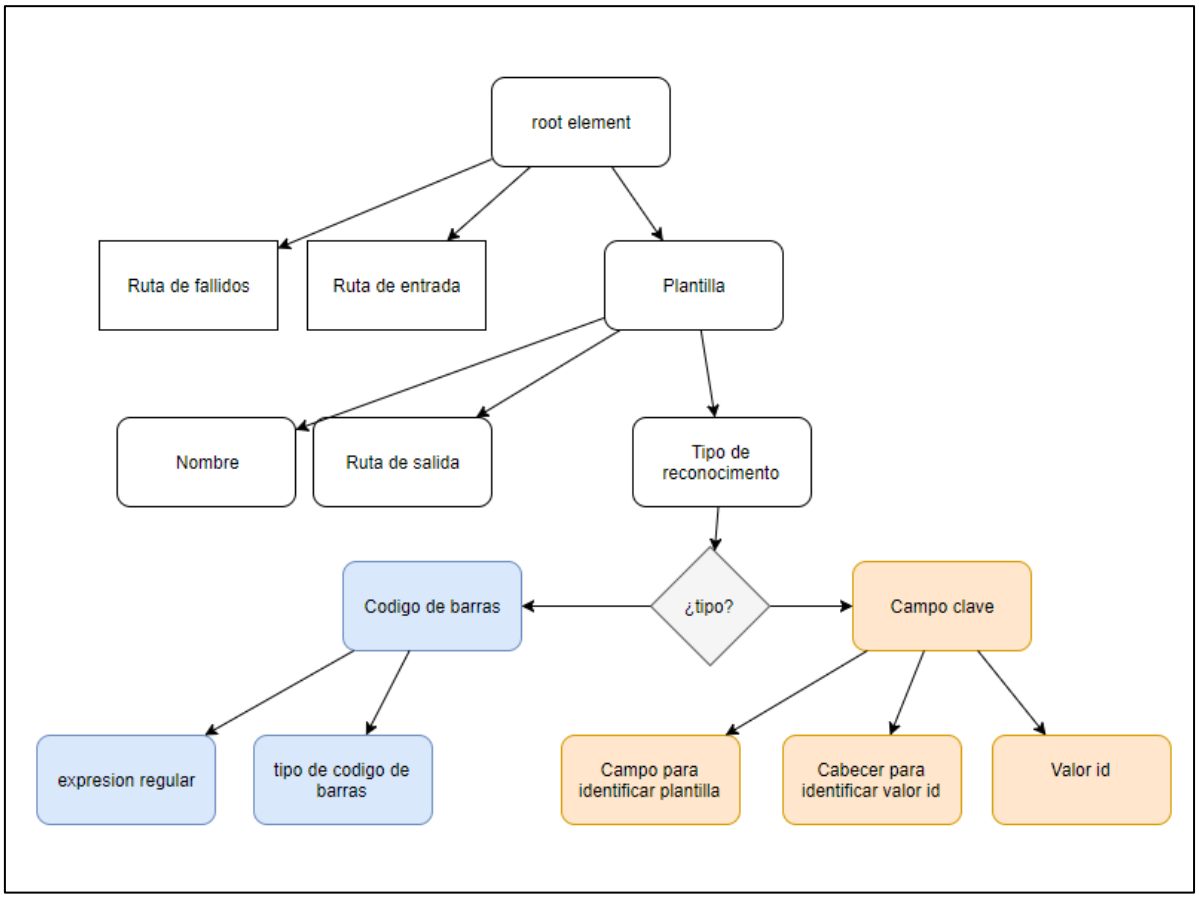


Figura 18 Diagrama genérico del DSL

Como podemos ver, parece que los nodos en color blanco ya están definidos y se produce una bifurcación a la hora de escoger el tipo de reconocimiento. Planteemos pues nuestro XML si no tenemos en cuenta el tipo.

Necesitamos un XML en inglés, con un nombre que sea claro y que permita a todas las partes interesadas comprenderlo fácilmente. Quedaría de esta forma:

```
<root>
    <entryPath>C:\Users\Javier Escribano\Desktop\entry</entryPath>
    <failedPath>C:\Users\Javier Escribano\Desktop\failed</failedPath>
    <template>
        <name>Template 1</name>
        <outputPath>
          C:\Users\Javier Escribano\Desktop\output\template1
        </outputPath>
        <recognizementType>
            <!-- TO BE DEFINED -->
        </recognizementType>
    </template>
</root>
```

Hasta el momento, vemos reflejado los nodos blancos de la figura 18 en el XML. Ya podemos ir observando las primeras restricciones en la tabla 2:

Campo	Restricción
Root	Elemento raíz obligatorio
entryPath	Ruta de entrada. Debe ser una carpeta. Obligatorio
failedPath	Ruta de fallidos. Debe ser una carpeta. Oblitatorio.
template	Nodo que contiene la información de la plantilla. Puede ser múltiple. Obligatorio.
Name	Nombre de la plantilla. Obligatorio.
outputPath	Ruta de salida para el PDF generado. Debe ser una carpeta. Obligatorio.
recognizementType	Tipo de reconocimiento. Obligatorio. Sus valores deben ser "barcodeFirstPage" o "keyField", correspondientes a identificación por "Codigo de Barras" o "Campo clave".

Tabla 2 Restricciones genéricas del DSL

3.2 Reconocimiento por código de barras

Siguiendo con el esquema de la sección anterior, ahora vamos a ampliar el XML para identificar documentos con código de barras. Como hemos mencionado, vamos a necesitar una expresión regular para filtrar códigos de barras y un campo que nos indique el tipo de código de barras.

La expresión regular para el valor del código de barras nos permitirá, en caso de que haya varios códigos de barras, detectar cual es el de nuestro interés. El tipo de código de barras nos indicará, en adición, que tipo es. El XML ampliado, por tanto, quedaría así:

```
<root>
    <entryPath>C:\Users\Javier Escribano\Desktop\entry</entryPath>
    <failedPath>C:\Users\Javier Escribano\Desktop\failed</failedPath>
    <template>
        <name>Template 1</name>
        <outputPath>
          C:\Users\Javier Escribano\Desktop\output\template1
        </outputPath>
        <recognizementType>
            barcodeFirstPage
        </recognizementType>
        <regExp>TPL\d*$</regExp>
        <barcodeType>CODE 39</barcodeType>
    </template>
</root>
```

Como se puede observar, se han incluido ambos campos, regExp para la expresión regular (*en este caso se validará que el código de barras comience por TPL después solamente vengan dígitos*) y barcodeType para el tipo del código de barras. Esto queda resumido en la tabla 3.

Campo	Restricción
regExp	Expresión regular de filtrado. Obligatorio si <recognizementType> es barcodeFirstPage. Aconsejable que sea una expresión regular bien formada.
barcodeType	Tipo del código de barras. Obligatorio si <recognizementType> es barcodeFirstPage. Debe ser "CODE 39", "QR", "EAN 13" o "EAN 8"

Tabla 3 Restricciones genéricas del DSL – Código de barras

3.3 Reconocimiento por campo clave

Este tipo de reconocimiento es el que más lógica lleva involucrada. Como comentamos anteriormente:

"*…necesitaremos especificar que **campos nos indican el tipo de factura**, que campo "**cabecera**" vamos a utilizar para localizar el campo identificativo y el propio **campo identificativo**. "*

Comencemos por la identificación del **tipo de factura**. Como se indicó en la sección que explica el funcionamiento de las plantillas flexibles, necesitaremos especificar una región donde localizar una o varias cadenas que me permitan identificar que tipo de documentos es, por ejemplo, si pretendo identificar una factura de telefónica, necesitaré encontrar las palabras "Factura" y "Telefónica" en una región concreta del documento.

Aparecen por tanto el concepto de región y valor a encontrar. La región se indica mediante el siguiente patrón en pixeles (*recordemos que la digitalización de las facturas será homogénea, no puede haber, por ejemplo, una factura a 200dpi y otra a 500dpi*)

Startx, starty, endx, endy

Donde x e y son los ejes de coordenadas expresados en pixeles. Esto conforma un rectángulo que especificará donde encontrar un valor. También necesitaremos establecer un id por cada valor especificado de forma que posteriormente podamos decir "a partir de este valor, quiero obtener la cabecera x ". Como comentamos en las plantillas flexibles, los valores se obtienen en base al campo que se ha utilizado para identificar a la plantilla, por lo que identificando estos campos posteriormente podremos referenciarlos.

Trasladándolo a XML tenemos:

```
<root>
    <entryPath>C:\Users\Javier Escribano\Desktop\entry</entryPath>
    <failedPath>C:\Users\Javier Escribano\Desktop\failed</failedPath>
    <template>
        <name>Template 1</name>
        <outputPath>C:\Users\Javier Escribano\Desktop\output\template1</outputPath>
        <recognizementType>
          keyField
        </recognizementType>
        <templateIdentification>
          <key id="1" region="8,522,26,105">Transportes</key>
          <key id="2" region="8,522,26,105">Fernandez</key>
          <key id="3" region="14,260,90,168">Factura</key>
        </templateIdentification>
    </template>
</root>
```

Hemos añadido, por tanto, el nodo <templateIdentification> que contiene un nodo <key> por cada valor que tratamos de encontrar. Como atributos de clave tenemos el identificador y la región ambos explicado en el párrafo anterior.

Visualmente observamos que pretendemos identificar el documento buscando en las regiones dadas las palabras Transportes, Fernández y Factura. Si el motor Tesseract OCR las encuentra, habremos identificado la plantilla. Veamos las restricciones comentadas en la tabla 4:

Campo	Restricción
templateIdentification	Contiene los elementos para identificar una plantilla. Obligatoria si recognizementType es keyField
key	Clave a encontrar. Al menos una key es obligatoria.
Key[“id”]	Identificador de la clave. Debe ser un numero positivo. Posteriormente se utilizará para ser relacionado.
Key[“region”]	Región donde buscaremos la palabra. Sigue el patrón en pixeles <startx, starty, endx, endy>

Tabla 4 Restricciones genéricas del DSL – Identificación de plantilla

Continuemos ahora con la identificación de la cabecera. Por ejemplo, si queremos encontrar el Numero de Factura, antes debemos posicionarlos en la propia cadena “Numero de Factura”. Esta cadena

se tiene que encontrar en el documento en base a un campo clave. De ahí que lo hayamos identificado con el campo <id>.

Posteriormente se verá más en detalle (*sección Implementación*), pero hay que destacar que cuando el motor OCR encuentre el campo clave para identificar la plantilla, por ejemplo, "Telefónica", nos devolverá el cuadrado exacto que rodea a la cadena telefónica, indicándonos el patrón <startx, starty, endx, endy> en pixeles de su localización en la imagen:

Entonces, para saber posicionar la localización del texto "Numero de factura" vamos a establecer una serie de relaciones respecto al rectángulo que forma el campo utilizado para identificar la plantilla (ejemplo superior de Telefónica).

Las posiciones posibles respecto este campo son "arriba de", "debajo de", "a la derecha de" y "a la izquierda de".

Arriba, abajo, derecha e izquierda de puede realizarse respecto la parte superior del cuadrado, la parte inferior y la izquierda del cuadrado o la derecha de él respectivamente.

Por lo tanto, si combinamos estas posiciones obtenemos:

- Arriba de la parte superior del rectángulo.
- Arriba de la parte inferior del rectángulo.
- Debajo de la parte superior del rectángulo.
- Debajo de la parte inferior del rectángulo.
- A la derecha de la derecha del rectángulo.
- A la derecha de la izquierda del rectángulo.
- A la izquierda de la izquierda del rectángulo.
- A la izquierda de la derecha del rectángulo.

Cada una de estas posiciones se debe indicar, como no, en pixeles. Los pixeles se deben indicar con un número entero. Se aceptan valores negativos, ¿Por qué? Veamos un ejemplo y así comprendamos mejor el concepto de posicionamiento:

Supongamos que, a partir del valor Telefónica, queremos encontrar la cabecera "Numero de Factura" y ya tenemos las coordenadas de la palabra "Telefónica" en el documento (*ver figura 19*):

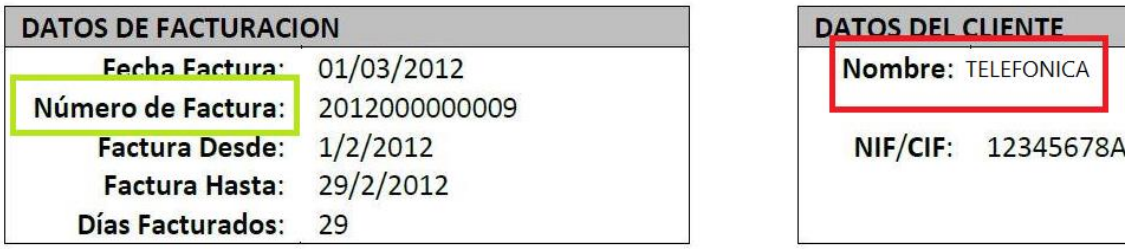


Figura 19 Localizar cabecera 1

Para que sea más claro, vamos a borrar de la imagen el resto de elementos y a dejar los que nos interesan para poder marcar el posicionamiento (*ver figura 20*):

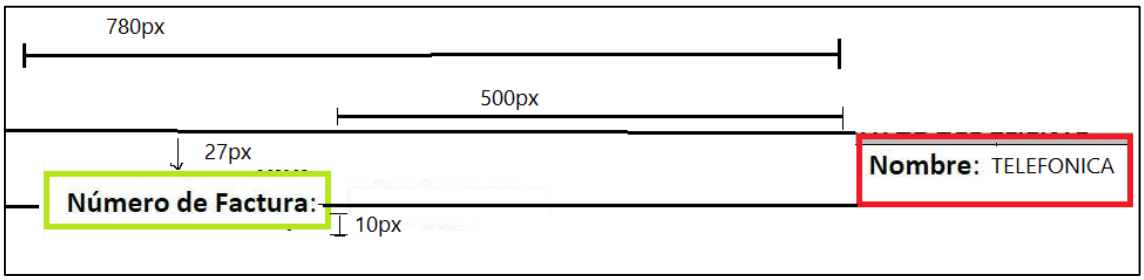


Figura 20 Localizar cabecera 2

Vamos a traducir lo que se visualiza en la figura 20 en posiciones:

- El rectángulo "número de factura" se sitúa mínimo 500 pixeles a la izquierda del rectángulo de "telefónica".
- El rectángulo "numero de factura" se sitúa a la derecha de la izquierda del rectángulo telefónica menos 780px. (se resta a la

posición de la izquierda del rectángulo de telefónica 780 pixeles, de ahí el valor negativo.

- La parte superior del rectángulo de "Numero de factura" se encuentra por debajo unos 20pixeles (*27 max*) de la parte superior del rectángulo de telefónica.
- La parte inferior del rectángulo "Numero de factura" se encuentra por encima de la parte inferior del rectángulo telefónica más 10px.

Para entender estas relaciones, incluimos un dibujo con las líneas ficticias que pretenden representar el posicionamiento (*ver figura 21*):

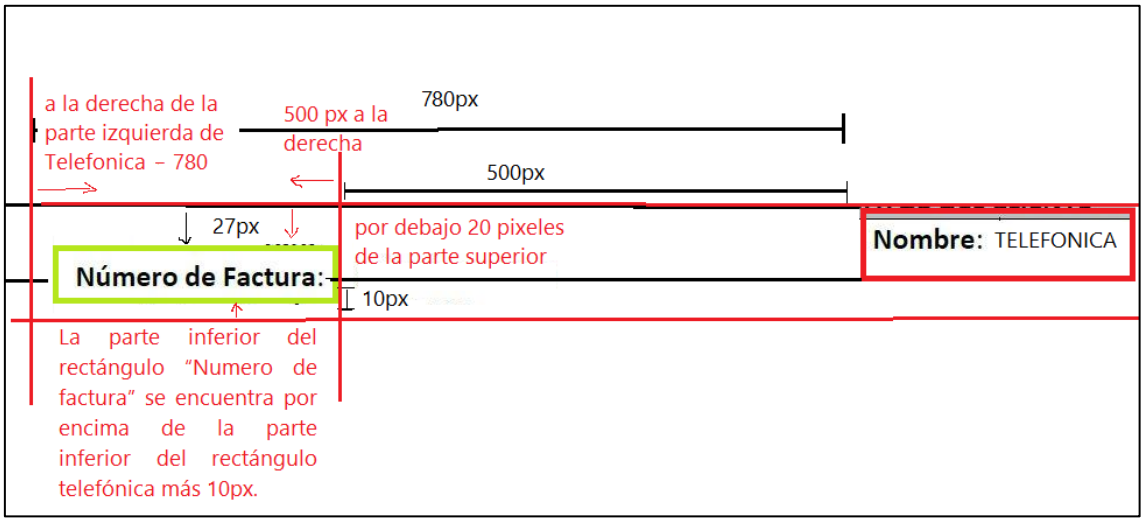


Figura 21 Localizar cabecera 3

La idea es posicionarnos por el eje de coordenadas usando valores positivos o negativos y valores relativos de abajo, arriba, izquierda derecha. Vamos a detallar el XML para encontrar la cabecera utilizando estos conceptos:

```
<root>
    <entryPath>C:\Users\Javier Escribano\Desktop\entry</entryPath>
    <failedPath>C:\Users\Javier Escribano\Desktop\failed</failedPath>
    <template>
        <name>Template 1</name>
        <outputPath>C:\Users\Javier Escribano\Desktop\output\template1</outputPath>
        <recognizementType>
          keyField
        </recognizementType>
        <templateIdentification>
          <key id="1" region="8,522,26,105">Telefonica</key>
          <key id="2" region="9,222,2,505">Factura</key>
        </templateIdentification>
        <fieldHeader>
          <value>Numero de factura</value>
          <fieldRef>1</fieldRef>
          <position>
            <left>
              <of>left</of>
              <pix>500</pix>
            </left>
            <below>
              <of>top</of>
              <pix>20</pix>
            </below>
            <above>
              <of>bottom</of>
              <pix>-10</pix>
            </above>
            <right>
              <of>left</of>
              <pix>-780</pix>
            </right>
          </position>
        </fieldHeader>
    </template>
</root>
```

Como observamos la cabecera la nombramos como <fieldHeader>. Dentro del nodo <fieldHeader> debemos incluir el valor de la cabecera a buscar (*recordamos, "Numero de factura"*), el id de la clave con el que nos intentamos posicionar (*recordamos, "telefónica", id = 1*) y la posición.

Como hemos visto para la posición necesitamos "donde nos ubicamos" respecto a "que" y "cuantos pixeles". Esto lo modelamos de la siguiente forma:

Above/Below of top/bottom X px

Right/Left of right/left X px

En la tabla 5 se especifican las restricciones para este nodo <fieldHeader>:

Campo	Restricción
fieldHeader	Nodo para localizar la cabecera. Obligatorio si recognizementType es "keyField"
value	Valor de la cabecera a encontrar. Obligatorio.
fieldRef	Obligatorio. Debe contener una referencia al id de una de las <keys>.
position	Nodo que contiene la información del posicionamiento. Obligatorio.

Tabla 5 Restricciones genéricas del DSL – Identificación de cabecera

Las restricciones para el nodo <position> las detallamos a continuación:

- El nodo position debe tener al menos un hijo.
- Los nodos hijos solamente pueden tener los valores <above><below><right><left>
- Cada uno de los nodos hijos, a su vez, deben tener obligatoriamente dos hijos <of><px>. El nodo <px> debe ser un valor entero. Los valores para el nodo <of> los detallamos a continuación:
    - Si padre == 'above' or padre == 'below', los valores de <of> deben ser <top> o <bottom>
    - Si padre == 'right' or padre 'left', los valores de <of> deben ser <right> o <left>

Para clarificar la estructura del DSL, vamos a ampliar el diagrama de la figura 18 en la figura 22:

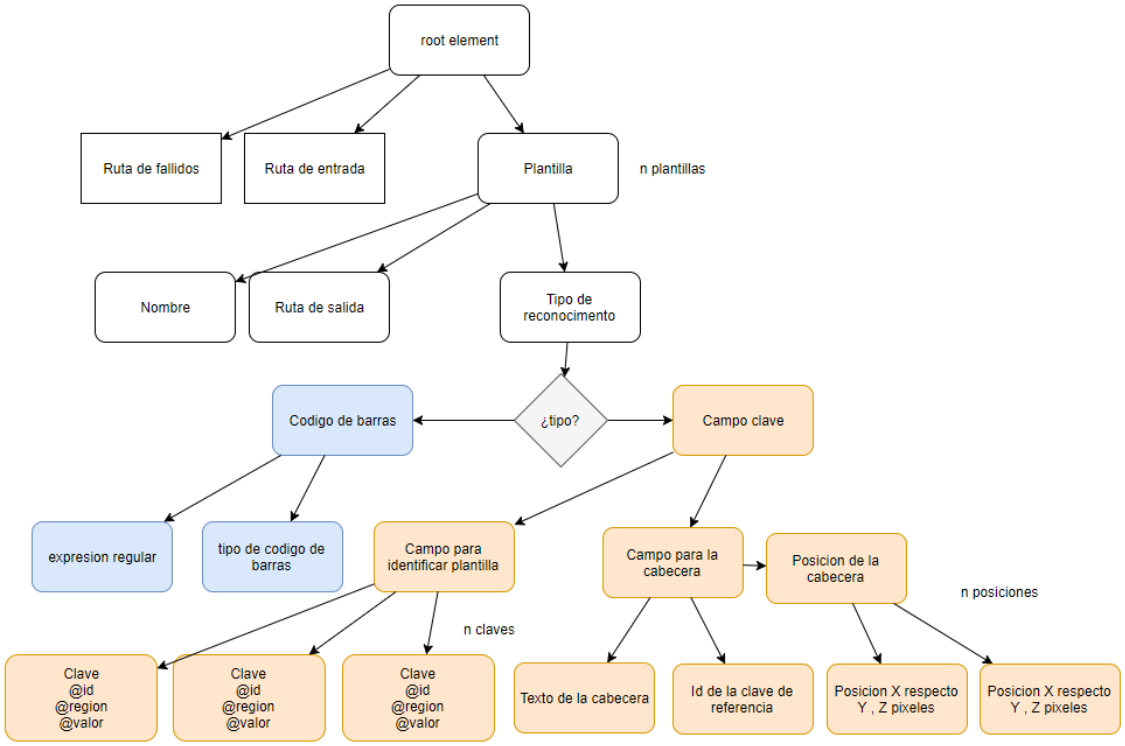


Figura 22 Diagrama genérico del DSL 2

Por último, una vez tenemos identificada la plantilla y la cabecera, tan solo nos queda obtener el valor objetivo, el valor que identificará al documento. Si en el punto anterior queríamos localizar el texto "Numero de factura", ahora debemos localizar el valor del numero de la factura, que es el objetivo final para el motor de OCR.

La posición del valor id la obtendremos a partir de la cabecera. Como ya la tenemos controlada en el campo <fieldHeader>, tan solo resta incluir otro nodo llamado <fieldValue> con las posiciones, tal y como hicimos con el nodo <position> de <fieldHeader>.

La lógica y restricciones para el nodo <fieldValue> son exactamente las mismas que para el nodo <position> dentro de <fieldHeader>. La única diferencia se haya en el comportamiento. <position> se localiza a partir del <key> (*recordamos, buscábamos "Numero de factura" desde el rectángulo de "Telefonica"*) y <fieldValue> a partir de <fieldHeader> (*ahora buscaremos un valor X desde el nodo "Numero de factura"*).

Por lo tanto, ahora el esquema definitivo de nuestro DSL (figura 23) queda así:

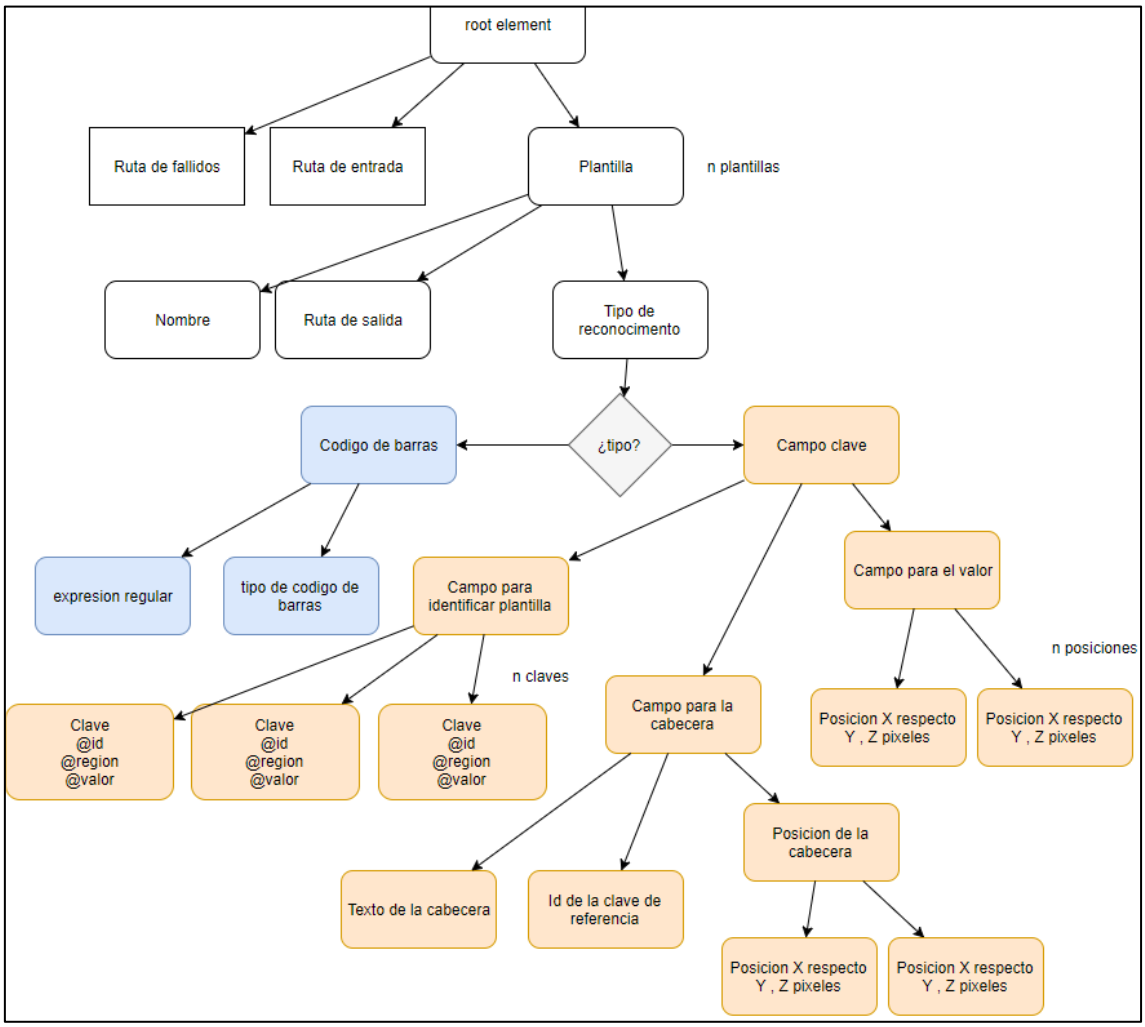


Figura 23 Diagrama definitivo del DSL

Dado que ya tenemos el diagrama definitivo de nuestro DSL, vamos a exponer un XML de ejemplo válido y explicaremos brevemente el comportamiento esperado:

```xml
<root>
    <entryPath>C:\Users\Javier Escribano\Desktop\entry</entryPath>
    <failedPath>C:\Users\Javier Escribano\Desktop\failed</failedPath>
    <template>
        <name>ProformaBarcode</name>
        <outputPath>C:\Users\Javier Escribano\Desktop\output\proformaBarcode</outputPath>
        <recognizementType>barcodeFirstPage</recognizementType>
        <regExp>LUZ\d*$</regExp>
        <barcodeType>CODE 39</barcodeType>
    </template>
    <template>
        <name>HolaLuz</name>
        <outputPath>C:\Users\Javier Escribano\Desktop\output\holaluz</outputPath>
        <templateIdentification>
            <key id="1" region="660,1400,40,300">Holaluz</key>
        </templateIdentification>
        <recognizementType>keyField</recognizementType>
        <fieldHeader>
            <value>Factura</value>
            <fieldRef>1</fieldRef>
            <position>
                <below>
                    <of>bottom</of>
                    <pix>3</pix>
                </below>
                <above>
                    <of>bottom</of>
                    <pix>-40</pix>
                </above>
                <left>
                    <of>left</of>
                    <pix>200</pix>
                </left>
                <right>
                    <of>left</of>
                    <pix>-830</pix>
                </right>
            </position>
        </fieldHeader>
        <fieldValue>
            <right>
                <of>right</of>
                <pix>10</pix>
            </right>
            <left>
                <of>right</of>
                <pix>-200</pix>
            </left>
            <above>
                <of>bottom</of>
                <pix>-4</pix>
            </above>
            <below>
                <of>top</of>
                <pix>-4</pix>
            </below>
        </fieldValue>
    </template>
<root>
```

Este XML contiene un path de entrada <entryPath> que apunta a ..*Desktop**Entry*. En esta carpeta deben estar las imágenes a procesar. En caso de que se produzca algún error o no se reconozca la plantilla asociada, los archivos se moverán a la carpeta <failedPath>.

Tenemos dos plantillas, "ProformaBarcode" y "HolaLuz". A simple vista podemos ver que la primera es más sencilla, ya que únicamente espera encontrar un código de barras del tipo Code 39 con un valor que siga la expresión "LUZ + dígitos numéricos".

Por otro lado, la segunda plantilla "HolaLuz" se identifica por campo clave. Para determinar que plantillas son del tipo HolaLuz buscaremos en la región < "660,1400,40,300"> el texto "HolaLuz".

Vemos por el nodo <fieldHeader> que queremos buscar el texto "Factura" en una posición determinada a partir del campo clave *<key id="1" region="660,1400,40,300">Holaluz</key>*. Con el valor de la cabecera podremos obtener el valor del número de factura <fieldValue>.

3.4 Soporte al DSL

Hasta aquí ya tenemos una visión de lo que pretendemos conseguir. Tenemos los conceptos teóricos claros y un DSL de nuestro dominio especificado y ejemplificado.

Para poder crear, modificar y mantener nuestro DSL hemos desarrollado un sencillo IDE que permita:

- Crear DSL's mediante el uso del XML.
- Modificar DSL existentes.
- Cargar y/o guardar en el equipo nuestro DSL.
- Generar un ejecutable a partir de nuestro DSL.
- Validar el XML para que se nos notifique de los posibles errores sintácticos o semánticos.

El IDE se ha desarrollado usando un proyecto de Windows Forms [14] en Visual Studio usando el lenguaje C# (*ver Herramientas seleccionadas*).

El IDE dispone de una barra superior de acciones, de un cuerpo para escribir nuestro DSL y una sección inferior de información (*ver figura 24*)

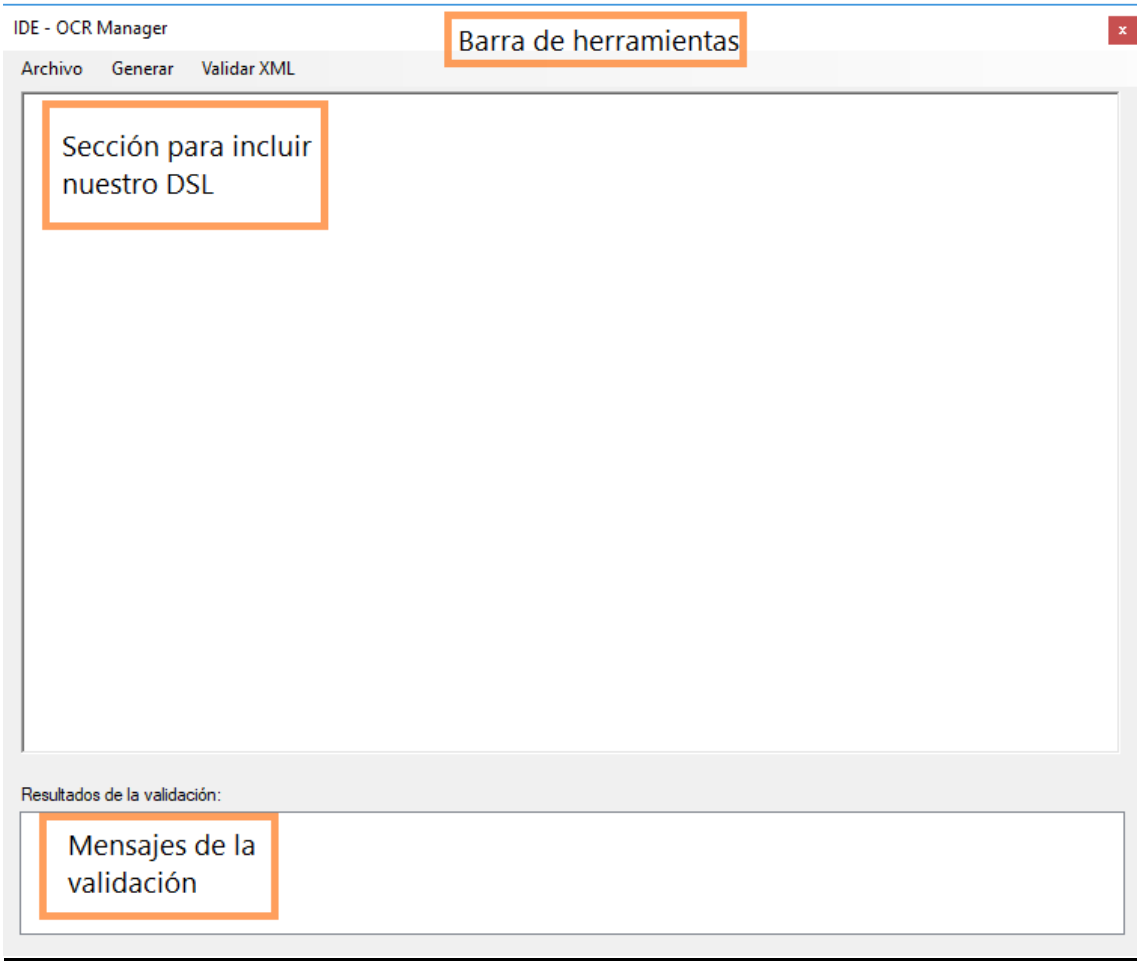


Figura 24 Interfaz principal del IDE

El menú superior contiene las **acciones** básicas del entorno integrado de desarrollo, que son:

- Archivo
    - **Cargar plantilla base**: Carga un DSL preestablecido con los campos necesarios para implementar una plantilla. De esta forma la tarea de crear la plantilla se simplifica y únicamente tendremos que modificar los valores de los nodos requeridos.
        - Barcode: Carga un DSL preestablecido específico para una plantilla donde el campo clave es un código de barras.

        - KeyField: Carga un DSL preestablecido específico para una plantilla donde el campo clave es un campo del propio documento.
    - **Cargar plantilla**: Carga la plantilla que seleccionemos de nuestro equipo.
    - **Guardar plantilla**: Guarda la plantilla en el lugar deseado en nuestro equipo.
    - **Cerrar**: Cierra la aplicación.

- **Generar**: Genera un ejecutable que permita probar la aplicación real de nuestro equipo.
- **Validar**: Valida sintáctica y semánticamente nuestro DSL y nos proporciona el resultado de la validación en la sección inferior habilitada para este propósito.

4. Implementación

Para la realización del proyecto, a parte del motor de OCR Tesseract mencionado anteriormente, hemos escogido una serie de herramientas. En cada apartado siguiente vamos a especificar la herramienta seleccionada y el motivo por el cual se ha escogido:

Visual Studio + C#: Utilizaremos C# [15] como lenguaje de desarrollo y Visual Studio [16] como entorno de programación. La elección de VS + C# se debe a que gran parte de las empresas trabajan con sistemas operativos Windows por lo que es necesario producir aplicaciones compatibles con esta plataforma.

Nuestro ejecutable potencialmente se convertirá en un servicio de Windows o una tarea programada. Además, necesitaremos un compilador para generar un ejecutable a través de nuestro DSL. VS nos proveerá de todo lo necesario.

Se desarrollará bajo el Framework 4.0 de .NET [17].

Tesseract wrapper for .NET [18]: Utilizaremos esta librería bajo licencia apache 2.0. Nos proporcionará los métodos necesarios para procesar un archivo por OCR y obtener los resultados.

PDFSharp.NET [19]: Utilizaremos esta librería Open Source para, dado un conjunto de imágenes, generar un PDF.

ZXing.NET [20]: Utilizaremos esta librería bajo licencia Apache 2.0 para leer los códigos de barras ya que soporta varios tipos (CODE 39, QR, EAN, etc…).

4.1 Implementación del IDE

En esta sección vamos a explicar la implementación del IDE que da soporte al DSL diferenciando en tres secciones, la de cargado/guardado de plantillas, la validación de nuestro DSL y la de generación del ejecutable:

Sección de cargado/guardado de plantillas

Para cargar y guardar plantillas simplemente utilizamos las clases de .NET FileDialog y StreamWriter/Reader para solicitar al usuario el fichero a cargar/guardar y posteriormente cargarlo o guardarlo respectivamente:

Lectura de un fichero .xml que contiene nuestro DSL y carga en el IDE en C#:

```
openFileDialog1.Filter = "XML-File | *.xml";
DialogResult result = openFileDialog1.ShowDialog();
if (result == DialogResult.OK)
{
    StreamReader sr = new StreamReader(openFileDialog1.FileName);
    string fileContent = sr.ReadToEnd();
    sr.Close();
    xmlEditor1.Text = fileContent;
}
```

Escritura de un fichero .xml que contiene nuestro DSL desarrollado en el IDE:

```
saveFileDialog1.Filter = "XML-File | *.xml";

DialogResult result = saveFileDialog1.ShowDialog();
if (result == DialogResult.OK)
{
    StreamWriter sw = new StreamWriter(saveFileDialog1.FileName);
    sw.Write(xmlEditor1.Text);
    sw.Close();
}
```

Para las plantillas base se han creado dos XML de forma que el desarrollador pueda invertir el tiempo en los valores deseados del

propio DSL en vez de la sintaxis de este. Los dos DSL base que se han establecido son los siguientes XML:

Base Barcode DSL

```
<root>
  <entryPath>C:\entry</entryPath>
  <failedPath>C:\failed</failedPath>
  <template>
      <name>TemplateName</name>
      <outputPath>C:\output\templateName</outputPath>
      <recognizementType>barcodeFirstPage</recognizementType>
      <regExp>\d*$</regExp>
      <barcodeType>CODE 39</barcodeType>
    </template>
</root>
```

Base KeyField DSL

```
<root>
  <entryPath>C:\entry</entryPath>
  <failedPath>C:\failed</failedPath>
  <template>
      <name>TemplateName</name>
      <outputPath>C:\output\templateName</outputPath>
      <templateIdentification>
        <key id="1" region="1,1,2,2">key string</key>
      </templateIdentification>
      <recognizementType>keyField</recognizementType>
      <fieldHeader>
        <value>header value string</value>
        <fieldRef>1</fieldRef>
        <position>
          <below>
            <of>bottom</of>
            <pix>10</pix>
          </below>
         <right>
            <of>left</of>
            <pix>-5</pix>
          </right>
        </position>
      </fieldHeader>
      <fieldValue>
        <left>
          <of>right</of>
          <pix>-200</pix>
        </left>
        <above>
          <of>bottom</of>
          <pix>-4</pix>
        </above>
      </fieldValue>
    </template>
</root>
```

Sección de la validación

Durante el desarrollo del DSL es crucial poder realizar una validación. Cada vez que queramos validar nuestro DSL únicamente tendremos que pulsar el botón del menú de herramientas "Validar".

El método de validación básicamente recorrerá el XML para encontrar errores sintácticos (*falta del elemento raíz <root>, nodo mal cerrado o abierto, caracteres ilegales, etc…)* y semánticos (*ver Especificación del DSL*) tales como ausencia de campos obligatorios o valores no permitidos.

Para realizar la validación sintáctica y semántica se recorre el XML y se valida nodo a nodo que respete nuestra especificación. Todo este proceso se envuelve en un try/catch que permita detectar errores sintácticos en el XML:

Lectura del XML:

```
XmlDocument doc = new XmlDocument();
XmlReaderSettings settings = new XmlReaderSettings();
settings.IgnoreComments = true;
XmlReader reader = XmlReader.Create(new StringReader(xmlEditor1.Text), settings);
doc.Load(reader);
```

Para ayudar a la compresión del código C# que se irá mostrando en el documento, explicaremos el uso de los elementos esenciales.

La clase XMLDocument nos permite gestionar un documento XML. El método load() tratará de cargar el contenido que nosotros le indiquemos. En el constructor de la clase XMLDocument se valida internamente que el XML pasado cumpla con la sintaxis. En caso de ser así, se producirá una excepción. Capturamos dicha excepción con un bloque try/catch y mostramos el mensaje en la sección de información habilitada en el IDE:

```
catch (Exception ex)
{
     listBox1.Items.Add("Error en la lectura del XML: " + ex.Message);
}
```

El trabajo nodo a nodo lo realizaremos mediante el uso de la clase XMLNode, que nos permite acceder tanto a los elementos y atributos del nodo XML como hacer consultas a los hijos de este.

Por ejemplo, veamos como validamos el nombre de la plantilla:

```
XmlNodeList templateNodes = doc.DocumentElement.SelectNodes("template");

foreach (XmlNode templateNode in templateNodes)
{
        string name = templateNode.SelectSingleNode("name").InnerText;
        if(name == "")
        {
           listBox1.Items.Add("No ha especificado un nombre para la plantilla");
        }

...
```

La clase XMLNodeList permite trabajar con colecciones de nodos XML, que es precisamente el valor de retorno del método SelectNodes(string).

Dada la sencillez en la implementación de la validación de un XML vamos a indicar un solo ejemplo más acerca de la validación de nuestro DSL. Veamos como validamos las coordenadas de la región de un campo clave:

```
try
{
     string currentRange = currentAttribute.InnerText;
     string[] rangeCoords = currentRange.Split(',');

     if(rangeCoords.Length != 4)
     {
        throw new Exception("Se necesitan 4 coordenadas: <startx, starty,
endx,endy>");
     }

     foreach(string coord in rangeCoords)
     {
         Int32.Parse(coord);
     }
}
catch(Exception ex)
{
     listBox1.Items.Add("El atributo " + currentAttribute.Name + " para la clave
" + keyNode.InnerText + " es incorrecto en la plantilla: " + name + " - " +
ex.Message);
}
```

Primero accedemos al atributo *range* que se encuentra en el XMLNode currentAttribute. Dividimos la cadena por el carácter ',' y

verificamos tanto que existan 4 coordenadas como el tipo de cada uno de los valores, ya que deben de ser números enteros (Int32.Parse(*string*)). En caso de que se produzca cualquier error, lo incluimos en la sección de información del IDE (*listbox1.Items.add*).

Sección generar

Una vez hemos creado nuestro DSL y está validado, se habilitará el botón de "Generar". Si pulsamos este botón veremos que aparece un modal que nos solicita la siguiente información (*ver figura 24*):

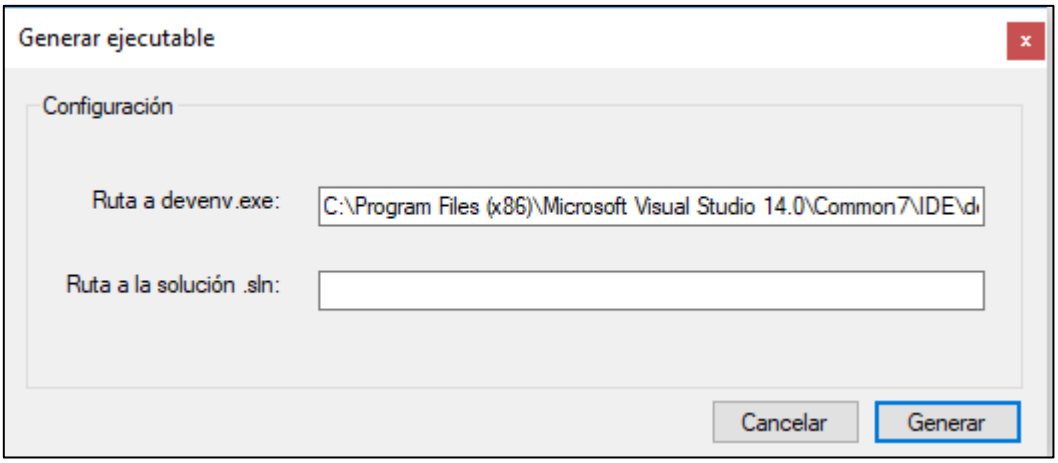


Figura 24 Opciones para la generación del exe

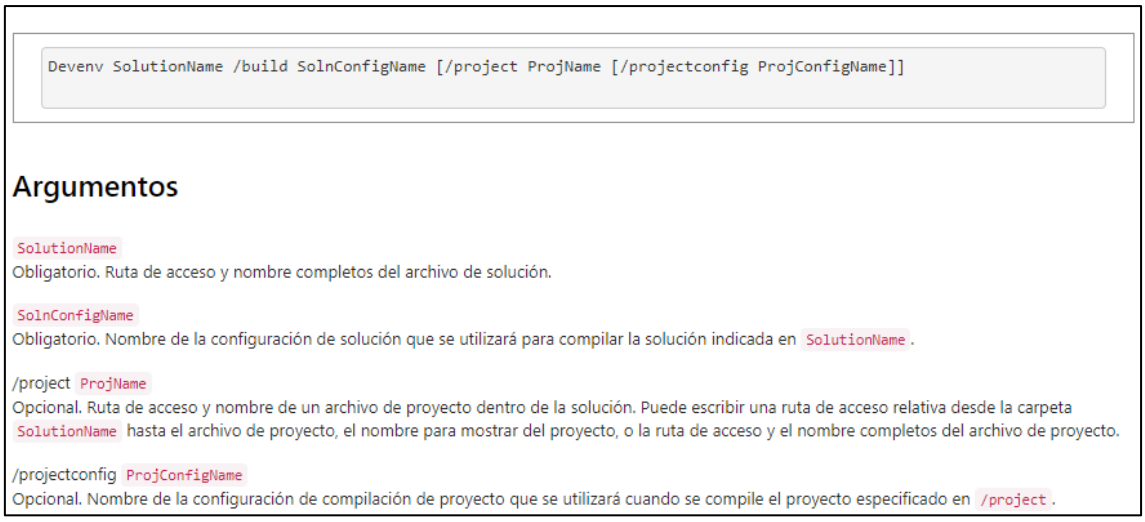
Se nos solicita lo siguiente:

1 Ruta a **devenv.exe**: Devenv.exe es una herramienta de Visual studio que nos va a permitir compilar una solución de visual studio externamente ejecutando el comando. Usaremos este ejecutable para compilar nuestra aplicación y así poner en marcha nuestro DSL. Generalmente este archivo se encuentra en la carpeta de instalación de Visual Studio: *C:\Program Files (x86)\Microsoft Visual Studio 14.0\Common7\IDE\devenv.exe*.
2 Ruta a la **solución .sln:** Aquí estableceremos la ruta al proyecto OCRManager.sln. Este proyecto es el núcleo de nuestra aplicación (*ver sección siguiente*). Es el encargado de interpretar nuestro DSL y generar el código necesario para aplicarlo. Lo

veremos más en detalle en la sección de “Implementación del DSL”.

Con ambas rutas, y pulsando el botón “Generar” podremos **compilar nuestro ejecutable personalizado con el DSL establecido.**

Para ello ejecutaremos en C# el siguiente comando (*ver figura 25*):



```
Devenv SolutionName /build SolnConfigName [/project ProjName [/projectconfig ProjConfigName]]
```

Argumentos

`SolutionName`
Obligatorio. Ruta de acceso y nombre completos del archivo de solución.

`SolnConfigName`
Obligatorio. Nombre de la configuración de solución que se utilizará para compilar la solución indicada en `SolutionName`.

/project `ProjName`
Opcional. Ruta de acceso y nombre de un archivo de proyecto dentro de la solución. Puede escribir una ruta de acceso relativa desde la carpeta `SolutionName` hasta el archivo de proyecto, el nombre para mostrar del proyecto, o la ruta de acceso y el nombre completos del archivo de proyecto.

/projectconfig `ProjConfigName`
Opcional. Nombre de la configuración de compilación de proyecto que se utilizará cuando se compile el proyecto especificado en `/project`.

Figura 25 Compilación externa de una solución de vs [21]

Como comentábamos, en la siguiente sección podremos ver donde localizar nuestro .exe recién compilado y como ejecutarlo. (*ver Implementación y Aplicación práctica*).

4.2 Implementación del proyecto OCR Manager

En esta sección nos vamos a adentrar en el núcleo del proyecto, en la solución que interpreta nuestro DSL y lo aplica. Los objetivos de la solución ***OCRManager.sln*** son los siguientes:

1- Tratar nuestro DSL e interpretarlo.
2- Reconocer imágenes por OCR.
3- Identificar los bloques de texto en las imágenes y catalogarlos en entidades útiles para la lógica de negocio.
4- Aplicar un algoritmo para detectar palabras semejantes.
5- Aplicar un algoritmo para detectar que palabras se encuentran en las regiones estáticas indicadas.
6- Aplicar un algoritmo para detectar que palabras se encuentran en las posiciones relativas a otros campos indicadas.
7- Generar un PDF nombrado con un identificador a partir de una lista de imágenes.
8- Leer códigos de barras en una imagen.
9- Agrupar imágenes que pertenezcan a una misma plantilla.
10- Loguear los pasos realizados por el ejecutable.
11- Mover a fallidos los archivos que no se han podido tratar
12- Mover a las carpetas correspondientes los archivos tratados con éxito.
13- Evitar problemas de memoria asociados a la lectura OCR y la gestión de ficheros.

La estructura genérica de la aplicación la reflejamos en la figura 26:

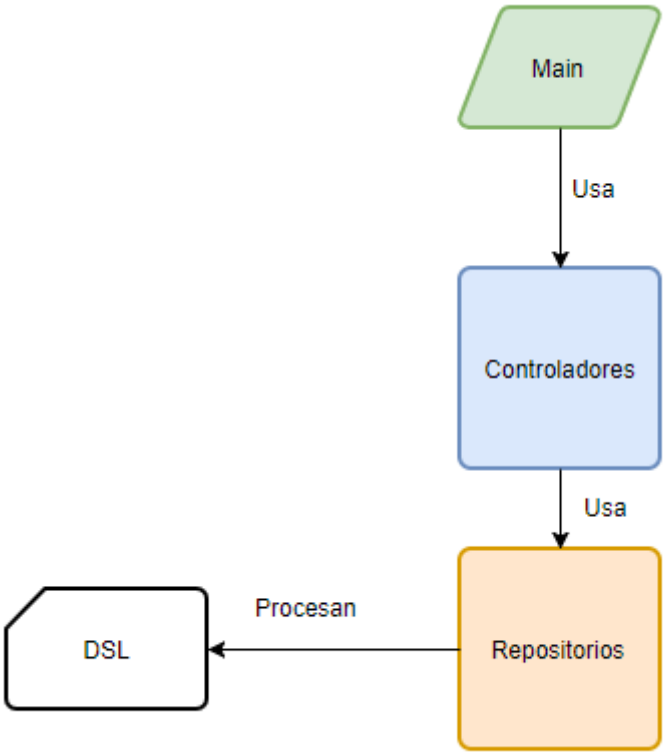


Figura 26 Estructura genérica de la aplicación

Estamos ante una arquitectura de 3 capas. Dispondremos de un **punto principal de ejecución** que usará **controladores** para controlar toda la lógica de la aplicación. Esta lógica necesitara usar **repositorios** para obtener la información de nuestro DSL particular. En la comunicación entre capas usaremos **entidades.** Las entidades contienen todos aquellos objetos (*clases*) que representan al negocio.

Si ampliamos esta imagen de nuestra aplicación e incluimos los archivos de nuestro proyecto involucrados tendremos (*ver figura 27*):

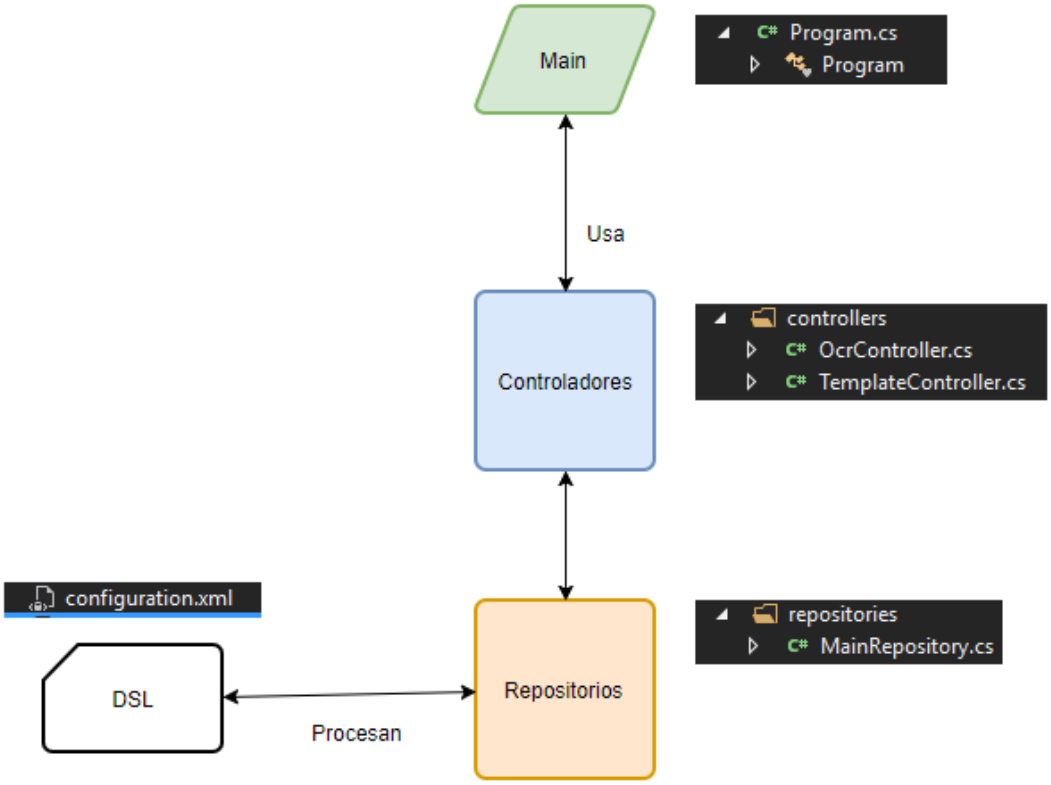


Figura 27 Estructura genérica de la aplicación y archivos del proyecto

Como vemos en la figura 27 tenemos dos controladores, OcrController y TemplateController:

- OcrController: Este controlador es el encargado de trabajar con el motor OCR de Tesseract. Sus funciones principales son iniciar el motor, obtener los resultados textuales categorizados, ubicar las posiciones y obtener los valores relativos.
- TemplateController: Este controlador es el encargado de obtener la información de nuestro lenguaje específico de domino.

Siguiendo la figura 27, tenemos el repositorio MainRepository. Este repositorio es el encargado de procesar directamente nuestro XML (**configuration.xml**), es decir, nuestro lenguaje especifico de domino para así obtener toda la información organizada de forma que resulte útil para la lógica del controlador.

Por último, main, nuestro punto de entrada, lo consituye el archivo Program.cs. Este archivo utilizará los controladores OcrController y

TemplateController para poner en marcha nuestra aplicación y, tras reconocer un conjunto de imágenes, aplicar el proceso sobre ellas y obtener los PDF's agrupados y renombrados.

A continuación, entraremos en detalles de la implementación de cada uno de los elementos descritos por este orden: entidades, repositorios, controladores y punto de entrada.

4.3 Entidades

El proyecto consta de un conjunto de 7 clases que representan la lógica de negocio (*ver figura 28*):

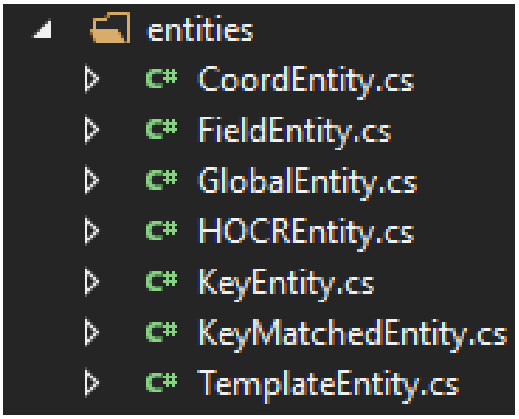


Figura 28 Entidades de la aplicación

Cada entidad tendrá los atributos necesarios junto con sus *getters* y *setters*. Repasemos cada una de ellas:

GlobalEntity: En esta entidad guardamos la información sobre la ruta de entrada y la ruta de fallidos de nuestra aplicación.

```
public class GlobalEntity
{
    private string entryPath = "";
    private string failedPath = "";

    public string EntryPath...

    public string FailedPath...
}
```

KeyEntity: Esta entidad contiene la información sobre los campos que vamos a utilizar para identificar la plantilla. Necesitaremos por tanto almacenar el valor que queremos encontrar y el id para que posteriormente se puede referenciar a él (*recordemos, <fieldHeader> necesita un campo <key> de referencia*).

```
public class KeyEntity
{
    private string id = "";
    private string value = "";
    private int startx = 0;
    private int endx = 0;
    private int starty = 0;
    private int endy = 0;

    public int Startx...

    public int Endx...

    public int Starty...

    public int Endy...

    public string Value...

    public string Id...
```

HOCREntity: Esta entidad contiene la información necesaria para cada ítem reconocido por el motor OCR, es decir, su ubicación dentro de la imagen, ancho y alto del rectángulo que forma la cadena leída por OCR y el valor de dicha cadena.

```
public class HOCREntity
{
    private int startx = 0;
    private int starty = 0;
    private int endx = 0;
    private int endy = 0;
    private int height = 0;
    private int width = 0;
    private string word = "";

    public string Word...

    public int Startx...

    public int Starty...

    public int Endx...

    public int Endy...

    public int Height...

    public int Width...
}
```

CoordEntity: Esta entidad contiene la información posicional de referencia. Es decir, si está arriba, abajo, izquierda o derecha del segmento rectangular superior-inferior-izquierda-derecha y a cuantos pixeles.

```
public class CoordEntity
{
    private string position = "";
    private string pixels = "";
    private string reference = "";

    public string Pixels...

    public string Position...

    public string Reference...
}
```

FieldEntity: Esta entidad almacena los campos para identificar los valores deseados por OCR. Por tanto, almacena la expresión regular y el tipo de código de barras o bien el valor de la cabecera y el de la posición a buscar (*haciendo uso de la entidad CoordEntity descrita anteriormente*).

```
public class FieldEntity
{
    private string recognizementType = "";
    private string regExp = "";
    private string barcodeType = "";
    private string headerValue = "";
    private string headerFieldRef = "";
    private List<CoordEntity> headerPosition = new List<CoordEntity>();
    private List<CoordEntity> fieldValuePosition = new List<CoordEntity>();

    public string RecognizementType...

    public string RegExp...

    public string HeaderValue...

    public string HeaderFieldRef...

    public List<CoordEntity> FieldValuePosition...

    public List<CoordEntity> HeaderPosition...

    public string BarcodeType...
}
```

TemplateEntity: Esta entidad guarda la información para una plantilla especificada en nuestro DSL. Hace uso de las entidades KeyEntity (*para guardar las claves utilizadas para identificar la plantilla*) y FieldEntity (*para posicionar los valores necesarios por OCR*). Almacena además el nombre de la plantilla y la ruta de salida.

```
public class TemplateEntity
{
    private string outputPath = "";
    private string name = "";

    //String array list that contains key elements to identify the template
    private List<KeyEntity> keys = new List<KeyEntity>();

    //Template key field to be exported as text using tesseract ocr
    private FieldEntity keyField = new FieldEntity();

    public string OutputPath...

    public string Name...

    public List<KeyEntity> Keys...

    public FieldEntity KeyField...
}
```

KeyMatchedEntity: Esta entidad se usará para almacenar toda la información cuando una plantilla ha sido enlazada con una imagen, es decir, cuando hemos sido capaces de decir "esta imagen pertenece a esta plantilla y hemos obtenido con éxito el valor identificativo". Hace uso por tanto de HOCREntity, TemplateEntity y KeyEntity.

```
public class KeyMatchedEntity
{
    private HOCREntity hocrEntity = new HOCREntity();
    private KeyEntity keyEntity = new KeyEntity();
    private TemplateEntity templateEntity = new TemplateEntity();
    private string barcodeValue = "";

    public HOCREntity HocrEntity...

    public KeyEntity KeyEntity...

    public TemplateEntity TemplateEntity...

    public string BarcodeValue...
}
```

4.4 Repositorio

El repositorio es el encargado de obtener la información de nuestro DSL y transformarla en entidades de negocio. Para ello disponemos de los siguientes métodos:

```
private string config = "";

public MainRepository(string _config)
{
    this.config = _config;
}

public GlobalEntity getGlobalConfiguration()...

public ArrayList getTemplates()...

private List<KeyEntity> getKeys(XmlNode keyNodes)...

private FieldEntity getBarcodeField(XmlNode parentNode)...

private FieldEntity getKeyField(XmlNode parentNode, XmlNode templateNode)...

private List<CoordEntity> getPositions(XmlNode parentNode)...

private CoordEntity getCoord(XmlNode parentNode, string type)...
```

Cada uno de estos métodos extrae la información requerida de nuestro DSL. Veamos el método **getTemplates**() que contiene el núcleo del código y nos ayudará a comprender el funcionamiento general de este.

El objetivo de este método es obtener una lista con toda la información de cada una de las plantillas establecidas en nuestro DSL.

El constructor del repositorio recibe el XML de nuestro DSL. Primero, por tanto, debemos cargarlo usando la clase XMLDocument:

```
XmlDocument doc = new XmlDocument();
doc.LoadXml(this.config);
XmlNodeList templateNodes = doc.DocumentElement.SelectNodes("template");
```

Una vez cargado el XML, seleccionamos todos los nodos del tipo "template", es decir, todas las plantillas de nuestro DSL y lo guardamos en la colección templateNodes, que es un objeto del tipo XmlNodeList.

Ahora, para cada nodo de la plantilla, vamos a extraer la información en entidades de negocio. Recorremos cada template node y extraemos la información común a todos ellos, que no es otra que el nombre de la plantilla, la ruta de salida y el tipo de reconocimiento:

```
foreach (XmlNode templateNode in templateNodes)
{
    //Read template generic fields
    TemplateEntity templateEntity = new TemplateEntity();
    templateEntity.OutputPath = templateNode.SelectSingleNode("outputPath").InnerText;
    templateEntity.Name = templateNode.SelectSingleNode("name").InnerText;
    string recognizementType = templateNode.SelectSingleNode("recognizementType").InnerText;
```

Ahora, dependiendo del tipo de reconocimiento, debemos extraer los valores asociados al reconocimiento por código de barras o por campo clave.

```
switch (recognizementType)
{
    case "barcodeFirstPage":
        templateEntity.KeyField = getBarcodeField(templateNode);
        templateEntity.KeyField.RecognizementType = recognizementType;
        break;

    case "keyField":
        //Read keys to identify the template
        XmlNode keys = templateNode.SelectSingleNode("templateIdentification");
        templateEntity.Keys = getKeys(keys);
        XmlNode fieldHeader = templateNode.SelectSingleNode("fieldHeader");
        templateEntity.KeyField = getKeyField(fieldHeader, templateNode);
        templateEntity.KeyField.RecognizementType = recognizementType;
        break;
}

templateList.Add(templateEntity);
```

Como vemos, dividimos ambos casos. En el primero tenemos que extraer la información del código de barras (expresión regular y tipo del código de barras) apoyándonos del método *getBarcodeField*().

```
private FieldEntity getBarcodeField(XmlNode parentNode)
{
    FieldEntity fieldEntity = new FieldEntity();
    fieldEntity.RegExp = parentNode.SelectSingleNode("regExp").InnerText;
    fieldEntity.BarcodeType = parentNode.SelectSingleNode("barcodeType").InnerText;
    return fieldEntity;
}
```

En el segundo caso, necesitaremos más métodos helper que nos ayuden a obtener toda la información relativa a la detección de campos clave.

```
  case "keyField":
      //Read keys to identify the template
      XmlNode keys = templateNode.SelectSingleNode("templateIdentification");
      templateEntity.Keys = getKeys(keys);
      XmlNode fieldHeader = templateNode.SelectSingleNode("fieldHeader");
      templateEntity.KeyField = getKeyField(fieldHeader, templateNode);
      templateEntity.KeyField.RecognizementType = recognizementType;
      break;
```

Primero obtenemos el nodo <templateIdentification>. Usando el método getKeys obtenemos todas las claves para identificar la plantilla actual. Como vemos se guarda en el atributo Keys, que no es más que una entidad KeyEntity (*ver entidades*).

```
private List<KeyEntity> getKeys(XmlNode keyNodes)
{
    List<KeyEntity> keyList = new List<KeyEntity>();

    foreach (XmlNode keyNode in keyNodes)
    {
        KeyEntity keyEntity = new KeyEntity();

        keyEntity.Value = keyNode.InnerText;
        string[] coords = keyNode.Attributes["region"].InnerText.Split(',');

        keyEntity.Startx = Int32.Parse(coords[0]);
        keyEntity.Endx = Int32.Parse(coords[1]);
        keyEntity.Starty = Int32.Parse(coords[2]);
        keyEntity.Endy = Int32.Parse(coords[3]);
        keyEntity.Id = keyNode.Attributes["id"].InnerText;

        keyList.Add(keyEntity);
    }

    return keyList;
}
```

Después tenemos que obtener la cabecera y el valor a encontrar. Mediante el método getKeyField() obtenemos la información necesaria y la guardamos en el atributo keyField, que es una entidad FieldEntity.

```
private FieldEntity getKeyField(XmlNode parentNode, XmlNode templateNode)
{
    FieldEntity fieldEntity = new FieldEntity();
    fieldEntity.HeaderValue = parentNode.SelectSingleNode("value").InnerText;
    fieldEntity.HeaderFieldRef = parentNode.SelectSingleNode("fieldRef").InnerText;
    fieldEntity.HeaderPosition = getPositions(parentNode.SelectSingleNode("position"));
    fieldEntity.FieldValuePosition = getPositions(templateNode.SelectSingleNode("fieldValue"));
    return fieldEntity;
}
```

En este último método, getKeyField, vemos que para obtener las posiciones de la cabecera y el valor final se hace uso del método getPositions. Este método es el encargado de procesar los nodos de tipo <posición> para obtener las entidades de tipo CoordEntity:

```
private List<CoordEntity> getPositions(XmlNode parentNode)
{
    List<CoordEntity> positions = new List<CoordEntity>();

    ArrayList typeList =
        new ArrayList(new string[] { "above", "below", "right", "left"});

    foreach(string type in typeList)
    {
        CoordEntity currentCoord = getCoord(parentNode, type);

        if(currentCoord.Pixels != "" && currentCoord.Reference != "")
        {
            currentCoord.Position = type;
            positions.Add(currentCoord);
        }
    }

    return positions;
}
```

Por último, la plantilla actual la incluimos en la lista, de forma que cuando termine el proceso, tengamos un ArrayList con todas las plantillas de nuestro DSL.

```
templateList.Add(templateEntity);
```

4.5 Controladores

En los controladores se encuentra toda la lógica de nuestra aplicación. Hacemos uso de dos controladores, TemplateController y OcrController. Comenzaremos por el primero dada su sencillez.

TemplateController

Este controlador es el encargado de obtener la información de nuestro lenguaje específico de domino.

```
namespace OCRManager.controllers
{
    public class TemplateController
    {
        private string config = "";
        private MainRepository mainRepository;

        public TemplateController(string _config)
        {
            this.config = _config;
            this.mainRepository = new MainRepository(this.config);
        }

        public ArrayList getTemplates()
        {
            ArrayList templates = this.mainRepository.getTemplates();
            return templates;
        }

        public GlobalEntity getGlobalConfiguration()
        {
            GlobalEntity globalEntity = this.mainRepository.getGlobalConfiguration();
            return globalEntity;
        }
    }
}
```

Este es el controlador, ¿sencillo verdad? Consta un constructor que recibe nuestro DSL y dos métodos, getTemplates y getGlobalConfiguration. Como detallamos en la Figura 26, los controladores hacen uso de los repositorios para obtener la información de nuestro DSL. Dado que este controlador carece de lógica tan solo tiene que instanciar nuestro repositorio y llamada a los métodos correspondientes para obtener la lista de plantillas y la GlobalEntity, que contiene el path de entrada de los documentos y la ruta en caso de fallidos.

OcrController

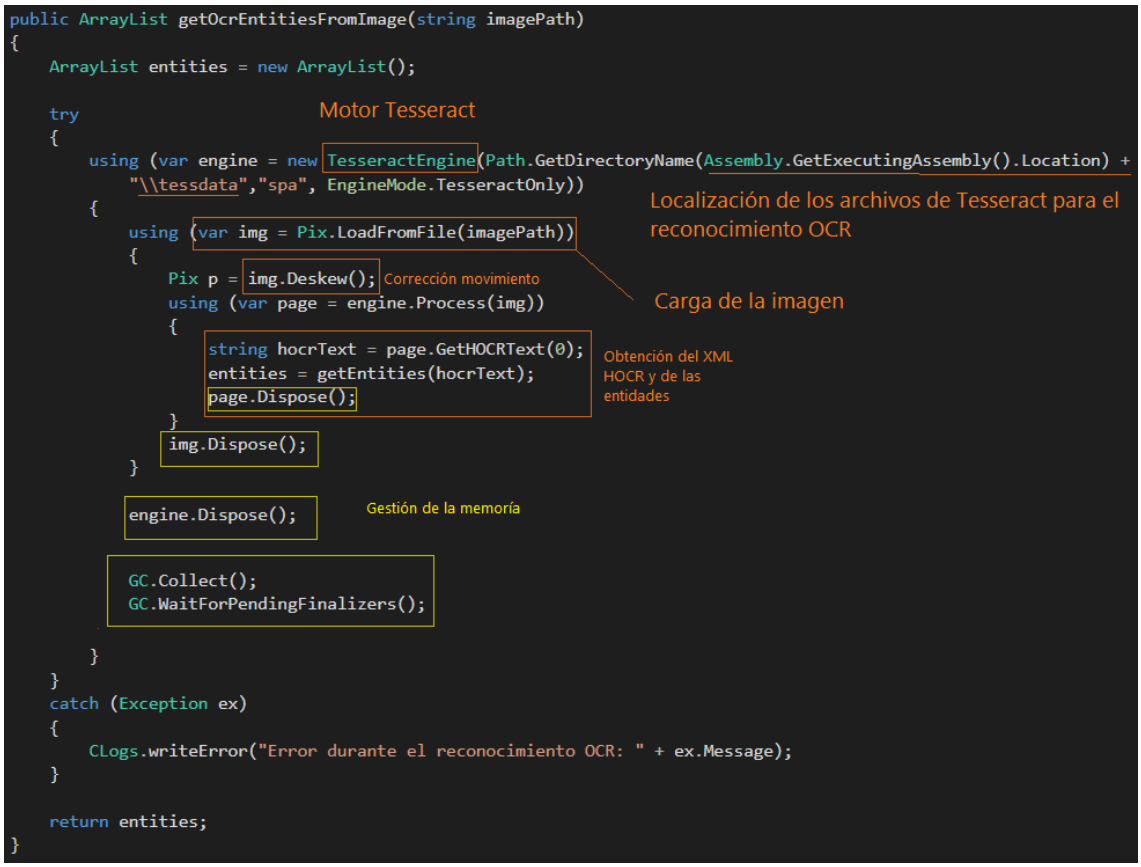
Este controlador es el encargado de trabajar con el motor OCR de Tesseract. Sus funciones principales son iniciar el motor, obtener los resultados textuales categorizados, ubicar las posiciones y obtener los valores relativos.

Este controlador es bastante más extenso que el anterior, contiene la lógica de la aplicación y además hace uso de las librerías para el reconocimiento óptico de caracteres. Veamos pues la lista de métodos de este controlador y entremos en detalle:

```
public class OcrController
{
    public OcrController()
    {}

    public ArrayList getOcrEntitiesFromImage(string imagePath)...

    private HOCREntity getEntity(string text, string coordinates)...

    private string removeStrangeChars(string str)...

    private bool checkText(string text)...

    private ArrayList getEntities(string hocrFile)...

    public HOCREntity getHeaderEntity(HOCREntity fieldEntity, ArrayList fields, TemplateEntity template)...

    public string getFieldValue(HOCREntity fieldEntity, ArrayList fields, TemplateEntity template)...

    private bool checkConditions(HOCREntity fieldEntity, HOCREntity currentEntity, IDictionary<string, string> conditions)...

    private static int CalcLevenshteinDistance(string a, string b)...

    public KeyMatchedEntity applyTemplates(ArrayList ocrEntities, ArrayList templates, string file)...

    private HOCREntity findKey(KeyEntity keyEntity, ArrayList ocrEntities)...
}
```

El método **getOcrEntitiesFromImage** es el encargado de aplicar el reconocimiento OCR a la imagen pasada en el parámetro imagePath (*string con la ruta a la imagen*). El resultado del reconocimiento OCR debe de ser una lista de entidades HOCREntity.

Veamos el método en la figura 28 con explicaciones añadidas:




```
public ArrayList getOcrEntitiesFromImage(string imagePath)
{
    ArrayList entities = new ArrayList();

    try
    {
        using (var engine = new TesseractEngine(Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location) +
            "\\tessdata","spa", EngineMode.TesseractOnly))
        {
            using (var img = Pix.LoadFromFile(imagePath))
            {
                Pix p = img.Deskew();
                using (var page = engine.Process(img))
                {
                    string hocrText = page.GetHOCRText(0);
                    entities = getEntities(hocrText);
                    page.Dispose();
                }
                img.Dispose();
            }

            engine.Dispose();

            GC.Collect();
            GC.WaitForPendingFinalizers();

        }
    }
    catch (Exception ex)
    {
        CLogs.writeError("Error durante el reconocimiento OCR: " + ex.Message);
    }

    return entities;
}
```

Figura 28 Método de reconocimiento OCR

El método comienza con la inicialización del motor de Tesseract, proveído por la libraría de Tesseract wrapper for .Net. En concreto el constructor de clase TesseractEngine requiere de los siguientes parámetros:

- Ruta a la carpeta Tessdata. Esta carpeta contiene los archivos necesarios para que Tesseract aplique el reconocimiento óptico.
- Lenguaje empleado en la imagen. Indicamos "spa" para decirle a Tesseract que el texto será en castellano.
- Modo del motor. Existen varias opciones aquí, pero en este proyecto vamos a usar únicamente el potencial del motor de Tesseract.

Una vez inicializado el motor cargamos la imagen mediante la clase Pix de .NET y el método LoadFromFile. Previo reconocimiento del OCR aplicamos el método Deskew [22] que corrige la inclinación y el movimiento de la imagen.

A continuación, el método process del objeto engine preparará el motor para el procesamiento de la imagen. Usamos el método GetHOCRText para obtener el XML con las entidades textuales del documento. Dado que el concepto de HOCR es nuevo, vamos a explicarlo en el siguiente punto.

HOCR

HOCR [23] es un estándar abierto de representación de datos para el texto formateado obtenido a través del reconocimiento óptico de caracteres (OCR). La definición codifica texto, estilo, información de diseño, métricas de confianza de reconocimiento y otra información utilizando XML en forma de HTML o XHTML.

Un ejemplo de HOCR sería el siguiente mostrado en la figura 29:

```
...
<p class='ocr_par' lang='deu' title="bbox930">
  <span class='ocr_line' title="bbox 348 797 1482 838; baseline -0.009 -6">
    <span class='ocrx_word' title='bbox 348 805 402 832; x_wconf 93'>Die</span>
    <span class='ocrx_word' title='bbox 421 804 697 832; x_wconf 90'>Darlehenssumme</span>
    <span class='ocrx_word' title='bbox 717 803 755 831; x_wconf 96'>ist</span>
    <span class='ocrx_word' title='bbox 773 803 802 831; x_wconf 96'>in</span>
    <span class='ocrx_word' title='bbox 821 803 917 830; x_wconf 96'>ihrem</span>
    <span class='ocrx_word' title='bbox 935 799 1180 838; x_wconf 95'>ursprünglichen</span>
    <span class='ocrx_word' title='bbox 1199 797 1343 832; x_wconf 95'>Umfange</span>
    <span class='ocrx_word' title='bbox 1362 805 1399 823; x_wconf 95'>zu</span>
    <span class='ocrx_word' title='bbox 1417 x_wconf 96'>ver-</span>
  </span>
  ...
```

Figura 29 Ejemplo de HOCR

Podemos observar los atributos de tipo **bbox 348 797 1472 832**. Son precisamente estos elementos los que describen las coordenadas del texto encontrado para ese documento, de ahí que nuestro DSL use el concepto de startx, starty, endx, endy para ubicar el texto.

Continuando con la explicación del método **getOcrEntitiesFromImage,** necesitamos pues procesar este HOCR en formato HTML y convertirlo a una entidad del tipo HOCREntity. De ello se encarga el método **getEntities**.

```
private ArrayList getEntities(string hocrFile)
{
    ArrayList entities = new ArrayList();

    XmlDocument doc = new XmlDocument();
    doc.XmlResolver = null;
    doc.LoadXml(hocrFile);

    XmlNodeList wordList = doc.SelectNodes("//span[@class = 'ocrx_word']");
    foreach (XmlNode row in wordList)
    {
        string text = row.InnerText;
        string coordinates = "";

        if (checkText(text))
        {
            coordinates = row.Attributes["title"].InnerText;
            entities.Add(this.getEntity(text, coordinates));
        }
    }

    return entities;
}
```

Como vemos el método getEntities selecciona los nodos cuya clase es "ocrx_word". "Ocrx_word" denota las palabras encontradas en la imagen. Una vez seleccionados, se recorren y se almacenan en entidades de negocio, junto con su ubicación y su valor.

Precisamente observamos el uso del método **getEntity**. Este método obtiene las coordenadas de la posición para guardar en la entidad HOCREntity las coordenadas, el ancho, el alto y el valor de la palabra.

```
private HOCREntity getEntity(string text, string coordinates)
{
    HOCREntity hocrEntity = new HOCREntity();
    hocrEntity.Word = text.Trim();

    coordinates = coordinates.Replace(";", " ");
    string[] coordinatesSplit = coordinates.Split(' ');

    //10 13 43 46 startx, starty, endx, endy
    //width = endx - startx
    //height = endy - starty
    hocrEntity.Startx = Int32.Parse(coordinatesSplit[1]);
    hocrEntity.Starty = Int32.Parse(coordinatesSplit[2]);
    hocrEntity.Endx = Int32.Parse(coordinatesSplit[3]);
    hocrEntity.Endy = Int32.Parse(coordinatesSplit[4]);
    hocrEntity.Width = Int32.Parse(coordinatesSplit[3]) - Int32.Parse(coordinatesSplit[1]);
    hocrEntity.Height = Int32.Parse(coordinatesSplit[4]) - Int32.Parse(coordinatesSplit[2]);

    return hocrEntity;
}
```

El valor del ancho (*width*) y del alto (*height*) son fácilmente calculables a partir de las coordenadas <startx, starty, endx, endy>.

Sigamos con el controlador OcrController. Tenemos dos métodos helper, **removeStrangeChars** y **checkText** cuya única función es formatear el texto, eliminar espacios y caracteres extraños.

```
private string removeStrangeChars(string str)
{
    Regex rgx = new Regex("[^a-zA-Z0-9 -]");
    str = rgx.Replace(str, "");
    return str;
}

private bool checkText(string text)
{
    if (text.Trim() == "")
    {
        return false;
    }

    if (removeStrangeChars(text) == "")
    {
        return false;
    }

    return true;
}
```

El método **getHeaderEntity** es el encargado de obtener la información para localizar la cabecera de la plantilla y devolver las entidades de negocio asociadas.

```
public HOCREntity getHeaderEntity(HOCREntity fieldEntity, ArrayList fields, TemplateEntity template)
{
    HOCREntity headerEntity = null;

    FieldEntity keyField = template.KeyField;
    IDictionary<string, string> conditions = new Dictionary<string, string>();
    ArrayList foundValues = new ArrayList();

    foreach (CoordEntity coordEntity in template.KeyField.HeaderPosition)
    {
        conditions.Add(coordEntity.Position + "-" + coordEntity.Reference, coordEntity.Pixels);
    }

    foreach (HOCREntity currentEntity in fields)
    {
        bool allowedResult = checkConditions(fieldEntity, currentEntity, conditions);

        if (allowedResult)
        {
            if(CalcLevenshteinDistance(template.KeyField.HeaderValue.ToLower(), currentEntity.Word.ToLower()) <= 2){
                headerEntity = new HOCREntity();
                headerEntity = currentEntity;
            }

        }
    }

    return headerEntity;
}
```

Agrupación de coordenadas

Comprobación de las condiciones de posicionamiento

Búsqueda de palabras semejantes y que estén dentro de las posiciones establecidas

Figura 30 Método para la obtención del header

El método recibe por parámetros la entidad HOCREntity de la cabecera, la lista de campos reconocidos por OCR y la plantilla que pretendemos usar.

Primero agrupamos las condiciones de la plantilla, de forma que se le puedan pasar al método **checkConditions**. Este método es el que verdaderamente calcular, dadas las especificaciones de posicionamiento, los bloques textuales que pertenecen a dichas posiciones. Para ello se utilizan una serie de cálculos matemáticos que determinan, en un eje de coordenadas cuya unidad son los pixeles, que campos se encuentran en la región proporcionada. Veamos el método checkConditions:

```
private bool checkConditions(HOCREntity fieldEntity, HOCREntity currentEntity, IDictionary<string, string> conditions)
{
    bool allowed = true;

    foreach (KeyValuePair<string, string> condition in conditions)
    {
        switch (condition.Key)
        {
            case "above-top":
                if ((currentEntity.Starty + currentEntity.Height) > (fieldEntity.Starty - Int32.Parse(condition.Value)))
                {
                    allowed = false;
                }

                break;

            case "above-bottom":
                if ((currentEntity.Starty + currentEntity.Height) > (fieldEntity.Endy - Int32.Parse(condition.Value)))
                {
                    allowed = false;
                }
                break;

            case "below-top":
                if (currentEntity.Starty < (fieldEntity.Starty + Int32.Parse(condition.Value)))
                {
                    allowed = false;
                }
                break;

            case "below-bottom":
                if (currentEntity.Starty < (fieldEntity.Endy + Int32.Parse(condition.Value)))
                {
                    allowed = false;
                }
                break;

            case "right-left":
                if (currentEntity.Startx < (fieldEntity.Startx + Int32.Parse(condition.Value)))
                {
                    allowed = false;
                }
                break;

            case "right-right":
                if (currentEntity.Startx < (fieldEntity.Endx + Int32.Parse(condition.Value)))
                {
                    allowed = false;
                }
                break;

            case "left-left":
                if (currentEntity.Startx + currentEntity.Width > (fieldEntity.Startx - Int32.Parse(condition.Value)))
                {
                    return false;
                }
                break;

            case "left-right":
                if (currentEntity.Startx + currentEntity.Width > (fieldEntity.Endx - Int32.Parse(condition.Value)))
                {
                    return false;
                }
                break;
        }

        if (!allowed) break;
    }

    return allowed;
}
```

El objetivo es recorrer a lista de condiciones y verificar si se cumplen para una entidad de HOCR dada bajo los criterios de una plantilla. Si devuelve falso, significará que esta entidad HOCR no cumple las

condiciones, por lo que la descartamos. Si devuelve cierto, significa que **esa entidad HOCR pertenece a la región proporcionada.**

Cada uno de los cases verifican todas las relaciones especificadas en nuestro DSL. Jugamos con las coordenadas <startx, starty, endx, endy> y el ancho y alto de los rectángulos formados para cada palabra encontrar por OCR para determinar si cumple las condiciones de posicionamiento, simplemente comparando los criterios de búsqueda.

Si por ejemplo buscamos un elemento que está arriba de la parte superior de otro elemento (above top), deberemos de cumplir la siguiente condición:

```
case "above-top":
    if ((currentEntity.Starty + currentEntity.Height) > (fieldEntity.Starty - Int32.Parse(condition.Value)))
    {
        allowed = false;
    }

    break;
```

Que no es más que comprobar que el comienzo en el eje y de mi entidad más mi alto debe estar por arriba (>) del alto de la entidad referencia menos los pixeles establecidos en el nodo <px> de nuestro DSL.

Restamos los pixeles porque si decimos "un elemento arriba de la parte superior de otro elemento al menos 50 pixeles", debemos tenerlo en cuenta en la comparación para restar esos 50 pixeles, ya que ir hacia arriba significa disminuir en el eje y. Para clarificar esto adjuntamos la figura 31 para explicar cómo funciona el posicionamiento por pixeles en una imagen:

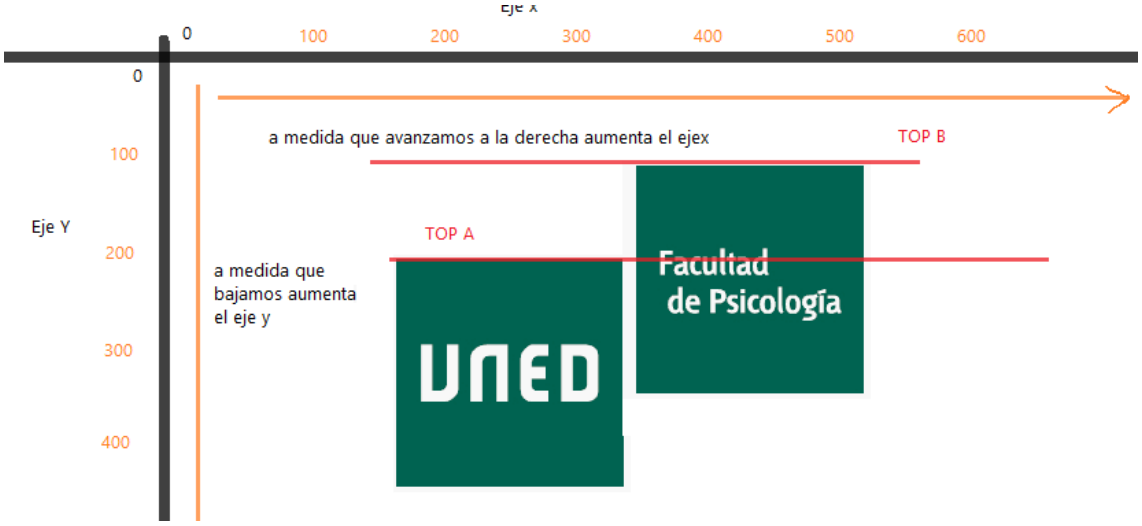


Figura 31 Posicionamiento en una imagen

Como vemos, el cuadrado verde de "Facultad de psicología" esta unos 100 pixeles por arriba del cuadrado de "UNED". Esto significa que si queremos decir que un elemento está por arriba de otros 100 pixeles debemos restar 100 al segmento de arriba del elemento de referencia (TOP A). En esto se basan los casos del método checkConditions.

Continuemos con el método getHeaderEntity. Una vez se han verificado las condiciones y tenemos una entidad HOCR que las cumple tenemos que verificar que el valor para esa "posible" cabecera se parece lo suficiente al valor especificado en el DSL.

Por ejemplo, si queremos obtener la cabecera "Numero de factura": y en el DSL hemos indicado Numero de factura, podemos encontrarnos con que el OCR ha encontrado el texto "Numer0 de f4ctura". Exactamente, se trata de nuestro campo, solo que el OCR no ha sido capaz de leerlo adecuadamente. Para ello utilizamos un algoritmo de Distancia Levenshtein [24] tal y como sigue:

```
private static int CalcLevenshteinDistance(string a, string b)
{
    if (String.IsNullOrEmpty(a) || String.IsNullOrEmpty(b)) return 0;

    int lengthA = a.Length;
    int lengthB = b.Length;
    var distances = new int[lengthA + 1, lengthB + 1];
    for (int i = 0; i <= lengthA; distances[i, 0] = i++) ;
    for (int j = 0; j <= lengthB; distances[0, j] = j++) ;

    for (int i = 1; i <= lengthA; i++)
        for (int j = 1; j <= lengthB; j++)
        {
            int cost = b[j - 1] == a[i - 1] ? 0 : 1;
            distances[i, j] = Math.Min
                (
                Math.Min(distances[i - 1, j] + 1, distances[i, j - 1] + 1),
                distances[i - 1, j - 1] + cost
                );
        }

    return distances[lengthA, lengthB];
}
```

De esta forma podremos descartar las palabras que no se parezcan a la deseada.

El método getFieldValue sigue la misma lógica que el método getHeaderEntity, solo que en vez de buscar el valor para la cabecera trata de buscar el valor clave en base a la cabecera (*recordemos, nodo <fieldValue>, con el que al final renombraremos el PDF resultante*).

Este método por tanto devolverá el valor buscado, por ejemplo, si tenemos ya la cabecera “Numero de factura” y a la derecha, en base a nuestro DSL, se encuentra el valor del número de factura, obtendremos el string con dicho valor. Como vemos, usa también el método checkConditions.

```
public string getFieldValue(HOCREntity fieldEntity, ArrayList fields, TemplateEntity template)
{
    string value = "";
    IDictionary<string, string> conditions = new Dictionary<string, string>();
    ArrayList foundValues = new ArrayList();

    foreach (CoordEntity coordEntity in template.KeyField.FieldValuePosition)
    {
        conditions.Add(coordEntity.Position + "-" + coordEntity.Reference, coordEntity.Pixels);
    }

    foreach(HOCREntity currentEntity in fields)
    {
        bool allowedResult = checkConditions(fieldEntity, currentEntity, conditions);

        if (allowedResult)
        {
            foundValues.Add(currentEntity.Word);
        }
    }

    foreach(string foundValue in foundValues)
    {
        value += foundValue + " ";
    }

    return value;
}
```

El último método importante que dejamos por ver en este controlador es **applyTemplates**. Bien, este método se encarga averiguar si una imagen dada pertenece o no a una plantilla. Recordemos que las plantillas se identifican mediante un campo clave o un código de barras.

Este método necesita, por tanto, recibir por parámetros la lista de entidad HOCR de la imagen, la lista de plantillas y la ruta a la propia imagen. Hay que destacar que para verificar si una imagen pertenece a alguna de nuestras plantillas se sigue un orden secuencial.

Vamos a ver en la figura 32 el método con anotaciones incluidas para mejorar su comprensión:


```
public KeyMatchedEntity applyTemplates(ArrayList ocrEntities, ArrayList templates, string fi
{
    KeyMatchedEntity keyMatchedEntity = null;
    HOCREntity foundEntity = null;

    foreach (TemplateEntity template in templates)
    {
        if(template.KeyField.RecognizementType == "keyField")
        {
            foreach (KeyEntity keyEntity in template.Keys)
            {
                foundEntity = findKey(keyEntity, ocrEntities);

                if (foundEntity == null)
                {
                    keyMatchedEntity = null;
                    break;
                }

                if (template.KeyField.HeaderFieldRef == keyEntity.Id)
                {
                    keyMatchedEntity = new KeyMatchedEntity();
                    keyMatchedEntity.TemplateEntity = template;
                    keyMatchedEntity.KeyEntity = keyEntity;
                    keyMatchedEntity.HocrEntity = foundEntity;
                }
            }

            if (keyMatchedEntity != null)
                break;
        }
        else if(template.KeyField.RecognizementType == "barcodeFirstPage")
        {
            BarcodeReader q = new BarcodeReader();
            q.Options.PossibleFormats = new List<BarcodeFormat>();

            switch (template.KeyField.BarcodeType)
            {
                case "CODE 39":
                    q.Options.PossibleFormats.Add(BarcodeFormat.CODE_39);
                    break;
                case "QR":
                    q.Options.PossibleFormats.Add(BarcodeFormat.QR_CODE);
                    break;
                case "EAN 13":
                    q.Options.PossibleFormats.Add(BarcodeFormat.EAN_13);
                    break;
                case "EAN 8":
                    q.Options.PossibleFormats.Add(BarcodeFormat.EAN_8);
                    break;
            }

            q.Options.TryHarder = true;
            Result result = q.Decode(new Bitmap(file));

            if(result != null)
            {
                string barcodeValue = result.Text;
                Regex rgx = new Regex(template.KeyField.RegExp);
                if (barcodeValue != null && barcodeValue != "" && rgx.IsMatch(@barcodeValue))
                {
                    CLogs.write("El codigo de barras " + barcodeValue + " coincide con la expresión regular");
                    keyMatchedEntity = new KeyMatchedEntity();
                    keyMatchedEntity.TemplateEntity = template;
                    keyMatchedEntity.BarcodeValue = barcodeValue;
                    break;
                }
            }
        }
    }

    return keyMatchedEntity;
}
```

Figura 32 Método applyTemplates

Primero este método recorre las plantillas para tratar de ver si la imagen pertenece a una de ellas. Existen dos casos:

- **La plantilla actual se identifica por código de barras**: En este caso usamos la libraría Zxing y los parámetros de nuestro DSL (tipo de código de barras y expresión regular) para ver si la imagen cumple con nuestro criterio. Para ello utilizamos las clases Regex y BarcodeReader. En caso afirmativo creamos la entidad keyMatchedEntity.
- **La plantilla actual se identifica por campo clave**: En este caso buscamos las claves establecidas en nuestro DSL usando el método **findKey**(). Si el método findKey nos devuelve un valor significa que sea ha localizado el campo clave y creamos la entidad keyMatchedEntity. Para que la plantilla se dé por enlazada todas las claves <key> tienen que hacer match. Por ejemplo, si para una plantilla tenemos como campos identificativos "Factura" y "Telefonica" y solamente encontramos "Telefonica" no consideraremos la plantilla como "matched".

Veamos el método **findKey**, encargado de verificar si existen los campos clave <key> en la imagen:

```
private HOCREntity findKey(KeyEntity keyEntity, ArrayList ocrEntities)
{
    HOCREntity foundEntity = null;
    foreach(HOCREntity hocrEntity in ocrEntities)
    {
        if(hocrEntity.Startx >= keyEntity.Startx && hocrEntity.Endx <= keyEntity.Endx &&
            hocrEntity.Starty >= keyEntity.Starty && hocrEntity.Endy <= keyEntity.Endy)
        {
            string word = hocrEntity.Word + " ";
            if (word != "" && CalcLevenshteinDistance(keyEntity.Value.ToLower(), word.ToLower()) <= 3)
            {
                foundEntity = hocrEntity;
                break;
            }

        }
    }

    return foundEntity;
}
```

Este método básicamente recorre todas las entidades HOCR de la imagen actual y trata de verificar si en la región <región> especificada en nuestro DSL se encuentra una palabra semejante (Levenshtein) a la especificada en el campo <key>.

4.6 Punto de entrada

Vamos a proceder a ver el punto de entrada de nuestra aplicación (*program.cs*) y así comprender como usamos los controladores, repositorios y entidades explicadas, de forma que se entienda el propósito de la aplicación.

El punto de entrada debe encargarse de realizar estas tareas:

- Leer nuestro DSL mediante el controlador TemplateController.
- Obtener las imágenes de la carpeta de entrada.
- Por cada imagen, obtener las entidades HOCR, es decir, usar el método getOcrEntitiesFromImage del controlador ocrController.
- Aplicar las plantillas de nuestro DSL con el método applyTemplates.
- Si primera la imagen pertenece a una plantilla se prepara para ser agrupada en un PDF, sino se mueve a fallidos.
- **En el momento en el que una imagen hace match con una plantilla, se considera que el resto de imágenes forman parte de ese documento hasta que no se hace match con otra plantilla.**
- Una vez terminado el proceso, se deben generar los PDFs en la ruta adecuada con el nombre identificativo.

En las siguientes figuras (33, 34, 35 y 36) explicaremos parte por parte el código del **program.cs**

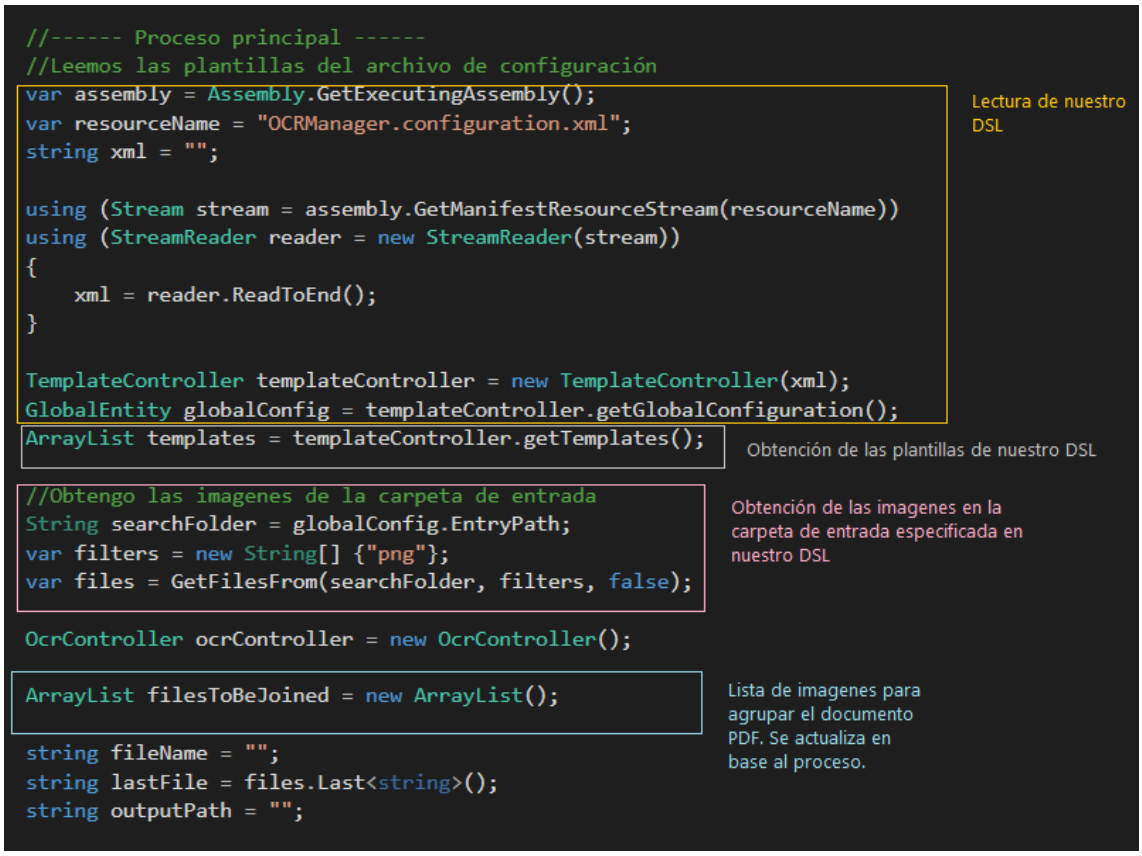


Figura 33 Punto de entrada Parte 1

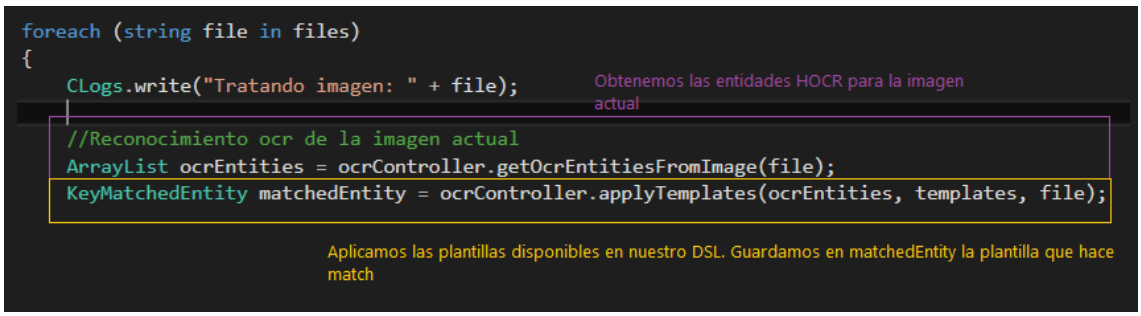


Figura 34 Punto de entrada Parte 2

En las 2 siguientes figuras veremos el caso en el cual se hace match con la plantilla y luego el caso en el que no se hace match.

```
//La siguiente imagen pertenece a una plantilla
if (matchedEntity != null)
{
    if (filesToBeJoined.Count > 0)
    {
        CLogs.write("Creando PDF con nombre: " + fileName + " que tiene " + filesToBeJoined.Count + " páginas");
        CreatePDF(filesToBeJoined, outputPath, fileName);
        filesToBeJoined.Clear();
        fileName = "";
    }

    outputPath = matchedEntity.TemplateEntity.OutputPath;

    if (matchedEntity.TemplateEntity.KeyField.RecognizementType == "barcodeFirstPage")
    {
        CLogs.write("La imagen: " + file + " ha hecho match con la plantilla " + matchedEntity.TemplateEntity.Name);
        filesToBeJoined.Add(file);
        fileName = matchedEntity.BarcodeValue;
        CLogs.write("Valor encontrado: " + fileName); ;
    }
    else
    {
        HOCREntity headerEntity = ocrController.getHeaderEntity(matchedEntity.HocrEntity, ocrEntities, matchedEntity.TemplateEntity);
        CLogs.write("La imagen: " + file + " ha hecho match con la plantilla " + matchedEntity.TemplateEntity.Name);
        filesToBeJoined.Add(file);
        fileName = ocrController.getFieldValue(headerEntity, ocrEntities, matchedEntity.TemplateEntity);
        CLogs.write("Valor encontrado: " + fileName);
    }

    if (file == lastFile)
    {
        CLogs.write("Creando PDF con nombre: " + fileName + " que tiene " + filesToBeJoined.Count + " páginas");
        CreatePDF(filesToBeJoined, outputPath, fileName);
        filesToBeJoined.Clear();
        fileName = "";
    }
}
```

Hemos hecho match con otra plantilla, creo el PDF con las imagenes que teniamos preparadas de la plantilla anterior

Busco el valor para el código de barras

Busco el valor para el campo clave

Si nos encontramos en la ultima imagen y hemos hecho match con la plantilla creamos el ultimo PDF con una sola pagina

Figura 35 Punto de entrada- Se hace match

```
//La siguiente imagen no pertenece a ninguna plantilla
else
{
    CLogs.write("La imagen " + file + " no pertenece a ninguna plantilla");

    if (file == lastFile)
    {
        if (filesToBeJoined.Count > 0)
        {
            filesToBeJoined.Add(file);
            CLogs.write("Creando PDF con nombre: " + fileName + " que tiene " + filesToBeJoined.Count + " páginas");
            CreatePDF(filesToBeJoined, outputPath, fileName);
            filesToBeJoined.Clear();
            fileName = "";
        }
    }
    else
    {
        if (filesToBeJoined.Count > 0)
        {
            filesToBeJoined.Add(file);
        }
        else
        {
            CLogs.write("Muevo a fallidos: " + file);
            File.Move(file, globalConfig.FailedPath + "\\" + Path.GetFileName(file));
        }
    }
}
```

Si es la ultima imagen y teniamos imagenes anteriormente la añado y creo el PDF

No es la ultima imagen por lo que nos limitamos a almacenar la imagen en la lista de archivos para formar un PDF

Si no hacemos match con la plantilla y no tenemos imagenes preparadas, movemos el archivo a fallidos

Figura 36 Punto de entrada- No se hace match

Hemos visto la referencia al método CreatePDF. Este método hace uso de la librería PDFSharp y, dada una lista de imágenes, elabora un PDF mediante la clase PdfDocument:

```
public static void CreatePDF(ArrayList filesToBeJoined, string outputPath, string fileName)
{
    try
    {
        //Creación del documento PDF
        using (PdfDocument doc = new PdfDocument())
        {
            int i = 0;
            foreach (string imageFile in filesToBeJoined)
            {
                int width = 500;
                PdfPage page = doc.AddPage();
                using (XImage img = XImage.FromFile(imageFile))
                {
                    // Calculate new height to keep image ratio
                    var height = (int)(((double)width / (double)img.PixelWidth) * img.PixelHeight);

                    // Change PDF Page size to match image
                    page.Width = width;
                    page.Height = height;

                    XGraphics gfx = XGraphics.FromPdfPage(page);
                    gfx.DrawImage(img, 0, 0, width, height);
                }
                i++;
            }

            doc.Save(outputPath + "\\" + fileName.Trim() + ".pdf");
            doc.Close();
        }

        CLogs.write("PDF guardado correctamente en: " + outputPath + "\\" + fileName + ".pdf");
    }
    catch (Exception ex)
    {
        CLogs.writeError("Error al crear el PDF: " + ex.Message);
    }
}
```

Finalmente, tras ejecutarse el código obtendremos los archivos agrupados en la carpeta de salida en formato PDF y renombrados acorde la especificación de nuestro DSL. Una vez vista la implementación vamos a ver en la siguiente sección un ejemplo práctico.

4.7 Ejemplo práctico de la aplicación

Hemos preparado una serie de imágenes de prueba, facturas concretamente, nombradas SCAN_(Numero) de forma que simule la digitalización (*se ha incluido desvíos en la imagen para ser más realistas)* de un lote de facturas con distinta tipología.

En este lote de facturas se han incluido 3 tipos de facturas:

- Facturas de Transportes Fernandez SL
- Facturas de HolaLuz
- Facturas Proforma

Para cada tipo de factura hemos incluido las siguientes hojas:

- 1 factura de Transportes Fernandez con 4 hojas
- 1 factura de Tranportes Fernandez con 2 hojas
- 1 factura de Tranportes Fernandez con 3 hojas
- 1 factura de HolaLuz con 3 hojas
- 1 factura de HolaLuz con 2 hojas
- 1 factura de HolaLuz con 1 hoja
- 3 facturas de proforma con 1 hoja cada una

Veamos en la figura 37 la lista de imágenes preparadas:

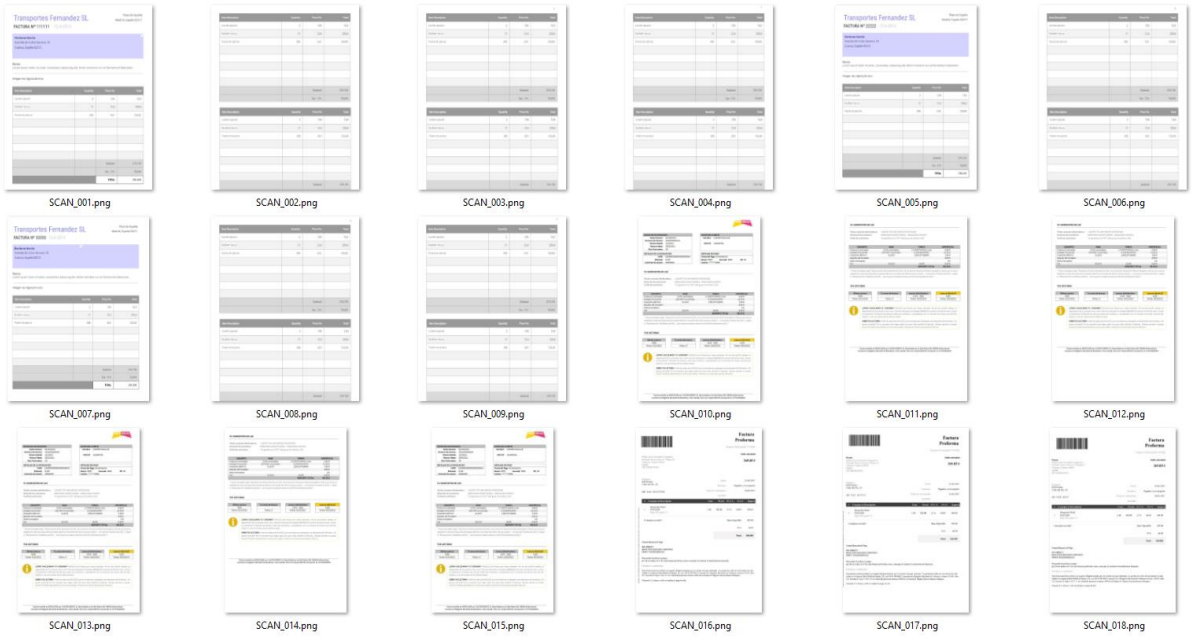


Figura 37 Facturas preparadas

La relación de imágenes png relacionada con cada tipo de factura es la siguiente:

Facturas Fernandez

Factura **Numero 111111**: SCAN_001, SCAN_002, SCAN_003, SCAN_004
Factura **Numero 22222**: SCAN_005, SCAN_006
Factura **Numero 33333:** SCAN_007, SCAN_008, SCAN_009

Facturas HolaLuz

Factura **Numero 20120000001:** SCAN_010, SCAN_011, SCAN_012
Factura **Numero 201200000003**: SCAN_013, SCAN_014
Factura **Numero 2012000009**: SCAN_015

Facturas Proforma (con código de barras)

Facturas SCAN_016 (valor código de barras **LUZ001**), SCAN_017 (valor código de barras **LUZ002**), SCAN_018 (valor código de barras **LUZ003**).

Supongamos que nuestros desarrolladores usando nuestro IDE y validando el XML con la función que habilitamos para ellos han creado un DSL para contemplar estas plantillas. Para poder elaborar este DSL han tenido que examinar las muestras png y determinar cómo identificar las plantillas y, sobre todo, como obtener de forma flexible los campos identificativos correspondientes con los numero de factura.

Tras **media hora de trabajo**, han llegado a desarrollar el siguiente DSL:

```
<root>
  <entryPath>C:\Users\Javier Escribano\Desktop\entry</entryPath>
  <failedPath>C:\Users\Javier Escribano\Desktop\failed</failedPath>
  <template>
      <name>ProformaBarcode</name>
      <outputPath>C:\Users\Javier
Escribano\Desktop\output\proformaBarcode</outputPath>
      <recognizementType>barcodeFirstPage</recognizementType>
      <regExp>LUZ\d*$</regExp>
      <barcodeType>CODE 39</barcodeType>
    </template>

  <template>
      <name>HolaLuz</name>
      <outputPath>C:\Users\Javier Escribano\Desktop\output\holaluz</outputPath>
      <templateIdentification>
        <key id="1" region="660,1400,40,300">Holaluz</key>
      </templateIdentification>
      <recognizementType>keyField</recognizementType>
      <fieldHeader>
        <value>Factura</value>
        <fieldRef>1</fieldRef>
        <position>
          <below>
            <of>bottom</of>
            <pix>3</pix>
          </below>
          <above>
            <of>bottom</of>
            <pix>-40</pix>
          </above>
          <left>
            <of>left</of>
            <pix>200</pix>
          </left>
         <right>
            <of>left</of>
            <pix>-830</pix>
          </right>
        </position>
      </fieldHeader>
      <fieldValue>
        <right>
          <of>right</of>
          <pix>10</pix>
        </right>
        <left>
          <of>right</of>
          <pix>-200</pix>
        </left>
        <above>
          <of>bottom</of>
          <pix>-4</pix>
        </above>
        <below>
          <of>top</of>
          <pix>-4</pix>
        </below>
      </fieldValue>
    </template>
  <template>
    <name>Transportes Fernandez</name>
```

```
    <outputPath>C:\Users\Javier
Escribano\Desktop\output\transportesFernandez</outputPath>
    <templateIdentification>
      <key id="1" region="8,522,26,105">Transportes</key>
      <key id="2" region="8,522,26,105">Fernandez</key>
      <key id="3" region="14,260,90,168">Factura</key>
    </templateIdentification>
    <recognizementType>keyField</recognizementType>
    <fieldHeader>
      <value>Factura</value>
      <fieldRef>1</fieldRef>
      <position>
        <below>
          <of>bottom</of>
          <pix>10</pix>
        </below>
        <above>
          <of>top</of>
          <pix>-100</pix>
        </above>
      </position>
    </fieldHeader>
    <fieldValue>
      <right>
        <of>right</of>
        <pix>10</pix>
      </right>
      <left>
        <of>right</of>
        <pix>-118</pix>
      </left>
      <above>
        <of>bottom</of>
        <pix>-1</pix>
      </above>
      <below>
        <of>top</of>
        <pix>-1</pix>
      </below>
    </fieldValue>
  </template>
</root>
```

Como vemos, este DSL contiene 3 nodos <template>, uno para cada tipo de plantilla. Los desarrolladores de este DSL han establecido las carpetas de entrada y salida. Además, con un editor de imágenes, han establecido las relaciones necesarias para identificar cada plantilla y obtener los valores requeridos.

Una vez listo el DSL, han usado la opción de Generar del IDE que ha compilado la aplicación y ha proporcionado un ejecutable en la carpeta *debug/release* del proyecto *OCRManager.sln*.

¿Que esperamos que realice nuestro ejecutable una vez lo ejecutemos? Bien, el resultado esperado sería que generara 3 carpetas, una por cada <outputPath>, quedando así:

```
C:\Users\Javier Escribano\Desktop\output\transportesFernandez
C:\Users\Javier Escribano\Desktop\output\holaluz
C:\Users\Javier Escribano\Desktop\output\proformaBarcode
```

Una vez creadas las carpetas, esperamos que sea capaz de identificar cada plantilla y, dentro de cada carpeta crear un documento PDF por cada factura (**son 9 en total**) con el nombre de su número de factura correspondiente.

Por ejemplo, si tenemos la factura *Factura **Numero 22222**: SCAN_005, SCAN_006 de Transportes Fernandez* esperamos que en la carpeta **C:\Users\Javier Escribano\Desktop\output\transportesFernandez** se cree un documento PDF que contenga, por orden, las siguientes imágenes: SCAN_005 y SCAN_006. El PDF debe de llamarse 22222.pdf, ya que es el número de la factura. De este modo, tendríamos clasificados e identificados los documentos y podrían seguir el flujo de la gestión documental.

Veamos las capturas del proceso ejecutado en la figura 38:

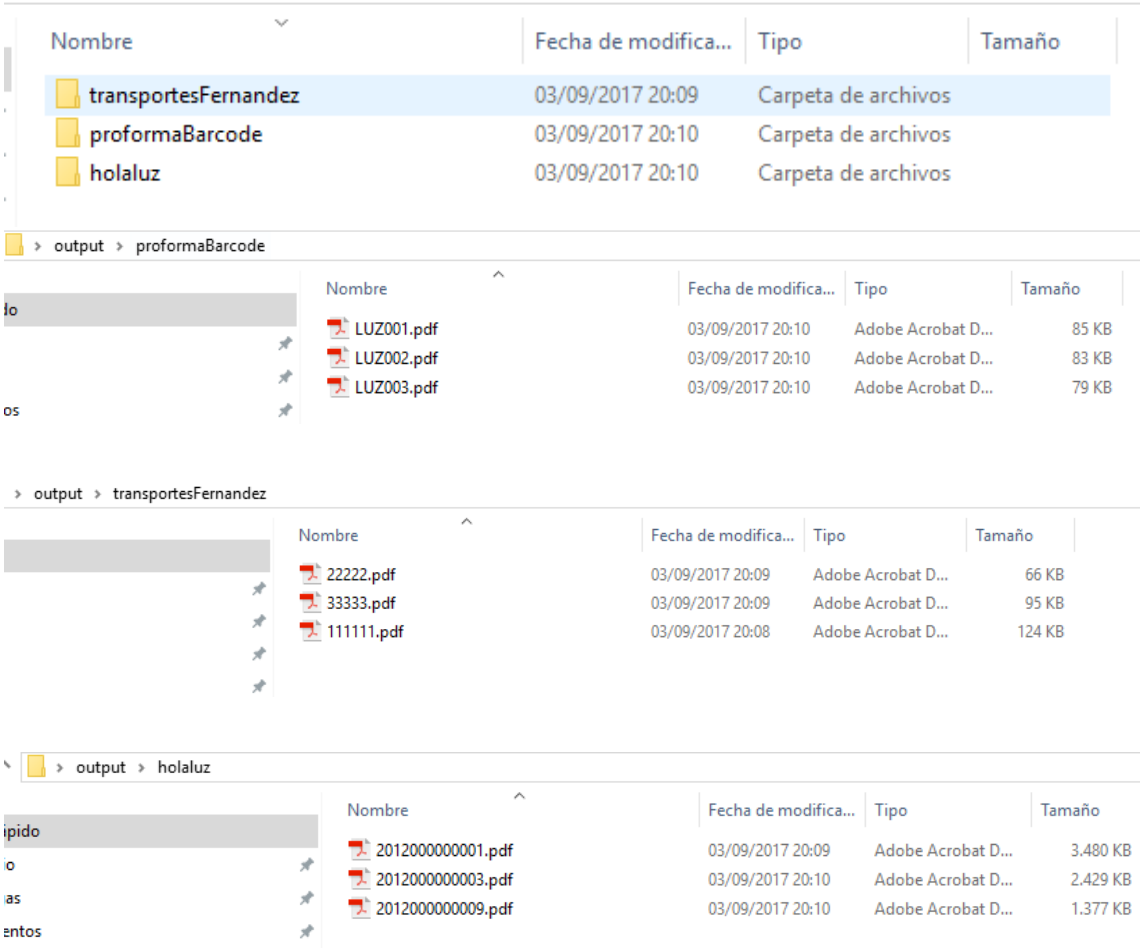


Figura 38 Resultado final

El proceso ha tardado <u>2 minutos y 10 segundos</u>. Efectivamente, nuestro DSL ha sido capaz de procesar las imágenes, agruparlas en sus correspondientes facturas y renombrarlas. Por ejemplo, el PDF 3333.pdf contiene las imágenes SCAN_007, SCAN_008, SCAN_009, como se observa en la figura 39:

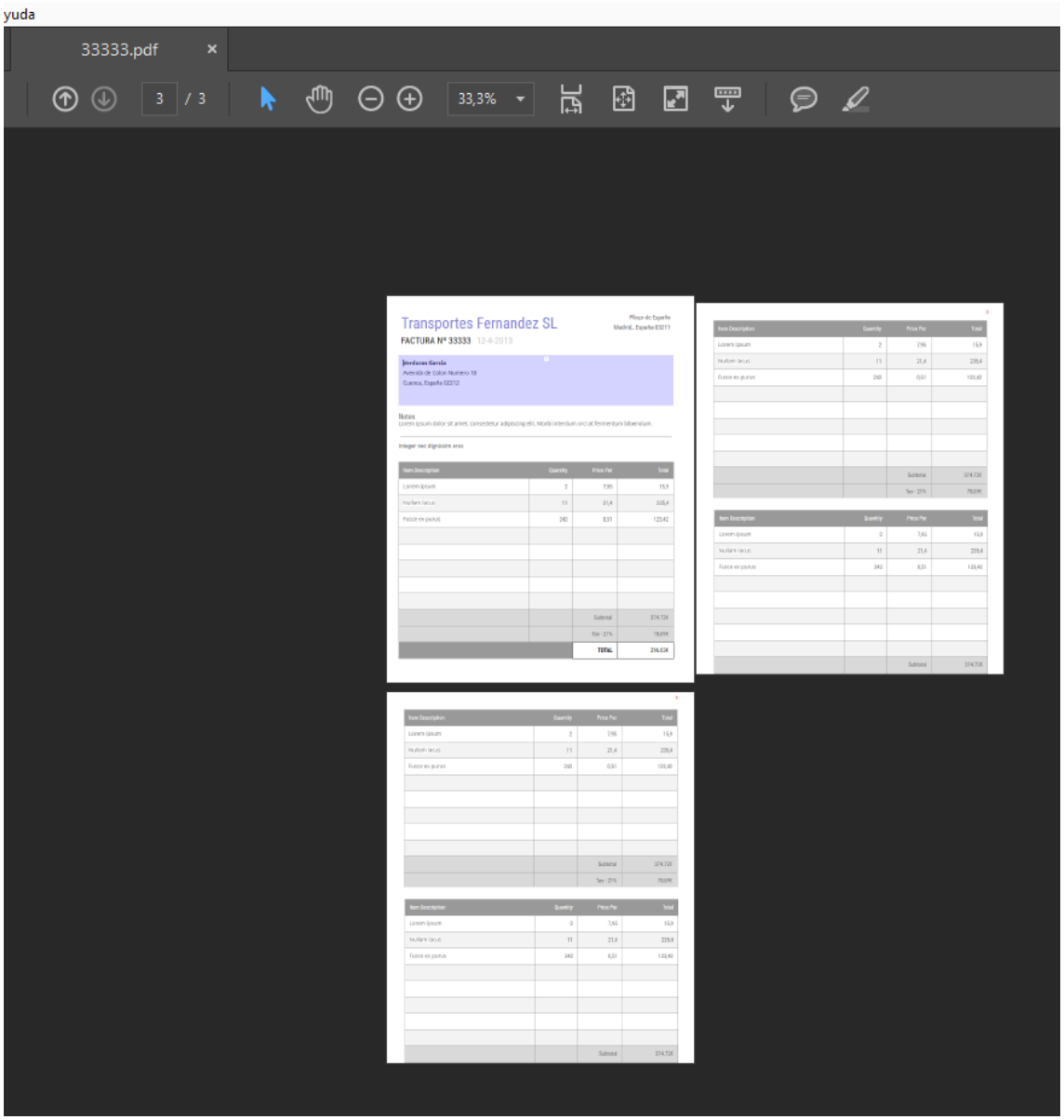


Figura 39 Resultado final - PDF

5 Conclusión y trabajos futuros

Como hemos podido ver en la sección anterior, **hemos conseguido el objetivo de clasificar e identificar una lista de imágenes mediante un lenguaje específico de dominio** en formato XML.

El lenguaje específico de dominio aquí planteado resulta bastante comprensible para desarrolladores que trabajan en la gestión documental ya que están habituados a este tipo de conceptos y planteamientos. El formato XML además permite una compresión visual y directa.

Proporciona un beneficio considerable ampliar Tesseract con este tipo de funcionalidades ya que permite, de una forma relativamente sencilla, disponer de una herramienta que permita mejorar procesos de gestión documental web.

Este proyecto es potencialmente mejorable, pero establece una base sobre la que crecer en funcionalidades. Es bastante **flexible y personalizable**, lo que resulta de gran interés para diversas empresas en las que los propios documentos no siguen un estándar bien establecido o bien necesitan objetivos muy concretos. Evidentemente no alcanza las funcionalidades de Abbyy, ni siquiera realizando las mejoras propuestas en la siguiente sección, pero **puede resultar una opción muy atractiva para empresas que no pueden acceder a usar una solución tan avanzada como Abbyy**, sea por los motivos que sean.

Nos hemos adaptado al mercado de la pequeña y mediana empresa y hemos desarrollado el proyecto bajo herramientas de Microsoft (*ampliamente usadas en este sector*), libres (*Tesseract, ZXing, PDFSharp*) y con una comunidad de usuarios bastante importante detrás. Se ha seguido un estilo de programación bastante aceptable para que sea ampliado y mantenido fácilmente, aunque también es susceptible de ser refactorizado en caso de que crezca la solución.

En la siguiente sección se mencionarán todas las líneas de trabajo futuras para poder mejorar el proyecto aquí expuesto.

Como trabajos futuros, planteamos ampliar el lenguaje específico de dominio aquí expuesto para incluir los siguientes puntos:

1- Mejorar la imagen mediante algoritmos de preprocesamiento y así obtener un mejor resultado en el reconocimiento textual.
2- Leer múltiples campos para obtener más información acerca del documento (*fecha de la factura, numero del proveedor, etc..*).
3- Tratar documentos en otro formato que no sea png.
4- Obtener todo el texto del documento para fines relacionados con la gestión documental (*búsqueda por contenido*).
5- Obtener campos identificativos que no estén en la primera página.
6- Leer códigos de barras en otros formatos a los establecidos.
7- Procesar documentos en distinta lengua al castellano.
8- Crear una interfaz visual que facilite la gestión de las posiciones.
9- Proporcionar un XML de salida personalizado con campos personalizables.
10- Rotar las imágenes automáticamente durante el preprocesamiento para contemplar los casos en los que la imagen venga rotada.
11- Permitir escoger entre si generar un ejecutable o un ejecutable listo para ser un servicio de Windows.
12- Permitir en el IDE realizar un test rápido del lenguaje específico del dominio con un conjunto de imágenes a petición del desarrollador. Este test mostrará los valores que se obtendrían con ese DSL y ese conjunto de imágenes.
13- Incluir más herramientas en el IDE que faciliten la creación/edición del DSL
14- Permitir exportar directamente un fichero HOCR por cada imagen para que terceras aplicaciones puedan integrarse.
15- Incluir un lenguaje que permita la gestión del Training de Tesseract

6 Listado de siglas y referencias

Siglas

OCR	Reconocimiento óptico de caracteres
DPI	Puntos por pulgada
DSL	Lenguaje específico de dominio
ID	Identificador
PDF	Formato de documento portátil
ERP	Planificador de recursos empresariales
PNG	Gráficos de red portables
HP	Hewlett-Packard
PX	Pixel
DB	Base de datos
SDK	Kit de desarrollo software
QA	Aseguramiento de la calidad
IDE	Entorno de desarrollo integrado
VS	Visual Studio (Microsoft)
XML	Lenguaje de marcado extensible
.NET	Framework de Microsoft
IVA	Impuesto sobre el valor añadido
.SLN	Extensión para las soluciones de Visual Studio
HOCR	Estándar para la representación de text obtenido por OCR

Referencias

[1]	D. Berchmans and S. S. Kumar, "Optical character recognition: An overview and an insight," *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, Kanyakumari, 2014, pp. 1361-1365. doi: 10.1109/ICCICCT.2014.6993174
[2]	Gallo, P. R. (2011). Gestión documental en las organizaciones. Editorial UOC.
[3]	Dante, P., & Dante, G. P. (2004). Gestión de la información: dimensiones e implementación para el éxito organizacional. Nuevo Parhadigma,.
[4]	D'ALÒS-MONER, Adela. La gestión documental: aspectos previos a su implementación. El profesional de la información, 2006, vol. 15, no 3, p. 222-226.
[5]	http://digitaldocu.es/es/index.php
[6]	https://en.wikipedia.org/wiki/Comparison_of_optical_character_recognition_software
[7]	SMITH, Ray. An overview of the Tesseract OCR engine. En Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on. IEEE, 2007. p. 629-633.
[8]	https://abbyy.technology/en:start
[9]	https://www.abbyy.com/es-es/finereader/
[10]	https://www.abbyy.com/es-es/flexicapture/
[11]	Smith, R., Antonova, D., & Lee, D. S. (2009, July). Adapting the Tesseract open source OCR engine for multilingual OCR. In Proceedings of the International Workshop on Multilingual OCR (p. 1). ACM.
[12]	https://martinfowler.com/dsl.html
[13]	A. Zisman, "An overview of XML," in *Computing & Control Engineering Journal*, vol. 11, no. 4, pp. 165-167, Aug. 2000. doi: 10.1049/cce:20000405
[14]	https://msdn.microsoft.com/es-es/library/dd30h2yb(v=vs.110).aspx
[15]	https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/
[16]	S. Amann, S. Proksch, S. Nadi and M. Mezini, "A Study of Visual Studio Usage in Practice," *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Suita, 2016, pp. 124-134. doi: 10.1109/SANER.2016.39

[17]	Saeed K. Rahimi; Frank S. Haug, "The Microsoft .NET Platform," in *Distributed Database Management Systems:A Practical Approach* , 1, Wiley-IEEE Press, 2010, pp.663-687 doi: 10.1002/9780470602379.ch18
[18]	https://github.com/charlesw/tesseract
[19]	http://www.pdfsharp.net/
[20]	https://github.com/micjahn/ZXing.Net
[21]	https://msdn.microsoft.com/es-es/library/b20w810z.aspx
[22]	SANSOM-WAI, Cindy Y.; WILLIAMS, Irene H.; TRETTER, Daniel R. Image processing system with image cropping and skew correction. U.S. Patent No 6,310,984, 30 Oct. 2001.
[23]	T. Breuel, "The hOCR Microformat for OCR Workflow and Results," *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, Parana, 2007, pp. 1063-1067. doi: 10.1109/ICDAR.2007.4377078
[24]	L. Yujian and L. Bo, "A Normalized Levenshtein Distance Metric," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1091-1095, June 2007. doi: 10.1109/TPAMI.2007.1078